Matrices and Simplex Algorithms

A. R. G. Heesterman
Department of Economics, University of Birmingham, U.K.

Matrices and Simplex Algorithms

A Textbook in Mathematical Programming and Its Associated Mathematical Topics

D. Reidel Publishing Company
Dordrecht : Holland / Boston : U.S.A. / London : England

Library of Congress Cataloging in Publication Data

CIP

Heesterman, A. R. G.
Matrices and simplex algorithms.

Includes index.
1. Programming (Mathematics) 2. Matrices. I. Title.
QA402.5.H43 1982 519.7 82-18540

ISBN-13:978-94-009-7943-7 e-ISBN-13:978-94-009-7941-3
DOI:10.1007/978-94-009-7941-3

Published by D. Reidel Publishing Company
P.O. Box 17, 3300 AA Dordrecht, Holland

Sold and distributed in the U.S.A. and Canada
by Kluwer Boston Inc.,
190 Old Derby Street, Hingham, MA 02043, U.S.A.

In all other countries, sold and distributed
by Kluwer Academic Publishers Group,
P.O. Box 322, 3300 AH Dordrecht, Holland

D. Reidel Publishing Company is a member of the Kluwer Group

Table of Contents

N.B. Answers to exercises are not included in this list of contents, but may be found in the places indicated with the exercises, usually at the end of the chapters.

Introduction

This is a textbook devoted to mathematical programming algorithms and the mathematics needed to understand such algorithms. It was mainly written for economists, but the mathematics itself obviously has relevance for other disciplines.

It is a textbook as well as, in parts, a contribution to new knowledge. There is, accordingly, a broad ordering of climbing sophistication, the earlier chapters being purely for the student, the later chapters being more specialist and containing some element of novelty on certain points. The book is edited in five parts.

Part I deals with elementary matrix operations, matrix inversion, determinants, etc.

Part II is mainly devoted to linear programming.

As far as students' readability is concerned, these two parts are elementary undergraduate material.

However, I would claim, in particular with respect to linear programming, that I do things more efficiently than the standard textbook approach has it. This refers mainly to the search for a feasible solution i.e. Chapter 9, and to upper and lower limits, i.e. Chapter 10. I have also argued that the standard textbook treatment of degeneracy misses a relevant problem, namely that of accuracy.

In short, I would invite anyone who has the task of writing or designing an LP-code, to first acquaint himself with my ideas.

Parts III and IV are concerned with nonlinear programming. Part III gives the bulk of the theory in general terms including additional matrix algebra. It was obviously necessary to introduce definiteness at this point, but a full discussion of latent roots is refrained from. Proofs are therefore given as far as possible, without reference to eigenvalues. However, certain results will have to be taken on trust by those readers who have no prior knowledge on this point.

The main contribution to the literature made in part III, probably is Chapter 15, i.e. to explain both the first-order conditions and the second order conditions for a constrained maximum, in terms where one may expect the student to actually understand this admittedly difficult problem.

The unconventional concept of subspace convexity is not, and cannot be a true novelty; it is equivalent to the more usual way of formulating the second order condition for a constrained maximum in terms of determinants.

Part IV is concerned with quadratic programming. It does not give a comprehensive survey of algorithms. It gives those algorithms which I considered the most efficient, and the easiest to explain and to be understood.

With respect to novelty, Chapters 16 and 17 do not contain any original ideas or novel approaches, but some of the ideas developed in Chapters 9 and 10 for the LP case are carried over into quadratic programming. Chapter 19 does however, offer an algorithm developed by myself, concerning quadratic programming with quadratic side-conditions.

Part V deals with integer programming.

As in the QP case, the basic ingredients are taken from the existing literature, but the branching algorithm of section 20.2, although based on a well-established approach, was developed by myself. Also, the use of upper and lower limits on the lines of Chapter 10 proved particularly useful in the integer programming context.

Nothing in this book is out of the reach of undergraduate students, but if it is to be read in its entirety by people without prior knowledge beyond "O" level mathematics, the consecutive ordering of the material becomes essential and a two-year period of assimilation with a break between Part II and Part III would be preferable. However Parts IV and V will generally be considered to be too specialist on grounds of relevance and curriculum load, and accordingly be considered more suitable for postgraduate students specializing in O.R. or mathematical programming.

I have, however, myself used some sections of Chapter 16, not so much because undergraduates need to know quadratic programming for its own sake, but as reinforcement of teaching optimality conditions.

Concerning presentation, it may be observed that more than half of the number of pages is taken up by numerical examples, graphical illustrations and text-listings of programme-code.

The numerical examples and graphical illustrations are obviously there for purely educational purposes as are the student exercises.

The code-listings serve, however a dual purpose to back up the descriptions of algorithms, and to be available for computational use. Some tension between these two purposes is obviously unavoidable. I have made an effort to make the programme-texts readable, not only for the machine but also for the human reader, but if illustration for the benefit of the human reader were the only consideration I would have to cut down on the number of pages of code-listing much more than I have in fact done.

It appears to be appropriate to comment here on the use of the computer-language i.e. Algol 60, rather than the more widely used Fortran. While it is true that I simply know Algol very much better than Fortran, the choice appears also to be justified on the following intrinsic grounds:

The use of alphanumerical labels which are meaningful to the human reader, e.g.

PHASE I:, MAKE THE STEP:, etc., and
corresponding goto statements e.g.

'GOTO' PHASE I; helps to bridge the gap between programme description and programme-text in a way which is difficult to achieve by comment (or its Fortran equivalent) only.

Many procedure and programme texts also contain alphanumerical labels which are there purely for the human reader, as there are no corresponding goto statements.

While such labels are also possible in an "Algol-like" language as, for example Pascal, they cannot be used in Fortran. Furthermore, Fortran is simply more primitive than Algol.

I have made extensive use of block-structure and dynamic arrays and as a result my programmes don't require more core-space than is strictly necessary. Also, I gather that not all versions of Fortran permit recursive calls in the way I have used them in section 5.6 for the calculation of determinants and in section 20.4 for branching in integer programming.

The value of the text-listings as a direct source of ready-made programme-text is further compromised by the presence of warning-messages, not only in the main programmes but also inside the procedures.

The presence of these warning messages enhances the readability for the human reader but, as they are system-specific, they will in general require adaptation if the algorithms are to be applied in a different machine-environment.

Acknowledgement

I gladly acknowledge the use of the facilities of the University of Birmingham Computer Center, as well as the help of Dr. P. Soldutos, my student at the time, in setting up the private graph-plotting package used to make the graphs in this book.

University of Birmingham
21st May, 1982

A.R.G. Heesterman

Part I

MATRICES, BLOCK-EQUATIONS, AND DETERMINANTS

CHAPTER III

CHAPTER IV

CHAPTER V

CHAPTER I

EQUATIONS-SYSTEMS AND TABLEAUX

1.1 Equations-systems

The following is an ordinary system of simultaneous linear equations:

$$0.867\,x_1 - 0.066\,x_2 = 240 \qquad (11)$$

$$-0.150\,x_1 + 0.850\,x_2 = 210 \qquad (12)$$

$$-0.167\,x_1 - 0.100\,x_2 + x_3 = 0 \qquad (13)$$

This system can be solved by performing certain operations: Divide (11) by 0.867 (multiply by 1/0.867), and obtain:

$$x_1 - 0.076\,x_2 = 277 \qquad (21)$$

Multiply (21) by 0.150 and 0.167, respectively, and obtain:

$$0.150\,x_1 - 0.011\,x_2 = 42 \qquad (22)$$

$$0.167\,x_1 - 0.013\,x_2 = 46 \qquad (23)$$

Re-name (21) as (31), add (22) to (12) to become (32), add (23) to (13) to become (33), and obtain a new system:

$$x_1 - 0.076\,x_2 = 277 \qquad (31)$$

$$0.839\,x_2 = 252 \qquad (32)$$

$$-0.113\,x_2 + x_3 = 46 \qquad (33)$$

Divide (32) by 0.839 (multiply by 1/0.839) to obtain:

$$x_2 = 300 \qquad (42)$$

Multiply (42) by 0.076 and 0.113, respectively, to obtain:

$$0.076\,x_2 = 23 \qquad (41)$$

$$0.113\,x_2 = 34 \qquad (43)$$

Add (41) to (31), to become (51); re-name (42) as (52); add (43) to (33), to become (53), and obtain a new system:

$$x_1 = 300 \quad (51)$$

$$x_2 = 300 \quad (52)$$

$$x_3 = 80 \quad (53)$$

1.2 The use of a tableau

In section 1.1 three equations were written 5 times. Each time the variable-names x_1, x_2, and x_3 were written again. In total we did this 15 times.

We can economize on our writing effort, by writing the names of the variables only once. And if the procedure for obtaining a system of equations from its predecessor is a standardized one, we can dispense with explaining it every time. We could have done the job by writing 5 tableaux, as listed below:

x_1	x_2	x_3	=
0.867	-0.066	-----	240
-0.150	0.850	-----	210
-0.167	-0.100	1,000	---
1.000	-0.076	-----	277
0.150	-0.011	-----	42
0.167	-0.013	-----	46
1.000	-0.076	-----	277
-----	0.839	-----	252
-----	-0.113	1,000	46
-----	1.000	-----	300
-----	0.076	-----	23
-----	0.113	-----	34
1,000	-----	-----	300
-----	1.000	-----	300
-----	-----	1.000	80

CHAPTER II

MATRIX NOTATION

2.1 The purpose of matrix notation

Matrix notation provides a very compact way of describing certain well-defined numerical operations. As such it saves writing and reading effort in written communication about numerical operations. This applies to communication between one human being and another. It also applies to machine-programming. For most computers, there is by now a certain body of established programmes, routines, carrying out specific matrix- and vector operations. Reference to such routines saves programming effort. The use of matrix notation has also facilitated the analysis of numerical problems. This refers in particular to the properties of linear equation-systems. Such facilitation is really a corollary of the reduction in effort. Problems, which were formerly too complicated to grasp, now become manageable.

2.2 Some definitions and conventions

A matrix is a rectangular grouping of numbers, its elements. The elements of a matrix are grouped into a number of rows; each row containing the same number of elements, reading from left to right. Alternatively, we can say that the elements of a matrix are grouped into a number of columns, each column containing the same number of elements, reading from top to bottom.

Matrices often occur as tableaux, containing statistical information. For example:

British consumer's expenditure, Central Statistical Office: "National Income and Expenditure" (1967),

	1964	1965	1966
Coal and coke	273	269	257
Electricity	381	417	435
Gas	167	194	223
Other	58	60	58

Another application of tableaux (or matrices), was met in section 1.2: the arrangement of computations.

The order parameters of a matrix are two non-negative integer numbers. They are always listed in the same order. The first number indicates the number of rows of the matrix; the second order-parameter indicates the number of columns in the matrix. Normally, order-parameters will be positive numbers. But a more general validity of certain statements in matrix algebra may be obtained, if we admit the value of zero as a borderline case.

The number of elements in each row is the number of columns in the matrix. And the number of elements in each column is the number of rows in the matrix. It follows that the number of elements in the matrix is the product of its order parameters. A matrix of which the two order-parameters are equal is called a square matrix.

To indicate a matrix, we can use a letter. A capital letter is always used for that purpose. In printed text, a capital letter indicating a matrix, is generally given in heavy print.

We might for instance have:

$$C = \begin{bmatrix} 273 & 269 & 257 \\ 381 & 417 & 435 \\ 167 & 194 & 223 \\ 58 & 60 & 58 \end{bmatrix}$$

A corresponding lower case letter, with 2 indices will indicate an individual element of a matrix.
For example, $c_{3,2} = 194$.
The indices are always given in the same order: first the index indicating the row, then the index indicating the column.

Not all numerical information is suitably presented in a rectangular array. Suppose for instance we were interested only in total expenditure on fuel and light:

	1964	1965	1966
Expenditure on fuel and light	879	940	973

This is a vector. A vector is a matrix of which one of the order parameter is known to be unity. We distinguish between rows (matrices with only one row) and columns (matrices with only one column). The (total) fuel and light expenditure, was presented as a row. We could have presented it as a column as well.

To indicate a vector, we can use a letter. For that purpose, one always uses a non-capital letter. Vectors are normally indicated with italic print or heavy print, or in typescript - including photographically reproduced typescript, as in this book, with

underlining, to avoid confusion with indices. One should of course not use capital letters, this would create confusion with matrices.

Vectors will normally be assumed to be columns. When we want to indicate a row, this will be done by adding a prime to the letter.

The order of a vector is determined by listing the value of only one order-parameter. Elements of a vector are indicated with a small (non-capital) letter, without heavy print or underlining, with one index.

For instance, we might write: let $\underline{t}$ be a column-vector of order 3. We can also indicate the fact that a vector is a column (row) by stating its order as m by 1 (1 by n).

We may then define a (column) vector of consumer's expenditure on fuel and light, of order 3 (by 1).

$$\underline{t} = \begin{bmatrix} 879 \\ 940 \\ 973 \end{bmatrix}$$

The corresponding row-vector will be of order 1 by 3:

$$\underline{t}' = \begin{bmatrix} 879 & 940 & 973 \end{bmatrix}$$

The statement $\underline{t}'$ is a row-vector of order 1 by 3 is legitimate, but gives more information than is strictly needed. The 1966 figure can be indicated either as $t_3 = 973$, or as $t'_3 = 973$. The prime is quite superfluous here; hence we normally write $t_3 = 973$. Elements of a vector should always be indicated with their index. The use of ordinary small letters without index or heavy print (underlining), is conventionally reserved for variables or coefficients of an integer nature, such as indices and order-parameters.

Occasionally, one may also meet a single coefficient, which is not an element of a matrix or vector. Such a scalar is then indicated with a Greek letter. If required, a scalar can be interpreted as a matrix of order 1 by 1, as a column of order 1, or as a row of order 1.

A matrix which satisfies the property $a_{ij} = a_{ji}$ (and therfore obviously m = n) is called a symmetric matrix, e.g.

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

2.3 The transpose of a matrix

In paragraph 2 of this chapter, we met the following tableau:

British consumer's expenditure, at 1958 constant prices, on fuel and light:

	1964	1965	1966
Coal and coke	273	269	257
Electricity	381	417	435
Gas	167	194	223
Other	58	60	58

Now consider the tableau:

British consumer's expenditure, at 1958 constant prices, on fuel and light:

	Coal & coke	Elect-ricity	Gas	Other
1964	273	381	167	58
1965	269	417	194	60
1966	257	435	223	58

It will be observed, that this tableau gives exactly the same numerical information, as the previous one. But the presentation is different.

The corresponding matrices are said to be each other's transpose

$$C = \begin{bmatrix} 273 & 269 & 257 \\ 381 & 417 & 435 \\ 167 & 194 & 223 \\ 58 & 60 & 58 \end{bmatrix} \quad \text{and} \quad C' = \begin{bmatrix} 273 & 381 & 167 & 58 \\ 269 & 417 & 194 & 60 \\ 257 & 435 & 223 & 58 \end{bmatrix}$$

The transpose of a matrix is another matrix, with the rows of the first matrix as columns, and the columns of the first matrix as rows. It is conventional to indicate a transposition by a prime.

It follows that if a matrix A is of order m by n, then A' is of order n by m. The transpose of the transpose will have the rows of the transpose (= the columns of the matrix itself) as columns, and the columns of the transpose (= the rows of the matrix itself) as rows. The transpose of the transpose is the matrix itself, i.e.

$$(A')' = A$$

2.4 Addition and subtraction

The sum of two matrices of the same order is a matrix of the same order again. The elements of the result are the sums of the corresponding elements of the operands. The relation

$$A + B = C$$

pre-supposes, that A, B, and C are all of the same order. Furthermore, the statement is equivalent to:

$$a_{i,j} + b_{i,j} = c_{i,j}$$

for all admissable values of i and j.

Since,

$$a_{i,j} + b_{i,j} = b_{i,j} + a_{i,j}$$

we also have:

$$A + B = B + A$$

Similarly, the statement

$$A - B = D$$

pre-supposes, that A, B and D are all of the same order, while

$$a_{i,j} - b_{i,j} = d_{i,j}$$

The same logic applies to vectors as well.
The statement:

$$\underline{a} + \underline{b} = \underline{b} + \underline{a} = \underline{c}$$

pre-supposes that $\underline{a}$, $\underline{b}$, and $\underline{c}$ are all of the same order and gives the information:

$$a_i + b_i = c_i$$

And the subtraction of one vector from another:

$$\underline{a} - \underline{b} = \underline{d}$$

means $a_i - b_i = d_i$, for all positive integer values of i, smaller than or equal to the (identical) orders of $\underline{a}$, $\underline{b}$ and $\underline{d}$.

One does not always use a separate letter for the result of a matrix (or vector) operation. Instead one can use compound

expressions within brackets, like (A + B); ($\underline{a}$ + $\underline{b}$), etc.

Matrices and vectors can be added and subtracted like ordinary numbers; by performing these operations on their corresponding elements.

This also applies to rows. It follows that if we have

$$\underline{a} + \underline{b} = \underline{c},$$

we will also have

$$\underline{a}' + \underline{b}' = \underline{c}'$$

Note that expressions like

$$\underline{a} + \underline{b}', \quad \text{or} \quad A + \underline{b}$$

(in general additions of operands of different order) are just plain nonsense.

Operations must be carried out on corresponding elements.

If the operands are not of the same order, there just <u>are</u> no corresponding elements.

Example of the addition of two matrices:

In section 1.2 several of such additions were performed, such as:

$$\begin{bmatrix} -0.150 & 0.850 & ----- & 210 \\ -0.167 & -0.100 & 1.000 & --- \end{bmatrix} + \begin{bmatrix} 0.150 & -0.011 & --- & 41 \\ 0.167 & -0.013 & --- & 46 \end{bmatrix}$$

$$= \begin{bmatrix} --- & 0.839 & ---- & 251 \\ --- & 0.113 & 1.000 & 46 \end{bmatrix}$$

Exercise: Find some more matrix additions, in the same paragraph. (This may involve permutation of the rows of the tableaux).

2.5 Matrix multiplication

The product A . B is defined only if the number of columns of A is equal to the number of rows of B. The result is a matrix with the same number of rows as A, and the same number of columns as B.

Let A be a matrix of order m by n, and B a matrix of order n by k. And let C be a matrix of order m by k. Then the numerical content of A, B, and C satisfies

$$A \,.\, B = C$$

if we have,

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$$

$$(i = 1, 2, \ldots m, \quad j = 1, 2, \ldots k)$$

From this definition it follows that we will not in general have

$$A \cdot B = B \cdot A$$

We cannot even say that the normal rule is $A \cdot B \neq B \cdot A$

The fact that A . B is defined does not imply that B . A is defined. And something that is not defined cannot even be unequal to something else.

Only if A, B, and C are all symmetric and of the same order we have the identity

$$\sum_{r=1}^{n} a_{ir} b_{rj} = \sum_{r=1}^{n} a_{ri} b_{jr} = \sum_{r=1}^{n} b_{jr} a_{ri} = c_{ij} = c_{ji}$$

or, using matrix notation

$$AB = A'B' = B'A' = C = C'$$

Examples:

$$\underset{A}{\begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}} \underset{B}{\begin{bmatrix} 1 & 1 & -2 \\ 1 & 2 & 3 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 3 & 5 & 4 \\ 0 & 1 & 5 \end{bmatrix}}; \quad \underset{D}{\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}} \underset{E}{\begin{bmatrix} 5 & -1 \\ -1 & 2 \end{bmatrix}} = \underset{F}{\begin{bmatrix} 3 & 3 \\ 7 & 4 \end{bmatrix}}$$

detail-example for $c_{1,2}$:

$$c_{1,2} = a_{1,1} \cdot b_{1,2} + a_{1,2} \cdot b_{2,2}$$
$$= 1 \cdot 1 + 2 \cdot 2$$
$$= 1 + 4 = 5$$

Exercise: Analyse in a similar way, the computation of the other elements of C.

2.6 The product of a matrix and a vector

Since a vector is a matrix with one of the order parameters being unity, vector-operations have been defined implicitly in the previous paragraph. The product of a matrix and a vector, if it is defined (if the operands are of consistent orders), is a vector.

Let A be matrix of order m by n. And let $\underline{b}$ be a vector of order n by 1. Then the product $\underline{c} = A\,\underline{b}$ exists, and is of order m by 1. Its numberical value satisfies

$$c_i = \sum_{j=1}^{n} a_{i,j} \cdot b_j$$

$$(i = 1, 2 \ldots m)$$

And for a row: Let A be of order m by n, and $\underline{d}'$ of order 1 by m. Then the product $\underline{f}' = \underline{d}'A$ exists, and is of order 1 by n. Its numerical value satisfies

$$f_j = \sum_{i=1}^{m} d_i \cdot a_{i,j}$$

$$(j = 1, 2 \ldots n)$$

The expression $A\,\underline{b}$ is known as post-multiplication of A by the column $\underline{b}$. The expression $\underline{d}'A$ is known as pre-multiplication of A by the row $\underline{d}'$. Note that post-multiplication of a matrix by a row is nonsense. The same applies to pre-multiplication of a matrix by a column. But the expression $\underline{b}'\,A'$ is legitimate. This is the pre-multiplication of the transpose of A, by the row $\underline{b}'$.

The statement $\underline{c} = A\,\underline{b}$ implies $\underline{c}' = \underline{b}'\,A'$. The two expressions are in fact identical, both giving the numerical information

$$c_i = \sum_{j=1}^{n} a_{i,j} \cdot b_j$$

Example:

The computation of a sum-total.

$$\underline{s}'\ C = \underline{t}'$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 273 & 269 & 257 \\ 381 & 417 & 435 \\ 167 & 194 & 223 \\ 58 & 60 & 58 \end{bmatrix} = \begin{bmatrix} 879 & 940 & 973 \end{bmatrix}$$

Here a vector of 4 unity-elements serves as an operator, to add the rows of C.

2.7 Vector-vector multiplication

Again, the definition of the product of two vectors is implicitly given in the definition of the product of two matrices. There are two cases:

The outer product is always defined. A row is pre-multiplied by a column, or, what is the same thing, a column is postmultiplied by a row. The product is a matrix. Let $\underline{k}$ be a column of order m, and let $\underline{v}'$ be a row of order n, $\underline{v}$ being the corresponding column. The product

$U = \underline{k} \cdot \underline{v}'$, then is of order m by n; $u_{i,j} = k_i \cdot v_j$.

$(i = 1, 2 \ldots m, j = 1, 2 \ldots n)$

The inner product of two vectors only exists, if the two vectors are of the same order. The result is of order 1 by 1, i.e. a scalar number. If $\underline{v}$ and $\underline{k}$ are (column) vectors of the same order n, the inner product

$\mu = \underline{v}' \underline{k}$ exists, and has the value $\sum_{i=1}^{n} v_i \cdot k_i$.

Example of an outer product:

The substitution, or rather the numerical operation representing a substitution carried out in section 1.2:

$$\begin{bmatrix} 0.150 \\ 0.167 \end{bmatrix} \begin{bmatrix} 1.000 & -0.076 & \text{----} & 276 \end{bmatrix} = \begin{bmatrix} 0.150 & -0.011 & \text{----} & 41 \\ 0.167 & -0.013 & \text{----} & 46 \end{bmatrix}$$

Example of an inner product:

The computation of a sum of squares.
Let $\underline{v}'$ be a row-vector, of order 4, e.g. $\underline{v}' = \begin{bmatrix} 2 & 5 & -1 & 7 \end{bmatrix}$
The sum of squares of the elements of $\underline{v}$ (or $\underline{v}'$) will be

$$\underline{v}' \cdot \underline{v} = \begin{bmatrix} 2 & 5 & -1 & 7 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ -1 \\ 7 \end{bmatrix}$$

$$= 2 \times 2 + 5 \times 5 + (-1 \times -1) + 7 \times 7$$
$$= 4 + 25 + 1 + 49$$
$$= 79.$$

Exercise:

Find and evaluate 5 different legitimate product-expressions of

two different vectors, both operands being always two of the following vectors

$$\underline{v}_1 = \begin{bmatrix}1\\2\end{bmatrix}, \underline{v}_2 = \begin{bmatrix}2\\3\end{bmatrix}, \underline{v}'_3 = [3, 2], \underline{v}'_4 = [4, -1, 2, 1].$$

2.8 Multiplication by a scalar

The multiplication of a matrix by a scalar, and the multiplication of a vector by a scalar, is defined as a multiplication of the individual elements of the matrix, or of the vector, by that scalar. The order in which the operands are written is arbitrary here.

$$C = \alpha B \text{ means } c_{i,j} = \alpha b_{i,j}$$

The expressions αB and $B\alpha$ are equivalent ways of writing the product of a matrix and a scalar. For a column the two expressions $\alpha \underline{b}$ and $\underline{b}\alpha$ are again both correct and equivalent. The same applies for a row $\alpha \underline{b}' = \underline{b}'\alpha$. But it will be prudent to write $\underline{b}\alpha$ and $\alpha \underline{b}'$, instead of $\alpha \underline{b}$ and $\underline{b}'\alpha$. The reason for this is the possibility to interpret a scalar as a matrix of order 1 by 1.

Example of the multiplication of a row by a scalar

$$1.153 \; . \; [0.867 \; -0.066 \; --- \; 240] = [1.000 \; -0.076 \; --- \; 277]$$

(See para 1.1 and 1.2, division of the (11) row by 0.867).

2.9 Matrix-matrix multiplication by columns or by rows

The product A . B = C can be evaluated by its columns:

$$A \, \underline{b}_j = \underline{c}_j$$

Here $\underline{b}_j$ is the j^{th} column of B, and $\underline{c}_j$ is the j^{th} column of C.

Example:

$$A = \begin{bmatrix}2 & -1\\1 & 3\end{bmatrix}$$

and

$$B = \begin{bmatrix}-1 & 5 & 0\\1 & -2 & 4\end{bmatrix}$$

We then have, for the <u>first</u> column of the result:

$$\begin{bmatrix}2 & -1\\1 & 3\end{bmatrix} \begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}-3\\2\end{bmatrix}$$

and for the second column

$$\begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ -2 \end{bmatrix} = \begin{bmatrix} 12 \\ -1 \end{bmatrix}$$

and for the third column

$$\begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -4 \\ 12 \end{bmatrix}$$

This gives for the whole matrix:

$$\begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} -1 & 5 & 0 \\ 1 & -2 & 4 \end{bmatrix} = \begin{bmatrix} -3 & 12 & -4 \\ 2 & -1 & 12 \end{bmatrix}$$

The above procedure is not something different from the definition of matrix multiplication in section 2.5. It is only an ordered way of carrying out the computations of the elements of C.

The product A . B = C can also be evaluated, by its rows:

$$\underline{a}_i' \, B = \underline{c}_i'$$

Here $\underline{a}_i'$ is the i^{th} row of A, and $\underline{c}_i'$ is the i^{th} row of C. We can illustrate this with the same matrices A, B and C.

$$\begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} -1 & 5 & 0 \\ 1 & -2 & 4 \end{bmatrix} = \begin{bmatrix} -3 & 12 & -4 \end{bmatrix}$$

(for the first row)
and (for the second row)

$$\begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} -1 & 5 & 0 \\ 1 & -2 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 12 \end{bmatrix}$$

Obviously, the result $C = \begin{bmatrix} -3 & 12 & -4 \\ 2 & -1 & 12 \end{bmatrix}$

should be the same, irrespective of the order in which the elements were computed.

2.10 Substitution

When two symbolic expressions are identical, one is allowed to interchange, or substitute, the one for the other. This postulate is vital to all algebra. It applies to matrix algebra as well.

Example:

Let A, B, $\underline{x}$, $\underline{y}$, and $\underline{b}$ be of orders m by n, n by k; n, k, and m. And let us have:

$$A\underline{x} = \underline{b} \quad \ldots\ldots\ldots\ldots \qquad (2.10.1)$$

and

$$B\underline{y} = \underline{x} \quad \ldots\ldots\ldots\ldots \qquad (2.10.2)$$

Now substitute $B\underline{y}$ for $\underline{x}$ by (2.10.2) into (2.10.1), to obtain:

$$A(B\underline{y}) = \underline{b} \quad \ldots\ldots\ldots\ldots \qquad (2.10.3)$$

2.11 Forms

Let $\underline{a}'$ be a row, of order 1 by m;
let B be a matrix, of order m by n;
let $\underline{c}$ be a column, of order n by 1.

Then

$$\underline{a}' (B\underline{c}) = (\underline{a}'B)\underline{c} \quad \ldots\ldots\ldots \qquad (2.11.1)$$

holds.

The relation (2.11.1) is an identity, as may be shown as follows:

Evaluate each i^{th} element of the column within the brackets on the left-hand side of (2.11.1) as

$$\sum_{j=1}^{n} b_{ij}\, c_j \quad ;$$

and each j^{th} element of the row within the brackets on the right-hand side of (2.11.1) as

$$\sum_{i=1}^{m} a_i\, b_{ij} \quad .$$

The expressions on both sides of (2.11.1) are of order 1 by 1, a scalar, and we may independently evaluate both sides of (2.11.1).

We obtain

$$\sum_{i=1}^{m} a_i \left(\sum_{j=1}^{n} b_{ij}\, c_j \right) = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_i\, b_{ij} \right) c_j \qquad (2.11.2)$$

This is a true statement, an identity, and hence the same is true for (2.11.1).

In both relations, the brackets are superfluous, and we may write

$$\sum_{i=1}^{m} a_i \sum_{j=1}^{n} b_{ij}\, c_j \equiv \sum_{j=1}^{n} c_j \sum_{i=1}^{m} a_i\, b_{ij}$$

$$\equiv \sum_{i=1}^{m} \sum_{j=1}^{n} a_i\, b_{ij}\, c_j \quad \ldots\ldots (2.11.3)$$

and its matrix-equivalent

$$\underline{a}'\,(B\underline{c}) \equiv (\underline{a}'B)\,\underline{c} \equiv \underline{a}'B\underline{c} \quad \ldots\ldots\ldots (2.11.4)$$

An expression of this type, i.e. a matrix pre-multiplied by a row and post-multiplied by a column, to yield a single number as result, is known as a form (or bilinear form).

Example:

The form $\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & -1 & 5 \\ 3 & 1 & 9 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$

may be evaluated either as

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 8 \end{bmatrix} = 5$$

or as

$$\begin{bmatrix} 2 & 0 & 7 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix} = 5$$

2.12 Recursive products

The identity $\underline{a}'\,(B\underline{c}) = (\underline{a}'B)\,c$ ..... (2.11.1)

is also true if $\underline{a}'$ is the i^{th} row of the matrix A.

If A has h rows, we may then write

$$\underline{a}_i'\,.\,(B\underline{c}) \equiv (\underline{a}_i'B)\,c \quad \ldots\ldots\ldots (2.12.1)$$

$(i = 1, 2 \ldots\ldots\ldots h)$

The same information, may be presented more compactly, as:

$$A\ (B\ \underline{c}) \equiv (A\ B)\ \underline{c} \quad \ldots\ldots\ldots \quad (2.12.2)$$

But also, $\underline{c}$ may be a column of a matrix C, which has k columns,

$$A\ (B\ \underline{c}_j) \equiv (A\ B)\ \underline{c}_j \quad \ldots\ldots\ldots \quad (2.12.3)$$

$(j = 1, 2 \ldots\ldots\ldots k)$

The information given by (2.12.3) is written more compactly as

$$A\ (B\ C) \equiv (A\ B)\ C \quad \ldots\ldots\ldots \quad (2.12.4)$$

A direct corrollary of (2.12.4) is

$$A\ (B\ C\ D) \equiv (A\ B)\ (C\ D) \equiv (A\ B\ C)\ D \quad \ldots\ldots \quad (2.12.5)$$

Clearly there is no point in writing brackets inside product expressions of this type, and (2.12.5) defines the notion of a recursive product ABCD.

Exercise

Find the numerical content of

$$ABC = \begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}1 & 3\\ -1 & -2\\ 2 & 5\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}$$

Check by independently calculating (AB)C and A(BC), finding the result the same.

Note

AB = BA is <u>not</u> generally true. This applies to recursive products just as well. Thus

$$\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix} = 5\ ,\ \text{but}\ \begin{bmatrix}1\\ 2\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}1 & 2\\ 2 & 4\end{bmatrix}$$

Even if the order is the same, the numerical content may not be

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}2 & 3\\ 4 & 5\end{bmatrix} = \begin{bmatrix}10 & 13\\ 22 & 29\end{bmatrix},\ \text{but}\ \begin{bmatrix}2 & 3\\ 4 & 5\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} = \begin{bmatrix}11 & 16\\ 19 & 28\end{bmatrix}.$$

2.13 The addition of several matrices

Since addition of matrices consists of addition of the elements the normal rules of addition and subtraction of numbers also

apply to matrices. The meaning of expressions like A + B - C. -A + B + αC, etc. will need no further elucidation. We obviously have

$$A + (B + C) = (A + B) + C, \text{ etc.}$$

In each case these expressions pre-suppose that all the operands are of the required (same) orders.

2.14 Transposition of compound expressions

The transpose of a sum-expression is the sum of the transposes of its terms.

$$(A + B + C + D)' = A' + B' + C' + D'$$

The transpose of a product is somewhat more complicated:

$$(A\ B\ C\ D)' = D'\ C'\ B'\ A'$$

In both cases, the transposition rule is a direct result from the definition of a sum, or a product, respectively. For a sum, the relation S = A + B and the relation S' = A' + B' both mean

$$s_{i,j} = a_{i,j} + b_{i,j}$$

If the number of columns of A and the number of rows of B are equal and indicated as r,

$$P = A \,.\, B \text{ is equivalent to } P' = B' \,.\, A'$$

Both expressions give the numerical information

$$P_{i,j} = \sum_{k=1}^{r} a_{i,k} \cdot b_{k,j} = \sum_{k=1}^{r} b_{k,j} \cdot a_{i,k}$$

2.15 Some special matrices and vectors

A matrix with two equal order-parameters is named a square matrix. (The number of columns is the same as the number of rows.) The vector, consisting of the elements of the matrix with equal row- and column-indices, is then the (main) diagonal.

Example

$$\begin{bmatrix} -1 & 5 & 8 \\ 3 & 6 & 4 \\ -2 & 8 & 7 \end{bmatrix}$$ is a square matrix;

$\begin{bmatrix} -1 \\ 6 \\ 7 \end{bmatrix}$ is the corresponding diagonal.

A square matrix, with non-zero elements on the main diagonal only, is a diagonal matrix.

A special kind of diagonal matrix is the unit matrix. This is a diagonal matrix, with all the elements on the diagonal being unity. Unit matrices are normally indicated with a capital letter I. When the order of the unit matrix is self-evident, it is not strictly necessary to indicate that order. But one may write an index beside the capital I, to indicate the order. A unit matrix of order n, can be indicated as I, or as I_n. Multiplication by a unit matrix amounts to copying, i.e.

$$I\,\underline{v} = \underline{v} \; ; \; \underline{w}'\,I = \underline{w}' \quad , \quad \text{and}$$

$$I\,A = A \; ; \; B\,I = B.$$

For this reason the unit matrix is also known as the "identity matrix".

A unit vector is a vector of zero's, and one single unity-element. The unit-vector is usually indicated with the letter $\underline{e}$. An index with the vector will indicate the place of the unity-element, not the order of the vector. If the order is not self-evident, it must be stated separately.

If $\underline{a}_j$ denotes the j^{th} column of A, we will have

$$\underline{a}_j = A\,\underline{e}_j$$

If $\underline{a}_i'$ denote the i^{th} row of A, we will have:

$$\underline{a}_i' = \underline{e}_i'\,A$$

Example

$$\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 12 \\ 22 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} = \begin{bmatrix} 21 & 22 & 23 \end{bmatrix}$$

Unit vectors are often met as operator, to isolate vectors out of matrices. If we restrict ourselves to matrix notation (there are other types of operators) an operator is a matrix, or a vector, not giving numerical information, but only serving a

purpose.

Another specially structured (square) matrix is a triangular matrix. There are two kinds of triangular matrices, upper-triangular matrices and lower-triangular matrices. As the name suggests, the non-zero part of a triangular matrix is triangularly shaped. An upper-triangular matrix has the non-zero elements only on and above the diagonal; a lower-triangular matrix has the non-zero part on or below the diagonal,

$$a_{ij} = 0 \text{ for } j < i, \text{ or } i < j \text{ for a lower-triangular matrix.}$$

The "non-zero" part may of course contain zeros as well.

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 5 \end{bmatrix} \quad \text{is an upper triangular matrix.}$$

What matters is that there are only zeros on the opposite side.

2.16 Matrix partitioning

A number of rows of a matrix may be grouped into a block-row. Again, a number of columns of a matrix may be grouped into a block-column.

Example

Let us have the matrix

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 5 & 6 \\ 0 & 0 & 7 & 8 \end{bmatrix}$$

We may then group the first two columns of A into a block-column, to be denoted as A_1:

$$A_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Regretfully, there is no standard notation for the partitioning of matrices. For block-rows, this may lead to ambiguity. The symbol A_1' could either be the transpose of A_1; or it could be the first block-row. Also, a series of matrices is sometimes indicated with indices, while they are not block-columns.

The position is somewhat better with blocks. A block is the intersection of a block-column and a block-row. Blocks can be indicated with two indices. The first index will indicate the block-row, the second index, the block-column. In our above example, A may then be partitioned as:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Here $A_{11} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $A_{22} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

While A_{12} and A_{21} are zero blocks.

The fact that two of the 4 blocks were zero is not quite accidental. Matrix partitioning is a useful device, particularly when some of the blocks are zero.

Vectors may also be partitioned. Again, once the partitioning is defined, an index will be useful in indicating a sub-vector.

Example

$$\underline{r}' = [5 \quad 6 \quad 8 \quad 0 \quad 0 \quad 0]$$

$$\underline{r}_1' = [5 \quad 6 \quad 8]$$

$$\underline{r}_2' = [0 \quad 0 \quad 0]$$

Another name for a partitioned matrix (vector) is a composite matrix (vector).

2.17 Multiplication by partitioning

Let the matrix A be partitioned into two block-rows

$$A = \begin{bmatrix} A_1' \\ A_2' \end{bmatrix}$$

Let A be of order m by n, and A_1' and A_2' of order m_1 by n and m_2 by n. Similarly, let B be of order n by k, and be partitioned into two block-columns.

$$B = [B_1 \quad B_2]$$

B_1 and B_2 being of order n by k_1 and n by k_2.

The product A B may then be evaluated as:

$$A\,B = \begin{bmatrix} A_1'B_1 & A_1'B_2 \\ A_2'B_1 & A_2'B_2 \end{bmatrix} \qquad \ldots\ldots \quad (2.17.1)$$

Proof

Take any element of A B to be indicated as the (i,j) element (i^{th} row, j^{th} column). This is the inner product of the i^{th} row of A and the j^{th} column of B, by both the left-hand and the right-hand side of (2.17.1). This is so, irrespective of the additional information, given by the right hand side of (2.17.1), that a row of A is a row of the first/second block-row of A, and a column of B is a column of the first/second block-column of B. q.e.d.

The statement is hardly a theorem at all, but merely a corrollary of the definition of matrix multiplication.

Example

$$\underset{A}{\begin{bmatrix} 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ -3.1 & -3.2 & -3.3 & -3.4 \\ -4.1 & -4.2 & -4.3 & -4.4 \end{bmatrix}} \underset{B}{\begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}} = \begin{bmatrix} 5.0 & -5.0 \\ 9.0 & -9.0 \\ -13.0 & 13.0 \\ -17.0 & 17.0 \end{bmatrix}$$

The 2 by 1 positive block of the product A B

$$[A\,B]_{1,1} = \begin{bmatrix} 5.0 \\ 9.0 \end{bmatrix}$$

may also be obtained as $A_1'B_1$

$$\begin{bmatrix} 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.0 \\ 9.0 \end{bmatrix}$$

The case discussed above, could be compared with the outer product of two vectors.

A corresponding analogy with the inner product of two vectors exists as well.

Let A (of order m by n) be partitioned as

$$A = [A_1 \quad A_2]$$

where the two block-columns A_1 and A_2 are of order m by n_1 and m by n_2. Let B (of order n by k) be partitioned as

$$B = \begin{bmatrix} B_1' \\ B_2' \end{bmatrix}$$

where the block-rows B_1' and B_2' will be of order n_1 by k and n_2 by k. The product A B may now be evaluated as

$$A\,B = A_1\,B_1' + A_2\,B_2' \qquad \ldots\ldots \qquad (2.17.2)$$

Again the statement hardly is a theorem at all.

Example

$$\begin{bmatrix} 1 & 2 & -3 & -4 \\ 5 & 6 & -7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} -3 & -4 \\ -7 & -8 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} =$$

$$\begin{bmatrix} 9 & 3 \\ 11 & 11 \end{bmatrix} + \begin{bmatrix} 7 & 7 \\ 15 & 15 \end{bmatrix} = \begin{bmatrix} 10 & 10 \\ 26 & 26 \end{bmatrix}$$

The two types of partitioning may be combined, and A and B may be partitioned as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad \text{and}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

The product A B may then be evaluated as

$$A\,B = \left[\begin{array}{lll|lll} A_{11} & B_{11} & + A_{12} & B_{21} & A_{11} & B_{12} & + A_{12} & B_{22} \\ \hline A_{21} & B_{11} & + A_{22} & B_{21} & A_{21} & B_{12} & + A_{22} & B_{22} \end{array}\right] \qquad (2.17.3)$$

The above relation assumes that all the expressions are legitimate, if they are, (2.17.3) is true, the two sides are equivalent.

It appears that we can treat blocks more or less like elements, and block-rows like rows and block-columns like columns.

Generally, we may obtain the (i, j) block of the matrix A B as the product of the i^{th} block-row of A and the j^{th} block-column of B.

$$[A\ B]_{i,j} = A_i' B_j = \sum_{h=1}^{p} A_{ih} B_{hj} \qquad (2.17.4)$$

Above it is assumed that A is partitioned in a number of block-rows, and B into an equal number of corresponding block-columns. It is also assumed that A is partitioned into p block-columns and B into p corresponding block-rows.

Exercise

Evaluate the products A B and B A where

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 1 & -1 \\ 3 & 4 & -1 & 1 \\ \hline 1 & -1 & 2 & 3 \\ -1 & 1 & 4 & 5 \end{array}\right], \qquad B = \left[\begin{array}{cc|cc} 1 & -1 & 1 & 2 \\ -1 & 1 & 3 & 4 \\ \hline 2 & 3 & 1 & -1 \\ 4 & 5 & -1 & 1 \end{array}\right]$$

Do not forget to take advantage of the triviality of multiplication by a unit-matrix (block)!

Answer to the exercise at the end of section 2.17.

$$B\ A = \left[\begin{array}{cc|cc} 9 & 7 & 28 & 19 \\ 5 & 3 & 16 & 12 \\ \hline 30 & 22 & 9 & 7 \\ 13 & 11 & 5 & 3 \end{array}\right]$$

$$A\ B = \left[\begin{array}{cc|cc} 9 & 7 & 38 & 28 \\ 5 & 3 & 21 & 17 \\ \hline 23 & 15 & 9 & 7 \\ 10 & 8 & 5 & 3 \end{array}\right]$$

2.18 Differentiation of matrix expressions

Rules for differentiating a number of not too complicated matrix expressions follow readily from the definition of the various matrix operations.

We shall need to come back to this point after introducing more complicated matrix expressions, but differentiation of the sums

and products of matrices and vectors is discussed here as follows:

The differentiation of a sum or difference expressions is obvious, and is given here only for the sake of completeness.

If a matrix A is defined as being the sum of two matrices B and C

$$C = A + B \tag{2.18.1}$$

and the elements of A and B are not constants, but variables, we obviously have

$$dC = dA + dB \tag{2.18.2}$$

Product expressions become more complicated.

From

$$C = A\,B \tag{2.18.3}$$

we obtain, for finite differences

$$\begin{aligned} C + \Delta C &= (A + \Delta A)(B + \Delta B) = A(B + \Delta B) + \Delta A(B + \Delta B) \\ &= A\,B + A\,\Delta B + \Delta A\,B + \Delta A\,\Delta B \end{aligned} \tag{2.18.4}$$

From which we obtain (for finite differences)

$$\Delta C = A\Delta B + \Delta A\,B + \Delta A\;\Delta B \tag{2.18.5}$$

As usual in differentiation, we assume infinitely small changes, and we are allowed to suppress products of infinitely small numbers. Therefore, from (2.18.5) we infer that (2.18.3) implies

$$dC = A\,dB + dA\,B \tag{2.18.6}$$

The usual differentiation of a rational function, i.e. $y = x^n$ implies $dy = nx^{n-1}\,dx$ applies to matrices, only for a square and symmetric matrix, i.e. for a symmetric matrix A, the matrix of first difference of A^p is

$$d(A^p) = pA^{p-1}\,dA \tag{2.18.7}$$

where A^p indicates the p^{th} power of A

$$A = A\,A \ldots\ldots\ldots A$$

We first illustrate this rule for n = 2.

For n = 2, we obtain from (2.18.6)

$$dA^2 = d(A\,A) = A\,dA + dA\,A \qquad (2.18.8)$$

Provided we also impose the requirement of symmetry on the matrix of differences, the two terms on the righthand side of (2.18.8) are identical, and the result is

$$dA^2 = 2A\,dA \qquad (2.18.9)$$

The more general result (2.18.7) may now be shown by recursive induction, i.e. if we assume it to be valid for some p, we obtain for q = p + 1, from (2.18.8)

$$\begin{aligned} d\,A^q = d(A^pA) &= d(A\,A^p) = A^p\,dA + p\,A^{p-1}\,dA\,A \\ &= A^p\,dA + p\,A^{p-1}\,A\,dA = A^p\,dA + pA^pdA \\ &= (p+1)A^p\,d\,A = q\,A^{q-1}\,dA \qquad (2.18.10) \end{aligned}$$

The more complicated problem of differentiating the p^{th} power of a square but non-symmetric matrix is not dealt with here.

Differentiation of compound matrix expressions is not in fact very widely practiced. Much more common is the differentiation of vectorial expressions in which matrices occur as known coefficients matrices. In particular, for a symmetric quadratic form

$$y = \underline{x}'\,A\,\underline{x}$$

where A is a symmetric matrix of fixed coefficients, the differential expression is

$$\begin{aligned} d\,y &= \underline{x}'\,d(A\underline{x}) + d\underline{x}'(A\underline{x}) \\ &= \underline{x}'\,Ad\underline{x} + d\,\underline{x}'\,A\underline{x} = 2\,\underline{x}'\,Ad\underline{x} \qquad (2.18.11) \end{aligned}$$

Example

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad \underline{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$d\,y = 2\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} d\underline{x} = 10dx_1 + 8dx_2$$

Matrix notation may also be used to obtain compact notation of partial differentiation.

The significance of the operation is, of course, the same as in "ordinary" calculus i.e. the partial derivative is the ratio between the change in the value of a function and a small change in one of its arguments, the other arguments of the function maintaining a constant value.

Matrix notation comes in because we may use matrix notation to indicate and to express as formulae, vectors partial derivatives, and sometimes matrices of partial derivatives. The quadratic form is again the most widely used example.

From (2.18.11), we obtain, by assuming $dx_i = 0$ for $i \neq j$, but $d x_j \neq 0$,

$$d y = 2 \underline{x}' \underline{a}_j d x_j \tag{2.18.12}$$

Hence, by the definition of a partial derivative

$$\frac{\partial y}{\partial x_j} = 2 \underline{x}' \underline{a}_j \quad \text{(all j)} \tag{2.18.13}$$

The expression (2.18.13) is not very efficient use of matrix notation, it was formulated purely to illustrate the relationship with ordinary calculus. A more usual and more compact hence more efficient - notation is

$$\frac{\partial y}{\partial \underline{x}} = \frac{\partial \underline{x}' \underline{\underline{A}} \underline{x}}{\partial \underline{x}} = 2 \underline{\underline{A}} \, d\underline{x} \tag{2.18.14}$$

It is however, quite possible to do the same for the matrix. If $\underline{a}$ is a known coefficients vector, and X a matrix of variables,

$$y + \Delta y = \underline{a}'(X + \Delta X) \underline{a} = \underline{a}' X \underline{a} + \underline{a}' \Delta X \underline{a} \tag{2.18.15}$$

obviously implies

$$d y = \underline{a}' d X \underline{a} \tag{2.18.16}$$

From which for $d x_{ij} \neq 0$, $d_{rk} = 0$ (at $r \neq i$, all $k \neq j$)

$$\frac{\partial y}{\partial x_{ij}} = a_i a_j \tag{2.18.17}$$

or using matrix notation (an outer product)

$$\frac{\partial \underline{a}' X \underline{a}}{\partial X} = \underline{a} \underline{a}' \tag{2.18.18}$$

Note, that no symmetry needs to be assumed in this case, and moreover,

$$\frac{\partial \underline{a}' X \underline{b}}{\partial X} = \underline{a}\,\underline{b}' \qquad (2.18.19)$$

is also true for a non-square matrix.

Exercise

Derive a formula for the matrix of second-order derivatives of a quadratic form.

$$\frac{\partial^2 \underline{x}' A \underline{x}}{\partial^2 \underline{x}} = \quad ?$$

2.19 Reading and printing of large matrices by electronic computers

There is, in practice a considerable discrepancy between the software provided as standard facilities on most computers and what is practical for handling large matrices and tableaux.

The two procedures listed in this section are offered as a possible way to fill this gap. (There are in fact other solutions to this problem.)

The precise action of these procedures will be clear from their text, and from the extensive use of comment in the ALGOL text itself, and from the names of the labels used. But some remarks about the processing of large matrices in general seem appropriate at this point.

Firstly, many large matrices and tableaux e.g. linear programming tableaux contain a very large percentage of zero elements. It is obviously a waste of time and cards, to punch all these zeros.

Secondly, simple errors, numbers in the wrong column, decimal errors do occur during card-punching. It seems wise to write programmes, bearing that fact of life in mind. Hence the warning messages for repeat requests at inappropriate points, and the automatic back-print of a matrix which has just been read.

Matrix-output is by itself a problem with most standard facilities, as these often make a ridiculously wasteful use of line-printer paper and of the users hands and eyes, forcing him to skim through a pile of paper in order to verify one element. Hence the print per block-column and the splitting of large block-columns into two square blocks per page.

The two procedures are now listed.

TEXT-LISTING OF THE MATRIX PRINTING PROCEDURE:

```
'PROCEDURE' MATO(MATR,SCALE,M,N,SR,SC);
'ARRAY' MATR; 'REAL' SCALE; 'INTEGER' M,N,SR,SC;

'BEGIN' 'INTEGER' NRDO,NRTD,NCOLDO,NCOLTD,
PAGEC,RPRINP,CPRINP,I,J;

  'COMMENT'
  MATO STANDS FOR MATRIX OUT.
  IT IS A MATRIX PRINTING PROCEDURE, PRINTING THE MATRIX
  IN BLOCKS.
  FOR AN 180 CHARACTER LINEPRINTER, THE ORDER OF THE BLOCKS
  IS PUT AT 15 BY 15.
  SR STANDS FOR SKIP ROWS.
  SC STANDS FOR SKIP COLUMNS.
  SCALE IS THE SCALE-FACTOR, I.E. THE MATRIX WILL BE PRINTED,
  MULTIPLIED BY SCALE.
  THE AUXILIARY VARIABLES NRDO,NRTD,NCOLDO,NCOLTD,PAGEC,
  RPRINP, AND CPRINP  STAND FOR:
  NUMBER OF ROWS DONE,
  NUMBER OF ROWS TO DO,
  NUMBER OF COLUMNS DONE,
  NUMBER OF COLUMNS TO DO,
  PAGE CONTROL,
  ROW PRINT PARAMETER (DETERMINATING
  THE NUMBER OF ROWS IN A BLOCK)
  COLUMN PRINT PARAMETER
  (DETERMINING THE NUMBER OF COLUMNS IN A BLOCK);

  NRDO:=NCOLDO:=0; NRTD:=M; NCOLTD:=N;

  PRINT FOR COLUMN IF APPROPRIATE:
  'IF' M > 15 'AND' N=1 'THEN' 'BEGIN'
    WRITETEXT('('LINEWISE%PRESENTATION%OF%A%COLUMN')');
    NEWLINE(1); 'END';

  START:
  NEWLINE(2);
  RPRINP:=CPRINP:=15;

  ADJUST NUMBER OF  OLUMNS:
  'IF' NCOLTD < CPRINP 'THEN' CPRINP:=NCOLTD;
  NCOLTD:=NCOLTD-CPRINP;

  CHECK FOR PAGE:
  'IF' M>15 'AND' N>15 'THEN' PAGEC:=NRDO/30-ENTIER(NRDO/30)
  'ELSE' PAGEC:=1;
  'IF' PAGEC=0 'THEN' 'BEGIN'
    'COMMENT'
    BLOCKS ALL TO START AT THE TOP OF A PAGE.
    THIS ALLOWS THE USER TO STICK THEM TOGETHER,
    WHILE KEEPING ROWS AND COLUMNS IN LINE.;
    RUNOUT;    'END';
```

```
  HEADING:
  'IF' M>15 'AND' N>1 'THEN' 'BEGIN'
    'COMMENT'
    IN THE INTEREST OF BEING ABLE TO FIND INDIVIDUAL ENTRIES
    OF THE MATRIX, A LARGISH MATRIX IS EDITED.
    THIS IS DONE BY PRINTING THE INDICES OF THE LEADING
    ELEMENT AT THE HEAD OF EACH BLOCK.;
    NEWLINE(2);
    WRITE(30,FORMAT('('SNDD')'),NRDO+1);
    WRITE(30,FORMAT('('SNDDDDD')'),NCOLDO+1); 'END';

  ADJUST NUMBER OF ROWS:
  'IF' NRTD < RPRINP 'THEN' RPRINP:=NRTD;
  NRTD:=NRTD-RPRINP;

  PRINT THE LINES:
  'FOR' I:=1 'STEP' 1 'UNTIL' RPRINP 'DO' 'BEGIN'

    NEWLINE EXCEPT FOR COLUMN:
    'IF' M>15 'AND' N=1 'THEN'  'GOTO' NO NEWLINE NEEDED;
    NEWLINE(1);
    NO NEWLINE NEEDED:

    'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO' 'BEGIN'
      'IF' MATR[SR+NRDO+I,SC+NCOLDO+J]=0 'THEN'
      WRITE(30,FORMAT('('SNDDDDSS')'),0)
      'ELSE' WRITE(30,FORMAT('('S-NDD.DD')'),
      SCALE*MATR[SR+NRDO+I,SC+NCOLDO+J]);
      'END'; 'END';

  ADJUST INDICES AND RETURN UNLESS READY:
  NRDO:=NRDO+RPRINP;
  'IF' NRDO=M 'THEN' 'BEGIN'
    NRTD:=M; NRDO:=0; NCOLDO:=NCOLDO+CPRINP;
    'IF' NCOLDO=N 'THEN' 'GOTO' END OF MATO;
    'GOTO' START; 'END' 'ELSE' 'GOTO' CHECK FOR PAGE;

  END OF MATO:  'END';
```

TEXT-LISTING OF THE MATRIX READING PROCEDURE:

```
'PROCEDURE' MATI(MATR,M,N,SR,SC);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC;
'BEGIN' 'INTEGER' I,J; 'REAL' NUM;
'PROCEDURE' MATO(MATR,SCALE,M,N,SR,SC);
'ARRAY' MATR; 'REAL' SCALE; 'INTEGER' M,N,SR,SC;  'ALGOL';

   'COMMENT' MATRIX READING PROCEDURE.
   MATI STANDS FOR MATRIX IN.
   SR STANDS FOR SKIP ROWS,
   I.E.  A LEADING BLOCK-ROW OF SR ROWS IS NOT FILLED.
   SC STANDS FOR SKIP COLUMNS,
   I.E.  A LEADING BLOCK-COLUMN OF SC COLUMNS IS NOT FILLED.
   M AND N ARE THE ORDER PARAMETERS OF THE BLOCK TO BE READ.
```

```
WITHIN A ROW OF A MATRIX, MATI INTERPRESTS NUMBERS
IN EXCESS OF 1000000 AS REPEAT-REQUESTS.
THE EXCESS OVER A MILLION IS THE MUMBER OF REPEATS, I.E.
36.52  1000100 MEANS  36.52, FOLLOWED BY 100 REPEATS OF
THAT SAME NUMBER,
HENCE IN TOTAL 101 TIMES  36.52 IS PUT IN THE MATRIX.;

NUM:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' MATR[SR+I,SC+J]:=0;

READ THE ROWS:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
   'IF' NUM < 1000000.5 'THEN' NUM:=READ;

   'IF' (J=1 'AND' NUM > 1000000.5)
   'OR' (J=N 'AND' NUM > 1000001.5)  'THEN' 'BEGIN'
      'COMMENT'
      THERE SHOULDN T BE A REPEAT REQUEST AT THIS POINT;
      NEWLINE(1);
      WRITETEXT('('UNEXPECTED%REPEATREQUEST%OF')');
      WRITE(30,FORMAT('('SNDDDDD.DS')'),NUM-1000000);
      'IF' J=1 'THEN' WRITETEXT('('AT%THE%
      BEGINNING%OF%ROW')');
      'IF' J=N 'THEN' WRITETEXT('('REMAINING%
      AT%THE%END%OF%ROW')');
      WRITE(30,FORMAT('('SNDDDD')'),I);
      NUM:=0;  'END';

   'IF' NUM < 1000000.5 'THEN' 'BEGIN'
      'IF' NUM=0 'THEN' 'GOTO' CELL FILLED;
      MATR[SR+I,SC+J]:=NUM 'END'
   'ELSE' 'BEGIN'
      MATR[SR+I,SC+J]:=MATR[SR+I,SC+J-1]; NUM:=NUM-1;
      'END';

   CELL FILLED: 'END';

MATO(MATR,1.0,M,N,SR,SC);
END OF MATI:  'END';
```

CHAPTER III

BLOCK-EQUATIONS AND MATRIX-INVERSION

3.1 Notation of linear systems

An "ordinary" system of linear equations can be formulated in matrix notation, as follows:

$$A \,.\, \underline{x} = \underline{b} \qquad (3.1.1)$$

Here, A is a known matrix, b is a known vector, while $\underline{x}$ is an as yet unknown vector. If A is of order n by n, then $\underline{x}$ must be of order n by 1, and $\underline{b}$ of order n by 1. The solution of such a system consists in evaluating $\underline{x}$ in such a way that the equations are satisfied. The evaluated vector $\underline{x}$ is then indicated as the solution vector.

3.2 Singularity

Consider the systems:

$$\begin{array}{rcrcl} x_1 & + & x_2 & = & 3 \\ 2x_1 & + & 2x_2 & = & 6 \end{array} \quad \text{and} \quad \begin{array}{rcrcl} x_1 & + & x_2 & = & 3 \\ 2x_1 & + & 2x_2 & = & 15 \end{array}$$

The first system is satisfied by $x_1 = 0$, with $x_2 = 3$. And also by $x_1 = 1$, with $x_2 = 2$; and also by $x_1 = -500$, with $x_2 = 503$. Any vector $\underline{x}$, satisfying $x_1 + x_2 = 3$, will also satisfy $2x_1 + 2x_2 = 6$.

On the other hand, the second system has no solution at all. The two systems have in common, a special condition of the matrix

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$

The second row of this matrix is proportional to the first. This condition is known as singularity. Singularity need not be so evident at a glance, as in this example. It may also be that a linear combination of rows is equal to another row.

The matrix $\begin{bmatrix} 1 & 5 & 3 \\ 2 & -3 & -7 \\ 2 & 4 & 0 \end{bmatrix}$ is singular as well.

A square matrix A is said to be singular, if there exists a non-zero vector $\underline{y}'$, such that $\underline{y}'\, A = 0$. (There exists then also a vector $\underline{x}$, such that $A\underline{x} = 0$, but no proof of that statement is offered at this stage.)

In our above two examples, we have

$$\begin{bmatrix}2 & -1\end{bmatrix}\begin{bmatrix}1 & 1\\ 2 & 2\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix}$$

with $\begin{bmatrix}1 & 1\\ 2 & 2\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix}$

and $\begin{bmatrix}14 & 6 & -13\end{bmatrix}\begin{bmatrix}1 & 5 & 3\\ 2 & -3 & -7\\ 2 & 4 & 0\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$

with $\begin{bmatrix}1 & 5 & 3\\ 2 & -3 & -7\\ 2 & 4 & 0\end{bmatrix}\begin{bmatrix}2\\ -1\\ 1\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$

Some of the more obvious cases of singularity are: proportionality of two rows (see above); proportionality of two columns; a zero row; or a zero column.

Example of singularity through a zero row:

$$\begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 3 & -5\\ 2 & 6 & 6\\ 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$$

with $\begin{bmatrix}-1 & 3 & -5\\ 2 & 6 & 6\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}-12\\ 1\\ 3\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$

It may sometimes happen, that a row of a matrix is almost proportional to some combination of the other rows. Such a system is known as ill-conditioned.

If A is singular, the system $A\underline{x} = \underline{b}$ cannot have a unique solution for $\underline{x}$, irrespective of the numerical content of $\underline{b}$. With an ill-conditioned system, we will have one of the rows being almost proportional to some combination of the other rows. In that case a unique solution of $\underline{x}$ does exist; but it is highly dependent on small variations in some elements of A and $\underline{b}$.

3.3 The elimination process

A non-singular system can be solved, by means of a series of successive elimination steps. (See also sections 1.1 and 1.2)

To this purpose, A and $\underline{b}$ are written in a tableau. The tableau

will be a composite matrix

$$T = \left[A \quad \underline{b} \right]$$

The elimination step considers each tableau as representing a new system. We can then indicate each previous tableau as T, and each next tableau as U.

A crucial role in the p^{th} elimination step is played by the element t_{pp}. At this stage, p-1 columns of A will be unit vectors $\underline{e}_j$ $(j = 1, 2, \ldots, p-1)$. The element t_{pp} is then the pivotal element or pivot. The finding of a non-zero pivot may require a permutation of rows. If A is non-singular, there should be at least one non-zero element t_{rp} $(p \leq r)$. We will then, if necessary, interchange the r^{th} and the p^{th} row.

Denoting the new tableau as U, we will then have:

$$\underline{u}'_p = t_{pp}^{-1} \underline{t}'_p$$

(divide the pivot-row through the pivot; or, what is the same: express the equation with a unity coefficient for x_p.)

And for the other rows of the tableau:

$$\underline{u}'_i = \underline{t}'_i - t_{i,p} \underline{u}'_p \qquad (i \neq p).$$

(Subtract the pivotal row, from the remaining rows, with such coefficients, as to cause vanishment of the remainder elements in the pivotal column; or, what is the same, eliminate x_p from the remainder equations, with help of the p^{th} equation.)

This procedure is repeated, until all columns of A (= the first n columns of T), have been transformed into unit vectors.

Example:

equations system:

$$\begin{aligned} .517\, x_1 &- .055\, x_2 - .293\, x_3 = .804 \\ -.077\, x_1 &+ .686\, x_2 - .010\, x_3 = .450 \\ -.056\, x_1 &- .022\, x_2 + .863\, x_3 = .750 \end{aligned}$$

$$T = \begin{bmatrix} .517 & -.055 & -.293 & .804 \\ -.077 & .686 & -.010 & .450 \\ -.056 & -.022 & .863 & .750 \end{bmatrix}$$

$$\underline{u}'_1 = \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \end{bmatrix} = (1/.517)\underline{t}'_1$$

$$.077\, \underline{u}'_1 = \begin{bmatrix} .077 & -.008 & -.044 & .120 \end{bmatrix}$$

$$.056\, \underline{u}'_1 = \begin{bmatrix} .056 & -.006 & -.032 & .087 \end{bmatrix}$$

$$\begin{array}{lcl} \underline{u}_1' & = & [1.000 \quad -.106 \quad -.567 \quad 1.555] \text{ (copied)} \\ \underline{u}_2' & = & [\quad .678 \quad -.054 \quad .570] = \underline{t}_2' + .077\ \underline{u}_1' \\ \underline{u}_3' & = & [\quad -.028 \quad .831 \quad .837] = \underline{t}_3' + .056\ \underline{u}_1' \end{array}$$

equations system:

$$\begin{array}{rcl} 1.000\ x_1 - .106\ x_2 - .567\ x_3 & = & 1.555 \\ .678\ x_2 - .054\ x_3 & = & .570 \\ -.028\ x_2 + .831\ x_3 & = & .837 \end{array}$$

$$T = \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \\ & .678 & -.054 & .570 \\ & -.028 & .831 & .837 \end{bmatrix}$$

$$\begin{array}{lcl} \underline{u}_2' & = & [\quad 1.000 \quad -.080 \quad .841] = (1/.678)\underline{t}_2' \\ .106\ \underline{u}_2' & = & [\quad .106 \quad -.008 \quad .089] \\ .028\ \underline{u}_2' & & [\quad .028 \quad -.002 \quad .024] \end{array}$$

$$\begin{array}{lcl} \underline{u}_1' & = & [1.000 \quad -.575 \quad 1.644] = \underline{t}_1' + .106\ \underline{u}_2' \\ \underline{u}_2' & = & [\quad 1.000 \quad -.080 \quad .841] \text{ (copied)} \\ \underline{u}_3' & = & [\quad .829 \quad .861] = \underline{t}_3' + .028\ \underline{u}_2' \end{array}$$

equations system:

$$\begin{array}{rcl} x_1 - .575\ x_3 & = & 1.644 \\ x_2 - .080\ x_3 & = & .841 \\ .829\ x_3 & = & .861 \end{array}$$

$$T = \begin{bmatrix} 1.000 & & -.575 & 1.644 \\ & 1.000 & -.080 & .841 \\ & & .829 & .861 \end{bmatrix}$$

$$\begin{array}{lcl} \underline{u}_3' & = & [\quad 1.000 \quad 1.039] \\ .575\ \underline{u}_3' & = & [\quad .575 \quad .597] \\ .080\ \underline{u}_3' & = & [\quad .080 \quad .083] \end{array}$$

$$\begin{array}{lcl} \underline{u}_1' & = & [1.000 \quad 2.241] = \underline{t}_1' + .575\ \underline{u}_3' \\ \underline{u}_2' & = & [\quad 1.000 \quad .924] = \underline{t}_2' + .080\ \underline{u}_3' \\ \underline{u}_3' & = & [\quad 1.000 \quad 1.039] \text{ (copied)} \end{array}$$

equations system:

$$\begin{array}{rcl} x_1 & = & 2.241 \\ x_2 & = & .924 \\ x_3 & = & 1.039 \end{array}$$

Verification:

$$\underset{A}{\begin{bmatrix} .517 & -.055 & -.293 \\ -.077 & .686 & -.010 \\ -.056 & -.022 & .863 \end{bmatrix}} \quad \underset{\underline{x}}{\begin{bmatrix} 2.241 \\ .924 \\ 1.039 \end{bmatrix}} = \underset{\underline{b}}{\begin{bmatrix} .804 \\ .450 \\ .750 \end{bmatrix}}$$

3.4 Computational arrangements

With an electronic computer, there is no need to store any other matrix, but T. With most computer codes, multiplication and subtraction can be done in one operation, and the new tableau is written over the old one, on the same place. For a human computor, such a procedure would quickly lead to errors. At the very least, a human computor will have to write out n successive tableaux T. More prudent (= less prone to errors), would be the writing of 2n -1 tableaux, each second tableau consisting of the intermediate vectors $t_{i,p}\,\underline{u}'_p$, together with the corresponding vector $\underline{u}'_p$. This was done in paragraph 3.3. In that example, the first row of every second tableau was the new pivot-row, $\underline{u}'_p$. In that way, computation of the successive vectors runs from the top of the page, to the bottom.

But this arrangement of the tableau is not the most orderly one. It would seem more logical, that $\underline{u}'_p$ should also be the p^{th} row of the intermediate tableau.

The intermediate tableau is of course a matrix in its own right, to be denoted as S.

$$S = \underline{e}_p\, t_{pp}^{-1}\, \underline{t}'_p - (\underline{t}_p - \underline{e}_p\, t_{pp})\, t_{pp}^{-1}\, \underline{t}'_p \qquad (3.4.1)$$

The two terms of (3.4.1) represent two separate expressions from the previous paragraph. The first term represents the new pivot-row, $u'_p = t_{pp}^{-1}\, t'_p$, together with a unit column, serving as an operator, in order to complete the row, with a set of zero rows, to a matrix of the required order. The second term represents the set of vectors $-t_{i,p}\,\underline{u}'_p$. Without the restriction $i \neq p$; this set of vectors would be $-\underline{t}_p\, t_{pp}^{-1}\, \underline{t}'_p$ (there would be n rows). Then $\underline{t}_p - \underline{e}_p\, t_{pp}$ is the pivotal column, with the pivotal element itself replaced by a zero.

Example: (previous paragraph, first elimination step)

$$S = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad 1.934 \quad \begin{bmatrix} .517 & -.055 & -.293 & .804 \end{bmatrix}$$

$$- \left(\begin{bmatrix} .517 \\ -.077 \\ -.056 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} .517 \right) \quad 1.934 \quad \begin{bmatrix} .517 & -.055 & -.293 & .804 \end{bmatrix}$$

$$= \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \end{bmatrix}$$

$$- \quad \begin{bmatrix} 0 \\ -.077 \\ -.056 \end{bmatrix} \quad \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \end{bmatrix}$$

$$= \quad \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \\ ---- & ---- & ---- & ---- \\ ---- & ---- & ---- & ---- \end{bmatrix}$$

$$+ \quad \begin{bmatrix} ---- & ---- & ---- & ---- \\ .077 & -.008 & -.044 & .120 \\ .056 & -.006 & -.032 & .087 \end{bmatrix}$$

$$= \quad \begin{bmatrix} 1.000 & -.106 & -.567 & 1.555 \\ .077 & -.008 & -.044 & .120 \\ .056 & -.006 & -.032 & .087 \end{bmatrix}$$

It is of course not suggested, that this elaborate calculation, should be written out in full at every iteration. One writes $\underline{u}'_p$ as the p^{th} row of S, and then writes out the remainder rows of S, in the reserved blank space.

Mathematically, it is also possible to simplify (3.4.1), quite a bit in fact:

$$S = \left(\underline{e}_p (1 + t_{pp}) - \underline{t}_p \right) t_{pp}^{-1} \underline{t}'_p \qquad (3.4.2)$$

Same example:

$$S = \left(\begin{bmatrix} 1.517 \\ --- \\ --- \end{bmatrix} - \begin{bmatrix} .517 \\ -.077 \\ -.056 \end{bmatrix} \right) 1.934 \begin{bmatrix} .517 & -.055 & -.293 & .804 \end{bmatrix}$$

The transformation of T and S, into the new tableau U = T, will then be:

$$U = (I - \underline{e}_p \underline{e}'_p)T + S \qquad (3.4.3)$$

The expression $(I - \underline{e}_p \underline{e}'_p)$, is again an operator; it is a unit matrix, from which the p^{th} diagonal unity element has been replaced by a zero. The whole operator serves to delete (replace by a zero row), the p^{th} row of T, the old pivot row.

3.5 The sum-count column

A special aspect of the arrangement of the tableau, for the purpose of manual computation, is the use of a sum-count of all the elements of a row. This will become an additional column of the tableau. The elements of the "check" column are the sum of all other elements in the same row. One can treat this column as any other column in the tableau. It is of course never transformed into a unit vector. All operations on the tableau involve: multiplication of a row with a certain number, and adding one row to another. These operations do not disturb the sum-count of the row. The "check" column remains the sum of the other columns.

Then, each time one has just computed a new row, one will verify the row-count. Is the element in the "check" column still the sum of the other elements in the row? If the difference is within the tolerance of a rounding-off error, one will adjust the elements in the "check" column. If a larger difference is found, a mistake has been made. That mistake can be:

a) in the sum-count itself- the row is correct after all.
b) in the newly computed row (do not forget the element in the "check" column).
c) in an earlier part of the computation; such a mistake remained undetected, either because the row-count was not verified at all, or because the answer was misread for the expected one.

The equation-system of section 3.3, could then be written and solved as follows:

x_1	x_2	x_3	=	check	
.517	-.055	-.293	.804	.973	= T
-.077	.686	-.010	.450	1.099	
-.056	-.022	.863	.750	1.535	
1.000	-.106	-.567	1.555	1.882	= S
.077	-.008	-.044	.120	.145	
.056	-.006	-.032	.087	.105	
1.000	-.106	-.567	1.555	1.882	= U = T
	.678	-.054	.570	1.194	(next iteration)
	-.028	.831	.837	1.640	
	.106	-.008	.089	.187	= S
	1.000	-.080	.841	1.761	
	.028	-.002	.024	.050	

$$\begin{bmatrix} 1.000 & & -.575 & 1.644 & 2.069 \\ & 1.000 & -.080 & .841 & 1.761 \\ & & .829 & .861 & 1.690 \end{bmatrix} = U = T \text{ (next iteration)}$$

$$\begin{bmatrix} & & .575 & .597 & 1.172 \\ & & .080 & .083 & .163 \\ & & 1.000 & 1.039 & 2.039 \end{bmatrix} = S$$

$$\begin{bmatrix} 1.000 & & & 2.241 & 3.241 \\ & 1.000 & & .924 & 1.924 \\ & & 1.000 & 1.039 & 2.039 \end{bmatrix} = U = T \text{ (end result)}$$

3.6 The system A $\underline{y}$ = B $\underline{x}$

Consider the equations-system:

$$.867\ y_1 - .066\ y_2 = .244\ x_1 + .150\ x_2 + .722\ x_3 + .556\ x_4$$

$$-.150\ y_1 + .850\ y_2 = .561\ x_1 + .173\ x_2 + .278\ x_3 + .222\ x_4$$

When both $\underline{y}$ and $\underline{x}$ are unknown, it will not be possible to "solve" the system, in the sense of finding a unique numerical value. But when A is square and non-singular, it will be possible to express $\underline{y}$ into $\underline{x}$. (Expression of $\underline{x}$ into $\underline{y}$ will be possible, only if B is square and non-singular.) The expression of $\underline{y}$ into $\underline{x}$ can be obtained, with the help of the elimination procedure, developed in paragraphs 3.3 -3.4.

Only, the tableau T will not just consist of a matrix A, and a vector, but of the matrix A, and another matrix B. The tableau will be a composite matrix, consisting of two block-columns; the matrices A and B, and of course there could be a "check" column.

For our numerical example, we will then have:

y_1	y_2	x_1	x_2	x_3	x_4	check	
.867	-.066	.244	.150	.722	.556	2.473	= T
-.150	.850	.561	.173	.278	.222	1.934	
1.000	-.076	.281	.173	.833	.641	2.852	= S
.150	-.011	.042	.026	.125	.096	.428	
1.000	-.076	.281	.173	.833	.641	2.852	= U = T
	.839	.603	.199	.403	.318	2.362	
	.076	.055	.018	.036	.029	.214	= S
	1.000	.719	.237	.480	.379	2.815	

$$\begin{bmatrix} 1.000 & & .336 & .191 & .869 & .670 & 3.066 \\ & 1.000 & .719 & .237 & .480 & .379 & 2.815 \end{bmatrix} = U = T$$

The outcome, $\underline{y} = \begin{bmatrix} .336 & .191 & .869 & .670 \\ .719 & .237 & .480 & .379 \end{bmatrix} \quad x$

is known as the reduced form of the system $A\underline{y} = B\underline{x}$. This assumes that A is square and non-singular. If the order of A is n by n, there will be n elements in $\underline{y}$. The number of variables in $\underline{x}$ can be any number, say m. Then B must be of order n by m. The order of the reduced form matrix will be the same.

Exercise

Check (by performing the calculation illustrated above), that the reduced form of the block-equation

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \underline{y} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \underline{x}, \text{ is correctly calculated as } \underline{y} = \begin{bmatrix} 4 & 6 \\ 7 & 10 \end{bmatrix} \underline{x}$$

3.7 The inverse

Consider the following equations-system:

$$x_1 + 2x_2 + 3x_3 = y_1$$

$$x_1 + x_2 + 2x_3 = y_2$$

$$x_1 + x_2 + x_3 = y_3$$

Or, more generally,

$$A\underline{x} = \underline{y} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3.7.1)$$

where A is a square and non-singular matrix; $\underline{x}$ and $\underline{y}$ unknown vectors of the same order as the matrix.

Like the system $A\underline{y} = B\underline{x}$ discussed in the previous paragraph, this system cannot be "solved", in the sense of evaluating $\underline{x}$. Both $\underline{x}$ and $\underline{y}$ are unknown vectors. But it is possible, to infer from (3.7.1), a matrix expression for $\underline{x}$ in $\underline{y}$, of the type

$$\underline{x} = B\underline{y} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3.7.2)$$

This is done by performing the elimination process, described in the previous paragraphs:

x_1	x_2	x_3	y_1	y_2	y_3	check	
1	2	3	1			7	T
1	1	2		1		5	
1	1	1			1	4	
1	2	3	1			7	S
-1	-2	-3	-1			-7	
-1	-2	-3	-1			-7	
1	2	3	1			7	U = T
	-1	-1	-1	1		-2	
	-1	-2	-1		1	-3	
	-2	-2	-2	2		-4	S
	1	1	1	-1		2	
	1	1	1	-1		2	
1		1	-1	2		3	U = T
	1	1	1	-1		2	
		-1		-1	1	-1	
		-1		-1	1	-1	S
		-1		-1	1	-1	
		1		1	-1	1	
1			-1	1	1	2	U = T
	1		1	-2	1	1	
		1		1	-1	1	

And we have found:

$$\begin{bmatrix} -1 & 1 & 1 \\ 1 & -2 & 1 \\ - & 1 & -1 \end{bmatrix} \quad \underline{y} = \underline{x}$$

In this form, we have a new system

$$B\,\underline{y} = \underline{x} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (3.7.3)$$

We may perform the same operation again, to obtain an expression of $\underline{x}$ in $\underline{y}$. But we already know that expression, since $A\,\underline{x} = y$ was given in the first place, by (3.7.1). The relationship of A and B, by (3.7.1) and (3.7.3), is named an inverse one. The matrix B is the inverse of A. This relationship is written in symbolic matrix notation as $B = A^{-1}$, but we still need to give a formal definition.

The following questions need an answer:

Can the operation as indicated earlier in this section (known as inversion by row operations) always be performed? And if not, what can be said about the class of matrices on which it can be applied? Are there any matrices for which inversion by row operations is not possible, but for which the relation indicated in this example may nevertheless exist?

To answer these questions, we begin by stating a somewhat more formal framework.

Definition

Two square matrices A and B (of equal order n by n), are said to satisfy the inverse relations for $\underline{x}$ and $\underline{y}$, if they satisfy

$$A\underline{x} = \underline{y} \quad (3.7.1) \text{ and } B\underline{y} = \underline{x} \quad (3.7.3).$$

The relations $A\underline{x} = \underline{y}$ and $B\underline{y} = \underline{x}$ may then themselves be indicated as the (vectoral) inverse relations.

This definition does not actually require that $\underline{x}$ and $\underline{y}$ are non-zero vectors, but we would normally think in terms of the non-trivial case $\underline{x} \neq 0$, $\underline{y} \neq 0$, as otherwise no requirement is stated.

Definition

The square matrix B is said to be the inverse of another square matrix A (of the same order), if

$A\underline{x} = \underline{y}$ implies $B\underline{y} = \underline{x}$, for all vectors $\underline{x}$
(the inverse relations to hold for all x).

Note that two matrices may satisfy the inverse relations for one vector $\underline{x}$ and corresponding $\underline{y}$ but not for another vector $\underline{x}$.

Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 2 & - \\ 1 & -1 & - \\ - & 2 & -2 \end{bmatrix}, \quad \underline{x} = \begin{bmatrix} 1 \\ - \\ - \end{bmatrix}, \quad \underline{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ - \\ - \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} -1 & 2 & - \\ 1 & -1 & - \\ - & 2 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ - \\ - \end{bmatrix}$$

(the inverse relations),

but for $\underline{x} = \begin{bmatrix}1\\1\\1\end{bmatrix}$, they do not apply

$$\begin{bmatrix}1 & 2 & 3\\1 & 1 & 2\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}6\\4\\3\end{bmatrix}, \quad \begin{bmatrix}-1 & 2 & -\\1 & -1 & -\\- & 2 & -2\end{bmatrix}\begin{bmatrix}6\\4\\3\end{bmatrix} \neq \begin{bmatrix}1\\1\\1\end{bmatrix}$$

Some properties of inverses and non-singular matrices:

Firstly, the definition of the inverse matrix says that the inverse relations are to hold for <u>all</u> vectors $\underline{x}$.

That includes unit vectors $\underline{x} = \underline{e}_j$ (j = 1, ... n).

Hence, if B is the inverse of A

$$A\,I = A \text{ implies } BA = I$$

A matrix premultiplied by its inverse, yields the unit matrix as product

$$A^{-1}A = I \qquad (3.7.4)$$

This property is an obvious practical test in checking whether a calculated inverse has been correctly calculated.

Example

$$A = \begin{bmatrix}1 & 2 & 3\\1 & 1 & 2\\1 & 1 & 1\end{bmatrix}, \quad A^{-1} = \begin{bmatrix}-1 & 1 & 1\\1 & -2 & 1\\- & 1 & -1\end{bmatrix} \text{ (calculated above)}$$

$$\begin{bmatrix}-1 & 1 & 1\\1 & -2 & 1\\- & 1 & -1\end{bmatrix}\begin{bmatrix}1 & 2 & 3\\1 & 1 & 2\\1 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & - & -\\- & 1 & -\\- & - & 1\end{bmatrix}$$

But if by error, the minus sign is omitted from the top lefthand element of the "inverse" we have

$$\begin{bmatrix}1 & 1 & 1\\1 & -2 & 1\\- & 1 & -1\end{bmatrix}\begin{bmatrix}1 & 2 & 3\\1 & 1 & 2\\1 & 1 & 1\end{bmatrix} = \begin{bmatrix}3 & 4 & 6\\- & 1 & 1\\- & - & 1\end{bmatrix} \neq \begin{bmatrix}1 & - & -\\- & 1 & -\\- & - & 1\end{bmatrix}$$

The following property is stated here without proof at this stage.

<u>Every non-singular matrix has an inverse</u>, (a proof of this statement follows in section 5.5).

Furthermore, from (3.7.4) we obtain, upon premultiplication by a row-vector $\underline{y}'$, of appropriate order:

$$\underline{y}' A^{-1} A = \underline{y}' \qquad (3.7.5)$$

The left-hand side expression of (3.7.5) can be non-zero, only if $\underline{y}' A^{-1}$ is non-zero, hence $\underline{y}' \neq 0$ implies $\underline{y}' A^{-1} \neq 0$, i.e. all inverses are non singular

Therefore, if (as will be shown in section 5.5), all non-singular matrices have an inverse, all inverses have an inverse.

Pre-multiplication of (3.7.4) by the inverse of A^{-1} and re-application of (3.7.4) for the inverse results in:

$$(A^{-1})^{-1} A^{-1} A = \left[(A^{-1})^{-1} A^{-1}\right] A = A = (A^{-1})^{-1} \qquad (3.7.6)$$

The inverse of a matrix, is the matrix itself

Since all inverses are non-singular we have the following corollary:

A singular matrix does not have an inverse

Furthermore, direct application of (3.7.4), for the inverse and substitution of A for $(A^{-1})^{-1}$ yields

$$A A^{-1} = I \qquad (3.7.7)$$

The product of a matrix and its inverse, is a unit matric irrespective of the use of pre- or postmultiplication.

We have so far spoken of "the" inverse, without actually proving that such a matrix is unique. In fact inverses are unique functions of the inverted matrices, and we have the following:

Theorem

If B and C are both square and of the same order as A, and both the inverse of A, they are equal to each other

Proof

If both B and C are inverses of A, this implies, according to (3.7.7),

$$A B - A C = I - I = [0] \qquad (3.7.8)$$

Since A has an inverse we may require, for any $\underline{z}'$, $\underline{y}' = \underline{z}' A^{-1}$, regardless of whether that inverse is B, C or possibly a third inverse of A.

Therefore

$$\underline{y}' [A B - A C] = \underline{z}' [B - C] \quad \text{(all } \underline{z}')$$

This condition can only be satisfied for all $\underline{z}'$, if B - C is a zero matrix, the left-hand side being zero by (3.7.8).
q.e.d.

Theorem

If, for some square matrices A, B and X of equal order, X being non-singular

$$A B X = X \tag{3.7.9}$$

is true

Then

Irrespective of what is initially assumed about the (non) singularity of A and B, A and B are each other's inverses

Proof:

Denote, for any $\underline{c}$, $X^{-1}\underline{c}$ as $\underline{z}$

Postmultiply (3.7.9) by $\underline{z} = X^{-1}\underline{c}$ to obtain

$$A B X X^{-1}\underline{c} = A B \underline{c} = X X^{-1} \underline{c} = \underline{c} \tag{3.7.10}$$

Re-name $\underline{c}$ ans $\underline{x}$ and $B\underline{c}$ as $\underline{y}$, and we see that

$$B \underline{c} = B \underline{x} = \underline{y} \text{ implies } A B \underline{c} = A \underline{y} = \underline{c} = \underline{x}$$

From which

$$B \underline{x} = \underline{y} \text{ implies } A \underline{y} = \underline{x} \quad \text{(all } \underline{x})$$

Showing that A is the inverse of B
q.e.d.

In terms of recognizing inverses, the most widely used application of this theorem is that of the unit matrix product property.

$$A B = I \tag{3.7.11}$$

not only follows from the inverse relationship - i.e. (3.7.4) and (3.7.7) but also proves it.

We now come back to the question of invertability by row operations. If the algorithm of inversion by row operations is understood as "pivoting" along the main diagonal, then there are non-singular matrices, which cannot be inverted in that way.

Example

$$A = \begin{bmatrix} - & 1 \\ 1 & - \end{bmatrix}$$

This is a non-singular matrix. Its inverse exists, it is in fact A itself as may be shown by verifying the unit-matrix product property

$$\begin{bmatrix} - & 1 \\ 1 & - \end{bmatrix} \begin{bmatrix} - & 1 \\ 1 & - \end{bmatrix} = \begin{bmatrix} 1 & - \\ - & 1 \end{bmatrix}$$

But plainly this matrix cannot be inverted along the main diagonal. However in section 3.10 we will develop an adaptation of the algorithm of inversion by row-operations, and that adaptation is capable of inverting all non-singular matrices.

Exercise

Verify (by inversion by row operations, as well as by pre- and post-multiplication) that the inverse of

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 3 & 2 & 1 \end{bmatrix} \text{ is correctly evaluated as } \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 3 & 2 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} -1 & 1 & - \\ 1 & -2 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

NB This matrix may be inverted by row-operations, pivots being taken on the main diagonal.

3.8 Inverse and reduced form

Consider again the equation-system:

$$A \underline{y} = B \underline{x} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3.8.1)$$

It is assumed, that A is a square and non-singular matrix; no similar assumption need to be made about B. The first order parameter, the number of rows of B, must be equal to the order of A; otherwise the expression $A \underline{y} = B \underline{x}$ would not be legitimate.

Now pre-multiply (3.8.1) with A^{-1}, the inverse of A:

$$A^{-1} \cdot A\,\underline{y} = A^{-1} \cdot B\,\underline{x}, \text{ or}$$

$$\underline{y} = A^{-1} B\,\underline{x} \quad \ldots\ldots\ldots\ldots\ldots\ldots \qquad (3.8.2)$$

The matrix $A^{-1} B$, and the corresponding block-equation (3.8.2), is the reduced form of (3.8.1). Despite the apparent symmetry between left- and right-hand side of (3.8.1), it does not follow, that the expression $B^{-1} A$ is meaningful. No assumption about non-singularity of B was made; the number of columns in B may even be different from the number of rows.

Example

(See also section 3.6) Compute the inverse of A.

y_1	y_2	z_1	z_2	check	
.867	-.066	1.000		1.801	T
-.150	.850		1.000	1.700	
1.000	-.076	1.153		2.077	S
.150	-.010	.173		.313	
1.000	-.076	1.153		2.077	U = T
	.840	.173	1.000	2.013	
	.076	.015	.091	.182	S
	1.000	.208	1.192	2.400	
1.000		1.168	.091	2.259	
	1.000	.208	1.192	2.400	

$$A^{-1} = \begin{bmatrix} 1.168 & .091 \\ .208 & 1.192 \end{bmatrix} \qquad B = \begin{bmatrix} .244 & .150 & .722 & .556 \\ .561 & .173 & .278 & .222 \end{bmatrix}$$

$$A^{-1} B = \begin{bmatrix} .336 & .190 & .868 & .670 \\ .719 & .237 & .480 & .379 \end{bmatrix}$$

the same result as obtained in paragraph 6 by direct operation on the rows of the full tableau.

3.9 Multiplication instead of division:
the all-integer elimination method

This method is useful, in case the initial coefficients matrix, which defines a block-equation, consists of integer numbers only.

The "normal" elimination procedure, is related to algebraic substitution; it starts with expressing a particular equation and the corresponding pivotal row, in a particular variable, with a unity coefficient.

Instead of dividing the pivotal row by the pivotal element, one may also multiply the rest of the tableau by that same element, avoiding the use of fractional numbers.

The usual procedure of subtracting an outer product is then performed with the old pivot-row and the old pivot-column, the outer product of which subtracted from the rest of the new tableau.

One may avoid the unnecessary generation of too large figures, by a later division of most of the tableau, by the pivotal element, after the next elimination step.

Example

Let us assume, we wish to invert the matrix

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 11 & -7 & 8 \\ -5 & 3 & 1 \end{bmatrix}$$

or equivalent, solve the system

$$\begin{aligned} 2\,y_1 + 5\,y_2 + y_3 &= z_1 \\ 11\,y_1 - 7\,y_2 + 8\,y_3 &= z_2 \\ -5\,y_1 + 3\,y_2 + y_3 &= z_3 \end{aligned}$$

with respect to y

We now proceed as follows:

Initial tableau

y_1	y_2	y_3	=	z_1	z_2	z_3	$\sum$	Ratio
2	5	1		1			9	1
11	-7	8			1		13	1
-5	3	1				1	-	1

Multiply rows 2 and 3 both by $t_{11} = 2$, to obtain

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio	
2	5	1	1			9	2	(not divided)
22	-14	16		2		26	2)	
-10	6	2			2	-	2)	multiplied

Subtract 11 times row 1 from row 2 and add 5 times row 1 to row 3:

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
2	5	1	1			9	2
-	-69	5	-11	2		-73	2
-	31	7	5		2	45	2

Multiply rows 1 and 3 by -69:

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio	
-138	-345	-69	-69			-621	-138	
	-69	5	-11	2		-73	-69	(diagonal cell)
	-2139	-483	-345		-138	-3105	-138	

Subtract 5 times row 2 from row 1 and 31 times row 2 from row 3:

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
-138		-94	-14	-10		-256	-138
	-69	5	-11	2		-73	-69
		-638	-4	-62	-138	-842	-138

At this point, we note that both rows 1 and 3 admit for division by 2, without losing their all-integer nature. This is systematic, and related to the "ratio" column. This column gives the ratios between the rows of the tableaux developed the all-integer method and the corresponding "fractional" tableaux as used so far. For reasons to be explained in section 5.8, it is always possible to "scale down" this column to the entry in the pivotal row.

Accordingly, we now write a new tableau, we divide row 1 and row 3 by 2

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
-69		-47	-7	-5		-128	-69
	-69	5	-11	2		-73	-69
		-319	-2	-31	-69	-421	-69

We now again start the third step with multiplication: we multiply rows 1 and 2 with -319

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
22011		14993	2233	1595		40832	22011
	22011	-1595	3509	-638		23287	22011
		-319	-2	-31	-69	-421	-319

Add 47 times row 3 to row 1 and subtract 5 yimes row 3 from row 2 to obtain

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
22011			2139	138	-3243	21045	22011
	22011		3519	-483	345	25392	22011
		-319	-2	-31	-69	-421	-319

Divide rows 1 and 2 by -69 and obtain:

y_1	y_2	y_3	z_1	z_2	z_3	$\sum$	Ratio
-319			-31	-2	47	-305	-319
	-319		-51	7	-5	-368	-319
		-319	-2	-31	-69	-421	-319

The inverse may now be obtained as:

$$A^{-1} = \frac{1}{-319}\begin{bmatrix} -31 & -2 & 47 \\ -51 & 7 & -5 \\ -2 & -31 & -69 \end{bmatrix} = \begin{bmatrix} 0.097 & 0.006 & -0.147 \\ 0.160 & -0.022 & 0.016 \\ 0.006 & 0.097 & 0.216 \end{bmatrix}$$

3.10 Row-permutation during inversion

In section 3.3 we mentioned the possibility that a system of the specification $A\underline{x} = \underline{b}$ may need re-arrangement of the rows of A, in order to find a non-zero pivot.

This is true for inversion by row-operations in general. We will illustrate this problem in connection with the calculation of an inverse

Example

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 0 \\ 2 & 1 & -2 \end{bmatrix}$$

One way to find out that an inverse exists, is to calculate it.

We take the equations-system

$$\begin{aligned} x_2 + 2x_3 &= y_1 \\ x_1 - x_2 &= y_2 \\ 2x_1 + x_2 - 2x_3 &= y_3 \end{aligned}$$

We interchange the first two equations

$$\begin{aligned} x_1 - x_2 &= y_2 \\ x_2 + 2x_3 &= y_1 \\ 2x_1 + x_2 - 2x_3 &= y_3 \end{aligned}$$

And proceed as before.

Eliminate x_1:

$$\begin{aligned} x_1 - x_2 &= y_2 \\ x_2 + 2x_3 &= y_1 \\ 3x_2 - 2x_3 &= -2y_2 + y_3 \end{aligned}$$

Eliminate x_2:

$$\begin{aligned} x_1 + 2x_3 &= y_1 + y_2 \\ x_2 + 2x_3 &= y_1 \\ -8x_3 &= -3y_1 - 2y_2 + y_3 \end{aligned}$$

Eliminate x_3:

$$\begin{aligned} x_1 &= \tfrac{1}{4}y_1 + \tfrac{1}{2}y_2 + \tfrac{1}{4}y_3 \\ x_2 &= \tfrac{1}{4}y_1 - \tfrac{1}{2}y_2 + \tfrac{1}{4}y_3 \\ x_3 &= \tfrac{3}{8}y_1 + \tfrac{1}{4}y_2 - \tfrac{1}{8}y_3 \end{aligned}$$

and we have established the inverse relationship

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 0 \\ 2 & 1 & -2 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ \frac{3}{8} & \frac{1}{4} & -\frac{1}{8} \end{bmatrix}$$

3.11 Block-Elimination

Consider the following partitioned system

$$\begin{aligned} A_{11}\,\underline{x}_1 + A_{12}\,\underline{x}_2 &= B_1\,\underline{y} \\ A_{21}\,\underline{x}_1 + A_{22}\,\underline{x}_2 &= B_2\,\underline{y} \end{aligned} \qquad (3.11.1)$$

where A_{11} is a square and non-singular block of the full matrix A.

We pre-multiply the first block-equation in (3.11.1) by A_{11}^{-1}, to obtain

$$\underline{x}_1 + A_{11}^{-1} A_{12}\,\underline{x}_2 = A_{11}^{-1} B_1\,\underline{y} \qquad (3.11.2)$$

Re-ordering of the terms in (3.11.2) yields a reduced form expression for $\underline{x}_1$ in terms of $\underline{x}_2$ and y

$$\underline{x}_1 = A_{11}^{-1} B_1\,\underline{y} - A_{11}^{-1} A_{12}\,\underline{x}_2 \qquad (3.11.3)$$

Substitution of this reduced form expression for $\underline{x}_1$ into the second block-equation of (3.11.1) yields:

$$\left[A_{22} - A_{21} A_{11}^{-1} A_{12}\right]\underline{x}_2 = \left[B_2 - A_{21} A_{11}^{-1} B_1\right]\underline{y} \qquad (3.11.4)$$

We now combine (3.11.4) with (3.11.2), to obtain:

$$\begin{aligned} \underline{x}_1 + A_{11}^{-1} A_{12}\,\underline{x}_2 &= A_{11}^{-1} B_1\,\underline{y} \\ \left[A_{22} - A_{21} A_{11}^{-1} A_{12}\right]\underline{x}_2 &= \left[B_2 - A_{21} A_{11}^{-1} B_1\right]\underline{y} \end{aligned} \qquad (3.11.5)$$

If we now compare (3.11.1) with (3.11.5) we note a remarkable similarity between the elimination process which we found applicable to a system of equations in numbers and a system of block-equations in blocks and partitioned vectors.

We might as well write the operation down in a tabular arrangement, as follows:

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}$
A_{11}	A_{12}		B_1
A_{21}	A_{22}		B_2

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}$
I	$A_{11}^{-1}A_{12}$		$A_{11}^{-1}B_1$
	$A_{22}-A_{21}A_{11}^{-1}A_{12}$		$B_2-A_{21}A_{11}^{-1}B_1$

The rules for block-elimination are very similar to the ones which we found applicable for the solution of a system of ordinary equations by elementary row-operations.

Recall the rules for ordinary elimination

a) Find, in the pivot-column a non-zero element in a row corresponding to a not yet used equation

b) If necessary, permute rows, so as to have this pivotal element on the diagonal

c) Multiply the pivotal row by the reciprocal of the diagonal pivotal element

d) Add a multiple of the updated pivotal row to the other rows, with the negative of the relevant element in the pivotal column as multiplier, thereby causing the element in the pivotal column to become zero

The similar rules for block-inversion are:

a) Find in the block-pivot column a square and non-singular block, in a block-row corresponding to a not yet used block-equation

b) If necessary permute block-rows, so as to have this pivotal block on the diagonal

c) Pre-multiply the pivotal block-row by the inverse of the pivotal diagonal block

d) Add the updated pivotal block-row, pre-multiplied by minus the relevant block in the pivotal block-column, to the other rows, thus causing blocks in the pivotal columns to vanish.

Block-elimination, or block-pivoting is useful, in particular if a partitioned matrix has zero or unit matrix blocks.

Example

"Calculate" the inverse of:

$$M = \left[\begin{array}{c|c} I & A_{12} \\ \hline A_{21} & \end{array}\right]$$

where both $A_{2,1}$ and $A_{1,2}$ are square and non-singular.

The partitioned form of the block-equation

$$M\,\underline{x} = \underline{y} \qquad (3.11.7)$$

is tabulated and we perform the elimination as follows:

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}_1$	$\underline{y}_2$
(I)	A_{12}		I	
A_{21}				I

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}_1$	$\underline{y}_2$
I	A_{12}		I	
	$(-A_{21}A_{12})$		$-A_{21}$	I

$\underline{x}_1$	$\underline{x}_2$	$\underline{y}_1$	$\underline{y}_2$
I			A_{21}^{-1}
	I	A_{12}^{-1}	$-A_{12}^{-1}A_{21}^{-1}$

It is now readily verified that

$$\left[\begin{array}{c|c} I & A_{12} \\ \hline A_{21} & \end{array}\right]^{-1} = \left[\begin{array}{c|c} & A_{21}^{-1} \\ \hline A_{12}^{-1} & -A_{12}^{-1}A_{21}^{-1} \end{array}\right] \qquad (3.11.8)$$

N.B. $[A_{2,1} \,.\, A_{1,2}]^{-1} = A_{1,2}^{-1}\, A_{2,1}^{-1}$, as will be shown in the next section.

3.12 Inversion of recursive products and transposes

The inversion of a recursive product of several matrices gives rise to a remarkable change in their ordering.

If A and B are square and non-singular matrices of the same order, we have the following formula

$$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1} \qquad (3.12.1)$$

i.e. we write the inverses of the individual matrices, but in inverse order.

To show that (3.12.1) is indeed the correct expression for the inverse of A . B, we only need to multiply $B^{-1}\ A^{-1}$ by AB.

This proof can be given either by pre-multiplication, or by post-multiplication. We give both:

By pre-multiplication of $B^{-1}\ A^{-1}$ by AB:

$$A \cdot B \cdot B^{-1} \cdot A^{-1} = A \cdot A^{-1} = I \qquad (3.12.2)$$

The obtainment of the unit matrix as the product is (one of the) definitions of the inverse, therefore (3.12.2) shows that A . B and $B^{-1}\ A^{-1}$ are each others inverses. Similarly, post-multiplication of $B^{-1}\ A^{-1}$ by AB gives:

$$B^{-1} \cdot A^{-1} \cdot A \cdot B = B^{-1} \cdot B = I \qquad (3.12.3)$$

Example

Calculate the inverse of

$$A \cdot B = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 10 \\ 3 & 10 & 26 \end{bmatrix}$$

Manual calculation of the inverse of a 3 by 3 matrix which is full of non-zero elements would take some time. The triangular matrices are rather quicker to invert, especially because computation is simplified by the diagonal unity-elements

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 5. & -4 & 1 \end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\\ 0 & 1 & 4\\ 0 & 0 & 1\end{bmatrix}^{-1} = \begin{bmatrix}1 & -2 & 5\\ 0 & 1 & -4\\ 0 & 0 & 1\end{bmatrix}$$

and we establish the inverse we were looking for as:

$$\left[\begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 3 & 4 & 1\end{bmatrix}\begin{bmatrix}1 & 2 & 3\\ 0 & 1 & 4\\ 0 & 0 & 1\end{bmatrix}\right]^{-1} = \begin{bmatrix}1 & -1 & 5\\ 0 & 1 & -4\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ -2 & 1 & 0\\ 5 & -4 & 1\end{bmatrix} =$$

$$\begin{bmatrix}30 & -22 & 5\\ -22 & 17 & -4\\ 5 & -4 & 1\end{bmatrix}$$

The correctness of this result is shown by verification of the inverse relationship

$$\begin{bmatrix}1 & 2 & 3\\ 2 & 5 & 10\\ 3 & 10 & 26\end{bmatrix}\begin{bmatrix}30 & -22 & 5\\ -22 & 17 & -4\\ 5 & -4 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

The example also suggests another relationship

$$\left[A'\right]^{-1} = \left[A^{-1}\right]' \qquad (3.12.4)$$

i.e. the inverse of the transpose of a matrix, is the transpose of its inverse. Once we establish

$$\begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 3 & 4 & 1\end{bmatrix}^{-1} = \begin{bmatrix}1 & 0 & 0\\ -2 & 1 & 0\\ 5 & -4 & 1\end{bmatrix}$$

the inverse relationship

$$\begin{bmatrix}1 & 2 & 3\\ 0 & 1 & 4\\ 0 & 0 & 1\end{bmatrix}^{-1} = \begin{bmatrix}1 & -2 & 5\\ 0 & 1 & -4\\ 0 & 0 & 1\end{bmatrix}$$

can be written down without further calculation.

That this is indeed so, is shown by writing the normal block-equation system

$$A\underline{x} = \underline{y} \qquad (3.7.1)$$

in transpose form, i.e.

$$\underline{x}'\,A' = \underline{y}' \qquad (3.12.5)$$

Post-multiplication of (3.12.5) by the inverse of A' yields

$$\underline{x}' A' [A']^{-1} = \underline{x}' = \underline{y}' [A']^{-1} \qquad (3.12.6)$$

Transposition of (3.12.6) will give us:

$$\underline{x} = ([A']^{-1})' \underline{y} \qquad (3.12.7)$$

where $([A']^{-1})'$ is the transpose of the inverse of A'.

However (3.7.1) inverted directly, gives us:

$$\underline{x} = A^{-1} \underline{y} \qquad (3.12.8)$$

Since A is square and non singular, there is one and only one inverse of A. But (3.12.7) shows $([A']^{-1})'$ to be the inverse of A. Therefore we conclude

$$([A']^{-1})' = A^{-1} \qquad (3.12.9)$$

q.e.d.

3.13 The differentiation of an inverse

From the inverse relationship $A A^{-1} = I$, we obtain (see section 2.18 for the rules for differentiating the product of two matrices)

$$dA\, A^{-1} + A\, dA^{-1} = [0] \qquad (3.13.1)$$

where the symbol $[0]$ indicates a null matrix. From which we obtain after re-ordering and pre-multiplication by A^{-1}.

$$dA^{-1} = - A^{-1} dA\, A^{-1} \qquad (3.13.2)$$

Just as in the case of differentiating the power of a matrix a more simple formula which is akin to a scalar expression, arises in the case of a symmetric matrix i.e.

$$dA^{-1} = - (A^{-1})^2 dA \qquad (3.13.3)$$

may be written as

$$dA^{-1} = - A^{-2} dA \qquad (3.13.4)$$

by analogy to $dx^{-1} = -x^{-2} dx$.

The most obvious application of (3.13.2) arises in case of uncertainty concerning the precise magnitude of the elements of A.

For example,

some of the elements of A may be obtained as a result of some kind of statistical estimation procedure, which also gives confidence intervals (uncertainty margins). Thus, A might be,

$$A = \begin{bmatrix} 1 & 0.50 \\ 0.41 & 1 \end{bmatrix} \pm \begin{bmatrix} - & 0.10 \\ 0.10 & - \end{bmatrix}$$

If A has the estimated content, we find

$$A^{-1} = \begin{bmatrix} 1.25 & -0.63 \\ -0.63 & 1.25 \end{bmatrix}$$, therefore, a first order

approximation of ΔA^{-1} is

$$\Delta A^{-1} = \begin{bmatrix} 1.25 & -0.63 \\ -0.63 & 1.25 \end{bmatrix} \Delta A \begin{bmatrix} 1.25 & -0.63 \\ -0.63 & -1.25 \end{bmatrix}$$

where ΔA may be any matrix of which the diagonal elements are zero, and the off diagonal elements up to 0.10 in absolute value.

Exercise

Derive a differential expression for the reduced form matrix $A^{-1} B$.

3.14 Text of an Inversion Procedure

The programmed procedure offered in this section computes a reduced form, as defined in Section 3.6.

The process of computation is substantially the elimination process outlined in sections 3.3 and 3.4.

The following points may now be elucidated here. Firstly, as indicated by the 'COMMENT' in the relevant loop, we may get the inverse as a reduced form if we put a unit-matrix as right-hand side. (See also Section 3.7).

In Section 3.10 we discussed the issue of zero elements on the main diagonal. When performing manual computations, it is practical to restrict the permutation of rows to the condition that a zero is found on the main diagonal.

The programmed procedure always interchanges rows, even if this may turn out to be a trivial operation. The loop opening with 'COMMENT' FIRST FIND A NON ZERO PIVOT; amounts to a search for the absolute largest element in the pivotal column, on or below the main diagonal.

If that search comes up with something which doesn't significantly differ from zero, the matrix is assumed to be singular and the inversion is abandoned. Otherwide the absolute largest element is found in the row with index r, and the diagonal row has the index k, the index of the pivotal column.

The operation just below the label PERMUTE:

amounts to interchanging the k^{th} and r^{th} row. That operation is trivial, if the element which is largest in absolute value is found on the main diagonal.

The actual elimination step is performed just below the label UPDATE: At this point there is a difference of substance with the procedure outlined in the earlier sections of this chapter. It so happens that no further computational use is made of the pivotal column, nor of any column with lower index, to the left of the pivotal column, once the elimination step has been made.

Since the end-result is purely in terms of the right-hand side, the updating of the pivotal-column and the rest of the tableau further to the left, is suppressed. There is no point in calculating numbers which are not required as results. The interpretation of the calculation-tableau as a system of equations on the lines of sections 1.1 and 3.3 does not hold for such a partially updated tableau. Nor does the sum count property (see Section 3.5), hold for a partially updated tableau. For these reasons, the use of this short-cut is not recommended for manual calculation.

```
'PROCEDURE' INVE(A,M,N);
'VALUE' N; 'ARRAY' A; 'INTEGER' M,N;
'BEGIN' 'INTEGER' I,J,R,K; 'REAL' P,NUM;
  'COMMENT'
  INVERSION BY ELEMENTARY ROW-OPERATIONS;

  'IF' N=0 'THEN' 'BEGIN'
     'COMMENT'
     N EQUAL ZERO MEANS THAT CALCULATION OF THE INVERSE
     RATHER THAN THE REDUCED FORM IS ASKED.
     THEREFORE WE EQUATE N TO M AND THE RIGHTHANDSIDE MATRIX
     TO THE UNIT MATRIX;
```

```
    N:=M;
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
      'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO' A[I,M+J]:=0;
      A[I,M+I]:=1; 'END'; 'END';

'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    'COMMENT' FIRST FIND A NON-ZERO PIVOT;
    P:=0;
    'FOR' I:=R 'STEP' 1 'UNTIL' M 'DO'
    'IF' ABS(A[I,R]) > ABS(P)
    'THEN' 'BEGIN' K:=I; P:=A[I,R] 'END';
    'IF' ABS(P) < 0.0001 'THEN' 'BEGIN'
      'COMMENT'
      IF THIS LOOP IS GONE THROUGH THE MATRIX HAS BEEN
      FOUND SINGULAR; 'GOTO' SINGULAR; 'END';

    PERMUTE:
    'FOR' J:=R 'STEP' 1 'UNTIL' M+N 'DO' 'BEGIN'
      NUM:=A[R,J]; A[R,J]:=A[K,J]; A[K,J]:=NUM; 'END';

    UPDATE:
    'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO' A[R,J]:=A[R,J]/P;
    'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M 'DO'
    'IF' 'NOT' A[I,R] = 0 'THEN'
    'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO'
    A[I,J]:=A[I,J] - A[R,J]*A[I,R];  'END';
'GOTO' END OF INVE;

SINGULAR:
NEWLINE(1);
WRITETEXT('('SINGULAR%WITH%ORDER')');
WRITE(30,FORMAT('('SNDDDD')',M);
END OF INVE:      'END';
```

CHAPTER IV

SOME OPERATORS AND THEIR USE

4.1 The summation vector

We met the term operator already in Section 2.15 (and by way of casual use before definition already in Section 2.6). An operator is a matrix, or a vector, which is not defined for the sake of the numerical information given by its coefficients, but in order to perform some kind of transformation on another matrix or vector. For example, we may define a summation vector $\underset{\sim}{s}_n$, of n unity element,

$$\underset{\sim}{s}_n = \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$$

If A is of the order m by n, $A\,\underset{\sim}{s}_n$ is the column-vector of the row-totals of A, and $\underset{\sim}{s}'_m A$ is the row-vector of the column-totals.

For example, we may have an input-output table* matrix

$$T = \begin{bmatrix} T'_1 \\ T'_2 \end{bmatrix} = \begin{bmatrix} T_1 & T_2 \end{bmatrix}$$

where the first block-row and block-column refer to the producing sectors. The input-output convention that column total equals row-total is then expressed as

$$\begin{bmatrix} T_1 \end{bmatrix}' \underset{\sim}{s}_n = T'_1 \underset{\sim}{s}_n$$

where $\begin{bmatrix} T_1 \end{bmatrix}'$ is the transpose of the first block column T_1 and T'_1 is the first block-row,

---

*See my book Forecasting Models for National Economic Planning [19], Chapter III.

e.g.

$$T = \begin{bmatrix} 40 & 20 & 50 & 20 & 130 & 50 & -10 \\ 45 & 45 & 115 & 23 & 50 & 20 & 2 \\ 50 & 30 & 40 & 10 & - & 20 & -10 \\ 30 & 30 & - & - & - & - & - \\ 100 & 120 & - & 80 & - & - & - \\ 35 & 55 & - & - & - & - & - \end{bmatrix}$$

and

$$\begin{bmatrix} 40 & 20 & 50 & 20 & 130 & 50 & -10 \\ 45 & 45 & 115 & 23 & 50 & 20 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 300 \\ 300 \end{bmatrix}$$

as well as

$$\begin{bmatrix} 40 & 45 & 50 & 30 & 100 & 35 \\ 20 & 45 & 30 & 30 & 120 & 55 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 300 \\ 300 \end{bmatrix}$$

4.2 The aggregation matrix

Let us have the following table of statistical data:

Years	Industries					
	1	2	3	4	5	6
1960	15	201	5	19	73	49
61	16	200	9	18	74	50
62	18	201	20	18	73	51
63	17	199	25	16	81	53
1964	19	200	29	16	83	52

As it happens, the interest of a research worker may be only on the totals of industries 1, 2 and 3 together, 4 and 5 together, and 6 alone.

The relevant transformation of the data is defined by the aggregation matrix

$$\begin{array}{c} \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \end{array} \begin{array}{c} \begin{array}{ccc} (a) & (b) & (c) \end{array} \\ \begin{bmatrix} 1 & & \\ 1 & & \\ 1 & & \\ & 1 & \\ & 1 & \\ & & 1 \end{bmatrix} \end{array}$$

and we have

$$\begin{bmatrix} 15 & 201 & 5 & 19 & 73 & 49 \\ 16 & 200 & 9 & 18 & 74 & 50 \\ 18 & 201 & 20 & 18 & 73 & 51 \\ 17 & 199 & 25 & 16 & 81 & 53 \\ 19 & 200 & 29 & 16 & 83 & 52 \end{bmatrix} \begin{bmatrix} 1 & - & - \\ 1 & - & - \\ 1 & - & - \\ - & 1 & - \\ - & 1 & - \\ - & - & 1 \end{bmatrix} = \begin{bmatrix} 221 & 92 & 49 \\ 225 & 92 & 50 \\ 239 & 91 & 51 \\ 241 & 97 & 53 \\ 248 & 99 & 52 \end{bmatrix}$$

The aggregation matrix has a quite typical structure. For each p^{th} column vector of the original data, to be aggregated into the q^{th} column vector in the aggregated matrix of data, there is a unit-vector as the p^{th} row in the aggregation matrix, with the unity-element in the q^{th} place.

Obviously, one may also aggregate rows. In that case one has to pre-multiply the data-matrix with an aggregation matrix of corresponding structure. Such a row-aggregation matrix will have unit-vectors as columns, the aggregation of the p^{th} row of the original data-matrix in the q^{th} row of the transformed data matrix is then represented by the p^{th} unity column of the aggregation matrix having the unity-element in the q^{th} position.

4.3 Vector-permutation

The term "permutation" applies to the relation between two vectors or two matrices which contain the same numerical information, in a different arrangement, e.g.

$$\begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 2 \\ 6 \\ 4 \\ 8 \end{bmatrix}$$

With matrices, we may distinguish between a column-permutation, the interchanging of two columns, and row-permutation, the

interchanging of two rows, e.g.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \quad \text{versus} \quad \begin{bmatrix} 1 & 2 & 4 & 3 \\ 5 & 6 & 8 & 7 \\ 9 & 10 & 12 & 11 \end{bmatrix}$$

is a column-permutation, and

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \quad \text{versus} \quad \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

is a row-permutation.

Because of its use in the next chapter on determinants we must also define the term single permutation.

Two matrices of the same order and containing the same numerical information, differ by one (single) permutation, if the one may be transformed into the other, by interchanging two adjoining vectors (rows, e.g. columns of the matrix)..

This definition includes the permutation of two adjoining elements of a vector, which is a matrix, of which one of the order parameters is known to be unity.

One may obtain more complicated re-arrangements of the elements of a matrix, by means of a succession of single permutations.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 7 & 8 \\ 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 4 \\ 1 & 2 \\ 7 & 8 \\ 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 3 \\ 2 & 1 \\ 8 & 7 \\ 6 & 5 \end{bmatrix}$$

The number of single permutations needed to transform a matrix A into another matrix B, is the same as needed to transform B into A; the same permutations may be performed in the reverse order.

We may then speak of the number of permutations by which B differs from A, without having to specify the transformation of A into B or vice versa.

Two matrices (of the same order and numerical content), are different by q permutations, if one needs q interchangings of adjoining vectors (e.g. q single permutations), in order to transform the one into the other.

Permutations may be defined as matrix-multiplication, by means of operators.

To interchange the i^{th} and r^{th} row of a matrix, we may pre-multiply that matrix by a permutation-operator, which is a matrix of unit vectors, which differs from the unit matrix in that the i^{th} and the r^{th} row have been interchanged.

Example

$$\begin{bmatrix} - & - & - & 1 \\ - & 1 & - & - \\ - & - & 1 & - \\ 1 & - & - & - \end{bmatrix} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \quad = \quad \begin{bmatrix} 7 & 8 \\ 3 & 4 \\ 5 & 6 \\ 1 & 2 \end{bmatrix}$$

(The first and the fourth row have been interchanged).

The reader will note, that the operator can also be obtained by the interchangement of the first and the fourth (i^{th} and r^{th}) column of a unit-matrix of appropriate order.

Permutation operators never consist of anything else but unit vectors; one single unity-element and all zeros in the rest of the column (row).

Column permutation may be defined in an analogeous way, as post-multiplication by a permutation operator.

The use of operators will now enable us to state and proof some theorems on vector-permutation.

Definition

A permutation operator is a square matrix, of which each row as well as each column consists of n - 1 zeros and a single unity element, n being the order of the operator.

Theorem

All rearrangements of the rows of a matrix may be obtained by means of pre-multiplication of the matrix by a permutation-operator.

Proof

Consider the transformation of the matrix A into the matrix B, by re-arrangement of the rows.

We postulate the existence of a transformation-operator T, (of some description), satisfying

$$B = T \cdot A \qquad (4.3.1)$$

By the nature of the row-rearrangement operator, A and B must be of the same order. Therefore T must be square, its order is the number of rows of both A and B. The assumption that such an operator-matrix T actually exists, is at this point arbitrary. But if we show that an operator T, of certain characteristics, actually satisfies (4.3.1) for all A and B, its existence is shown in the process. We therefore investigate the properties which T must satisfy, in order to comply with (4.3.1).

For each $\underline{a}'_i$, for each i^{th} row of A, there must be a new row index r, to become $\underline{b}'_r$, the r^{th} row of B.

This operation is defined by pre-multiplication of A, by an operator T, which has a unit vector as the r^{th} row, with the unity-element in the i^{th} position.

If there are m rows in A, there will also be m rows in B, and we need m unit-vectors as rows of the operator, to define the m rows of B.

Also, the i^{th} row of A must be transformed into one and only one single row of B. To this purpose we need one and only one single unity-element in the i^{th} column of the operator. This makes the operator into a permutation-operator as defined.
q.e.d.

Corollary

Any rearrangement of the columns of a matrix may be obtained by post-multiplication of the matrix, with a permutation-operator.

In that case the transformation of the j^{th} column of a matrix A, into the k^{th} column of a matrix B, is defined by the k^{th} column of the operator, which has the unity-element in the j^{th} row.

Exercise

Find the permutation-operator P, which transforms

$$\begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{bmatrix} \text{ into } P \cdot \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{bmatrix} = \begin{bmatrix} 31 & 32 & 33 & 34 \\ 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \end{bmatrix}$$

CHAPTER V

DETERMINANTS AND RANK

5.1 Determinants and Minors

The determinant is a scalar function of a square matrix. It is conventional to indicate the determinant by the symbol for the matrix, placed between two vertical lines e.g. if A is a square matrix, we write $|A|$ for the determinant of A. The determinant is best defined recursively.*

For a matrix of order 1 by 1, the determinant is the one single element, e.g.

$$\left|[7]\right| = 7$$

For matrices of order 2 and upwards, we have the recursive relation

$$|A| = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \, |A_{1j}| \qquad (5.1.1)$$

where n is the order** of A, and A_{1j} is a matrix of order n-1, namely A less its first row and its j^{th} column. The determinant of such a matrix of smaller order is called a minor.

Example

$$A = \begin{vmatrix} 1 & 2 & 0 \\ 3 & -4 & 5 \\ 0 & 6 & 0 \end{vmatrix}$$

$$A = 1\begin{vmatrix} -4 & 5 \\ 6 & 0 \end{vmatrix} - 2\begin{vmatrix} 3 & 5 \\ 0 & 0 \end{vmatrix} + 0\begin{vmatrix} 3 & -4 \\ 0 & 6 \end{vmatrix}$$

---

* As a result many proofs concerning determinants are obtained by means of induction or recursive application, e.g. the theorem is true, as a restatement of a definition for matrices of order 1 or 2, but can be shown to be true for matrices of order k+1, if it is assumed to be true and valid for all matrices of order k. The proof is then carried k = 1, 2, 3, ... etc.

** All matrices in this chapter will be square and of order n, unless otherwise stated.

We now develop the minors corresponding to the non-zero elements of the first row of A.

$$|A_{11}| = -4|0| \quad -5|6| = -30$$

$$|A_{12}| = 3|0| \quad -5|0| = 0$$

Hence we find

$$|A| = 1 \times -30 + 0 = -30$$

The concept of a minor may be generalized.

If A is a square matrix of order n, the minor $|A_{ij}|$ is the determinant of a square matrix of order n-1, which is A, less the i^{th} row and the j^{th} column.

The minor $|A_{ij}|$ may also be indicated as the minor of a_{ij} where a_{ij} is an element of A, on the intersection of the i^{th} row and the j^{th} column.

The corresponding expression $|A_{ij}|(-1)^{i+j}$ which often occurs in the development of determinants is known as the co-factor of the element $a_{i,j}$.

Hence for $A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{vmatrix}$

we have $A_{3,2} = \begin{vmatrix} 1 & 3 & 4 \\ 5 & 7 & 8 \\ 13 & 15 & 16 \end{vmatrix}$, $c_{3,2} = -\begin{vmatrix} 1 & 3 & 4 \\ 5 & 7 & 8 \\ 13 & 15 & 16 \end{vmatrix}$

The square matrix, from which the determinant is taken will be indicated by its two indices*, i.e.

A_{ij} is A less the i^{th} row and the j^{th} column.

---

* There obviously is a potential possibility of confusion between minors and blocks (see Sections 2.16 and 2.17). It should, however be clear from the context, whether A_{ij} is the intersection of the i^{th} block-row and the j^{th} block-column (a block) of A, with deletion of the i^{th} row and the j^{th} column (a minor matrix). In practice, block-notation and determinant-formulae do not occur together, at least not at this elementary level.

The matrix A_{ij} will be called a minor-matrix. If no confusion between the minor matrix and the minor, which is its determinant, is possible, we may also use the term "minor" for both the minor-matrix and its determinant.

It will be useful, to have a symbol for a minor of a minor.

$\left|\left|A_{ij}\right|_{rk}\right|$ will be the minor of A_{ij}, arising from deletion of the r^{th} row and the k^{th} column from the original A.

e.g. $$A = \begin{vmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{vmatrix}$$

$$A_{2,3} = \begin{vmatrix} 11 & 12 & 14 \\ 31 & 32 & 34 \\ 41 & 42 & 44 \end{vmatrix}$$

$$\left|\left|A_{2,3}\right|_{1,4}\right| = \begin{vmatrix} 31 & 32 \\ 41 & 42 \end{vmatrix}$$

If A is of order n, then $\left|\left|A_{i,j}\right|_{r,k}\right|$ is the determinant of a matrix of order n-2, which is A, less its i^{th} and its r^{th} row, and less the j^{th} and k^{th} column. Note that both sets of indices refer to the original matrix A. It follows that

$$\left|\left|A_{i,j}\right|_{r,k}\right| = \left|\left|A_{r,k}\right|_{i,j}\right| \quad \ldots\ldots \quad (5.1.2)$$

Theorem

For any square matrix A, of order n,

$$\sum_{j=1}^{n} a_{ij}(-1)^{i+j}|A_{ij}| = |A| \quad \ldots\ldots \quad (5.1.3)$$

holds for i = 1, 2, ... n.

i.e. we may develop the determinant of A, by any row, not just the first one.

Proof

For i = 1 the statement coincides with the definition of a determinant.

For $n \geq i > 1$ the proof is as follows:

Suppose the theorem is true for matrices up to the order $n = q$, e.g. $n = 1$, $n = 2$ ..... $n = q$.

Then for $n = q + 1$, we may evaluate $|A|$, by its definitional expression (5.1.1), but evaluate each of the minors A_{1j} by means of (5.1.3).

$$\text{e.g. } |A_{1j}| = \sum_{k=1}^{j-1} a_{ik}(-1)^{i+k-1} \left| |A_{1j}|_{rk} \right| + \sum_{k=j+1}^{n} a_{ik}(-1)^{i+k-2} \left| |A_{1j}|_{rk} \right| \qquad (5.1.4)$$

The difference in the exponent of -1 arises because a_{ik} belongs to the $(i-1)^{th}$ row of A_{1j}, the first row of A not being part of A_{1j}.

For $k > j$ a similar effect arises for the columns. The element $a_{i,k}$ belongs to the k^{th} column of A, but to the $(k-1)^{th}$ column of A_{1j}. Hence the exponent of -1 in the second group of terms is $i + k - 2$.

For $|A|$ we then obtain, by (5.1.1) and (5.1.4)

$$|A| = \sum_{j=1}^{n} a_{1j} \left(\sum_{k=1}^{j-1} a_{ik}(-1)^{i+k-1} \left| |A_{1j}|_{ik} \right| + \sum_{k=j+1}^{n} a_{ik}(-1)^{i+k-2} \left| |A_{1j}|_{ik} \right| \right) \quad \ldots \qquad (5.1.5)$$

$(i = 2, 3, \ldots\ldots n)$

Now note, that in (5.1.5) each term contains a common factor $a_{1j}\, a_{ik}$. Therefore we change nothing if we place a_{ik} outside the brackets and a_{1j} inside.

The summation is then over $k = 1$ to n
and $j = 1, 2$ ..... $k-1$
and $j = k+1$ ..... n.

This is equivalent to the original, as this expresses in both cases the requirement $j \neq k$.

(For $j = k$, we cannot exclude the k^{th} column of the original matrix from A_{ij} since this column is not part of A_{ij}).

We obtain

$$|A| = \sum_{k=1}^{n} a_{ik} \left(\sum_{j=1}^{k-1} a_{1j}(-1)^{i+j-1} \left| |A_{1j}|_{ik} \right| + \sum_{j=k+1}^{n} a_{1j}(-1)^{i+j-2} \left| |A_{1j}|_{ik} \right| \right) \qquad (5.1.6)$$

$(i = 2, 3 \ldots\ldots\ldots n)$

However, by (5.1.2) we have

$$\left| |A_{1j}|_{ik} \right| = \left| |A_{ik}|_{1j} \right|$$

Hence we obtain

$$|A| = \sum_{k=1}^{n} a_{ik} \left(\sum_{j=1}^{k-1} a_{1j}(-1)^{i+j-1} \left| |A_{ik}|_{1j} \right| + \sum_{j=k+1}^{n} a_{1j}(-1)^{i+j-2} \left| |A_{ik}|_{1j} \right| \right) \qquad (5.1.7)$$

The expression within brackets is nothing else but $|A_{ik}|$, developed by the minors of its first row by the original definition of a determinant.

Then, on the assumption that the theorem is true for $n = q$, it is shown to be true for $n = q + 1$ as well.

However, for $n = q = 2$ we have

$$\begin{aligned} |A| &= a_{11}|A_{11}| - a_{12}|A_{12}| \\ &= a_{11}\,a_{22} - a_{12}\,a_{21} \\ &= -a_{21}\,a_{12} + a_{22}\,a_{11} \\ &= -a_{21}|A_{21}| + a_{22}|A_{22}| \end{aligned}$$

which shows the theorem to be true for $n = q = 2$.

Hence it is true for $n = 3$, $n = 4$, etc.
q.e.d.

Example

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

Develop $|A|$ by the minors of the first row.

$$|A| = 1\begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2\begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3\begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

$$= 1\,(5 \times 9 - 8 \times 6) - 2\,(4 \times 9 - 7 \times 6) + 3\,(4 \times 8 - 7 \times 5)$$

$$= -4\,(2 \times 9 - 3 \times 8) + 5\,(1 \times 9 - 3 \times 7) - 6(1 \times 8 - 2 \times 7)$$

$$= -4\begin{vmatrix} 2 & 3 \\ 8 & 9 \end{vmatrix} + 5\begin{vmatrix} 1 & 3 \\ 7 & 9 \end{vmatrix} - 6\begin{vmatrix} 1 & 2 \\ 7 & 8 \end{vmatrix}$$

which is $|A|$, developed by the minors of the second row

Theorem

$$\sum_{i=1}^{n} a_{i1}(-1)^{i+1}|A_{i1}| = |A| \qquad \ldots\ldots \quad (5.1.8)$$

e.g. we can develop $|A|$ by the minors of the first column, in the same way as we may do for the first row (any row).

Proof

Develop $|A|$ by the minors of the first row

$$|A| = \sum_{j=1}^{n} a_{1j}\,(-1)^{1+j}|A_{1j}|$$

Suppose the theorem to be true for n = q.

Then for n = q + 1 we may develop all the minors, except A_{11} by the first column

$$|A_{1j}| = \sum_{i=2}^{n} a_{i1}(-1)^{i}\left||A_{1j}|_{i1}\right|$$

We may now re-write $|A|$ as

$$|A| = a_{11}|A_{11}| + \sum_{j=2}^{n} a_{1j}(-1)^{1+j}|A_{1j}|$$

$$= a_{11}|A_{11}| + \sum_{j=2}^{n} a_{1j}(-1)^{1+j} \sum_{i=2}^{n} a_{i1}(-1)^{i} \left| |A_{1j}|_{i1} \right|$$

However $\left| |A_{1j}|_{i1} \right| = \left| |A_{i1}|_{1j} \right|$

and hence

$$|A| = a_{11}|A_{11}| + \sum_{j=2}^{n} a_{1j}(-1)^{1+j} \sum_{i=2}^{n} a_{i1}(-1)^{i} \left| |A_{i1}|_{1j} \right|$$

$$= a_{11}|A_{11}| + \sum_{i=2}^{n} a_{i1}(-1)^{i+1} \sum_{j=2}^{n} a_{1j}(-1)^{j} \left| |A_{i1}|_{1j} \right|$$

Now note that, by the original definition of a determinant, we find:

$$\sum_{j=2}^{n} a_{1j}(-1)^{j} \left| |A_{i1}|_{1j} \right| = |A_{i1}|$$

developed by its first row.

Hence

$$|A| = a_{11}|A_{11}| + \sum_{i=2}^{n} a_{i1} (-1)^{i+1}|A_{i1}|$$

$$= \sum_{j=1}^{n} a_{ij}(-1)^{i+1}|A_{i1}|$$

Hence, on the assumption that the theorem is true for n = q, we have shown that it is true for n = q + 1.

However, for n = 2, we have

$$|A| = a_{11}\, a_{22} - a_{12}\, a_{21}$$

$$= a_{11}|A_{11}| - a_{12}|A_{12}|$$

which shows the theorem to be true for n = 2.

Hence it is true for

$$n = 2 + 1 = 3, \quad n = 3 + 1 = 4 \ldots\ldots\ldots\ldots \text{ etc.}$$

q.e.d.

By analogy to our proof for the development of a determinant by the minors of an arbitrary row, we also have:

$$\sum_{i=1}^{n} a_{ij}(-1)^{i+j}|A_{ij}| = |A| \qquad \ldots\ldots \qquad (5.1.9)$$

Conclusion

The determinant may be developed by means of any row or any column.

Exercise

Develop $\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$, by the first row, and also by the third column. The answers should obviously be the same (zero).

5.2 Permutations of determinants

Under this heading we discuss the determinant of any square matrix B, which may be obtained from another matrix A, by means of vector-permutation (see section (4.3)) as well as one case of elentswise reordering, the transpose.

Theorem

The interchanging of two adjoining rows causes a change in the sign of the determinant, but leaves the absolute value unchanged.

Proof

Let B be related to A, by permutation of the i^{th} and the $(i+1)^{th}$ rows. e.g. the i^{th} row of B is the $i+1^{th}$ row of A and the $(i+1)^{th}$ row of B is the i^{th} row of A, and for

$$r = 1, 2, \ldots\ldots i - 1, i + 2, \ldots\ldots\ldots n$$

(all other rows)

the r^{th} row of B equals the r^{th} row of A.

Now develop $|B|$ by its i^{th} row

$$|B| = \sum_{j=1}^{n} b_{ij}(-1)^{i+j}\left|B_{ij}\right|$$

However, by assumption, we have

$$b_{ij} = a_{j+1,j} \quad \text{and} \quad |B_{ij}| = |A_{i+1,\,j}|$$

since B less the i^{th} row and A less the $i + 1^{th}$ row contains the same numbers.

Therefore

$$|B| = \sum_{j=1}^{n} a_{i+1,\,j}(-1)^{i+j} |A_{i+1,\,j}| .$$

Since $(-1)^{i+j} = -(-1)^{i+1+j}$

we have obtained:

$$|B| = -\sum_{j=1}^{n} a_{i+1,\,j}(-1)^{i+1+j}|A_{i+1,\,j}|$$

which is $-|A|$

q.e.d.

Corollary

If two adjoining rows of a square matrix are equal to each other, the determinant is zero.

Theorem

Permutation of two rows of a matrix, whether adjoining or not, causes the sign of the determinant to change, its absolute value remaining unchanged.

Proof

If the i^{th} and the $(i + q)^{th}$ rows are to be interchanged, this may be effected by 2. q - 1 simple permutations. There are q - 1 rows between the i^{th} and the $(i + q)^{th}$ row;

we need to interchange the i^{th} row with the $(i + 1)^{th}$, the $(i + 2)^{th}$, ... and the $(i+ q - 1)^{th}$ in order to obtain a matrix in which the (original) i^{th} and the $(i + q)^{th}$ are adjoinging rows.

These are q-1 simple permutations. It then takes one simple permutation to interchange the $(i + q)^{th}$ row with the (now adjoining) old i^{th} row, and again q - 1 simple permutations to interchange the old $(i + q)^{th}$ row with the (old) $(i + q - 1)^{th}$ row, the (old) $(i + q - 1)^{th}$ row etc., in order to obtain a matrix with the (old) $(i + q)^{th}$ row in the i^{th} position.

The total number of simple permutations is always odd.

The determinant is then multiplied by

$$(-1)^{[2(q-1) + 1]} = -1$$

q.e.d.

Corollary

The determinant of a square matrix with two equal rows is zero.

The corresponding theorem for columns is now stated without further proof:

Permutation of two columns causes the sign of the determinant to change, the absolute value of the determinant remaining unchanged.

Corollary

A square matrix containing two equal columns has a zero determinant.

Theorem

$$|A'| = |A|$$

Proof

Suppose this theorem to be true for n = q, the theorem may then be applied for the minors of a matrix of order n = q + 1.

Develop $|A'|$ by the first row of A' which is the first column of A

$$|A'| = \sum_{i=1}^{n} a_{i1}(-1)^{1+i}|A'_{1i}|$$

As just observed, we assume the theorem to be true for the minors, hence we may write $|A_{i1}|$ for $|A'_{1i}|$

$$|A'| = \sum_{i=1}^{n} a_{i1}(-1)^{1+i}|A_{i1}|$$

which is A, developed by the first column of A.

Which shows that the theorem is true for n = q + 1, if it is true for n = q = 1.

However, for n = q = 1, we have

$$|A'| = |A| = a_{11}$$

Hence the theorem is true for n = 1; hence it is also true for n = 1 + 1 = 2, for n = 2 + 1 = 3, ... etc.
q.e.d.

5.3 Proportionality of vectors

Theorem

If two matrices A and B differ only in one row, or one column n, the difference being that one row of B is the corresponding row of A multiplied by a scalar α, e.g. $\underline{b}_i' = \alpha \underline{a}_i'$ the determinant of B is α times the determinant of A.

$$|B| = \alpha |A|.$$

Proof

Develop both determinants by the i^{th} row.

$$|B| = \sum_{j=1}^{n} b_{ij} (-1)^{i+j} |B_{ij}|$$

$$= \sum_{j=1}^{n} \alpha a_{ij} (-1)^{i+j} |B_{ij}|$$

$$= \alpha \sum_{j=1}^{n} a_{ij} (-1)^{i+j} |A_{ij}| = \alpha |A|$$

q.e.d.

Theorem

If a square matrix C (of order n) has two proportional rows

$$c_{ij} = \alpha c_{ri} \quad .... \quad (5.3.1)$$

$(i \neq r, j = 1, 2 ..... n)$

then $|C| = 0$

Proof

Suppose two matrices A and B were the same except for two rows, for which

$$b_{ij} = \alpha a_{ij}$$
$$b_{rj} = \alpha^{-1} a_{ij} \qquad \text{was true for } j = 1, 2 \ldots\ldots n$$

By our previous theorem, we would obtain

$$|B| = \alpha.\alpha^{-1}|A| = |A| \qquad \ldots\ldots \qquad (5.3.2)$$

However, we may obtain B from A, by row-permutation; hence

$$|B| = -|A| \quad \text{is true.} \qquad \ldots\ldots \qquad (5.3.3)$$

From (5.3.2) and (5.3.3) we infer $A = B = O$

However, the described relation is the one between C and C itself. Hence

$$|C| = |C| = O \qquad \ldots\ldots \qquad (5.3.4)$$

is true.
q.e.d

The two proceeding theorems have an obvious

Corollary by analogy:

If a column of a matrix is multiplied by a scalar number, the determinant is multiplied by that same number.

If two columns of a matrix are proportional the determinant is zero.

Theorem

If two square matrices A and B, of the same order, are different in one row only, and the difference consists in the addition of a proportionality of one row to another row, their determinants are equal.

e.g. <u>if</u> $b_{ij} = a_{ij}$ for $j = 1, 2, \ldots\ldots n$

and $i = 1, 2 \ldots\ldots r - 1, r + 1, \ldots\ldots n$

and $b_{ri} = a_{rj} + \gamma a_{hj}$

for $j = 1, 2 \ldots\ldots n$, and a specific r and a specific h, then

$$|B| = |A|$$

Proof

Assume the theory to be true for a particular $q > 2$

Then for $n = q + 1$

we may develop $|B|$ by the minors of a row with row-index

$$i \neq r \quad \text{and} \quad i \neq h.$$

We may then apply the theorem to the minors which are of order q and write $|A|_{ij}$ for $|B|_{ij}$. Also for such a row we have $b_{ij} = a_{ij}$

Hence

$$|B| = \sum_{j=1}^{n} a_{ij} (-1)^{i+j} |A|_{ij} = |A|$$

The theorem is therefore true for $n = q + 1$ if it is true for $n = 1$. For $n = 2$ the theorem holds, since

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} + \gamma a_{11} & a_{22} + \gamma a_{12} \end{vmatrix} = a_{11}(a_{22} + \gamma a_{12}) - (a_{21} + \gamma a_{11})(a_{12})$$

$$= a_{11}\, a_{22} + \gamma a_{11}\, a_{12} - a_{21}\, a_{12} - \gamma a_{11}\, a_{12}$$

$$= a_{11}\, a_{22} - a_{21}\, a_{12} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

Hence the theorem is true for $n = 2 + 1 = 3$, for $n = 3 + 1 = 4$, ... etc.
q.e.d.

Corollary by analogy

Adding a proportionality of a column to another column does not influence the determinant.

With the help of this theorem, we can prove the following Theorem*

A singular matrix has a zero determinant

Proof

Let A be a square matrix of order n, and let A be singular. Then, by the definition of singularity

$\underline{y}'A = 0$ is true for some $\underline{y}' \neq 0$.

Partition A into a single and a remaining block-row; and $\underline{y}'$ into single element and a remaining vector.

$$A = \begin{vmatrix} \underline{a}_1' \\ \underline{A}_2' \end{vmatrix} \qquad \underline{y}' = |y_1, \underline{y}_2'|$$

$$\underline{a}_1' y_1 + A_2' \underline{y}_2 = 0 \qquad \ldots\ldots \qquad (5.3.5)$$

Since row permutation does not change the absolute value of the determinant, we may without loss of generality interchange the rows of A and corresponding elements of $\underline{y}'$.

We will then require

$$y_1 \neq 0$$

we further assume $\underline{a}_1' \neq 0$.

(In the case $\underline{a}_1' = 0$ the theorem is trivial but true.)

Hence by (5.3.5) we obtain

$$\underline{y}_2' A_2' = -y_1 \underline{a}_1' \neq 0 \qquad \ldots\ldots \qquad (5.3.6)$$

Accordingly, we may subtract each i^{th} row ($i = 2, 3 \ldots n$) weighted with the finite proportionality factor $y_i y_1^{-1}$, from the first row, without changing the determinant.

The result is a matrix with a zero row, showing

$$|A| = 0$$

q.e.d

* The reverse of this theorem (a matrix with a zero determinant is singular) is also true, but its proof has to be deferred (to Section 5.5).

The alternative singularity definition $\underline{Ax} = 0$, $\underline{x} \neq 0$ (not so far used in this book) also implies $|A| = 0$, since $|A'| = 0$.

Example

$$A = \begin{vmatrix} -1 & 1 & 0 & 2 \\ 2 & 3 & 0 & 1 \\ 4 & -6 & 2 & 5 \\ 5 & -2 & 2 & 8 \end{vmatrix} = 0$$

This can, of course, be shown in two ways, the hard way, by calculation of the determinant, or more easily, by finding the fourth (bottom) row to be the sum of the three others, or

$$\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 2 \\ 2 & 3 & 0 & 1 \\ 4 & -6 & 2 & 5 \\ 5 & -2 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

Exercise

Find a suitable combination of the rows, to show the singularity of $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

5.4 Decomposability of a determinant

How many different permutation operators of a given order can be written? At this point we recall the definition of a permutation operator. This is a square matrix, which may be obtained by re-ordering the rows of a unit matrix. Alternatively, a permutation operator may be defined as a permutation of the columns of the unit matrix; the result is the same.

Then if the order is n, there are n rows in which to place the unity element in the first column, n - 1 rows in which one may place the unity element in the second column, e.g. all excluding the one in which the first element is unity.

The total number of choices is $n(n - 1)$, ... $2 \cdot 1$
i.e. $n!$

By precisely the same logic there are also $n!$ recursive products of n numbers in a determinant.

Developing the determinant by the first column, we have n choices for the first factor in such a recursive product.

Then, developing each minor, we have n - 1 choices for the relevant element in the first column in the minor, etc.

Example

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

3 choices in first column. For a_{11} we have the term $a_{11}|A|_{11}$ and the group of permutation operators

$$\begin{vmatrix} 1 & - & - \\ - & 1 & - \\ - & - & 1 \end{vmatrix} \quad \text{and} \quad \begin{vmatrix} 1 & - & - \\ - & - & 1 \\ - & 1 & - \end{vmatrix}$$

For a_{21} we have the term $-a_{21}|A|_{21}$ and the group of permutation operators

$$\begin{vmatrix} - & 1 & - \\ 1 & - & - \\ - & - & 1 \end{vmatrix} \quad \text{and} \quad \begin{vmatrix} - & - & 1 \\ 1 & - & - \\ - & 1 & - \end{vmatrix}$$

etc.

The full list of permutation operators or recursive products may be enumerated as

$$\begin{vmatrix} 1 & - & - \\ - & 1 & - \\ - & - & 1 \end{vmatrix} \quad \text{for} \quad a_{11}\, a_{22}\, a_{33}$$

$$\begin{vmatrix} 1 & - & - \\ - & - & 1 \\ - & 1 & - \end{vmatrix} \quad \text{for} \quad -a_{11}\, a_{32}\, a_{23}$$

$$\begin{vmatrix} - & 1 & - \\ 1 & - & - \\ - & - & 1 \end{vmatrix} \quad \text{for} \quad -a_{21}\, a_{12}\, a_{33}$$

$$\begin{vmatrix} - & - & 1 \\ 1 & - & - \\ - & 1 & - \end{vmatrix} \quad \text{for} \quad a_{21}\, a_{32}\, a_{13}$$

$$\begin{vmatrix} - & 1 & - \\ - & - & 1 \\ 1 & - & - \end{vmatrix} \quad \text{for} \quad a_{31}\, a_{12}\, a_{23}$$

$$\begin{vmatrix} - & - & 1 \\ - & 1 & - \\ 1 & - & - \end{vmatrix} \quad \text{for } -a_{31}\, a_{22}\, a_{13}$$

The reader will note, that the signs of these recursive products are the signs of the determinants of the corresponding permutation operators.

The sign of such a recursive product is dependent only on the indices of the corresponding elements and these are the same as the indices of the non-zero elements of the permutation matrix.

We now come to the following result:

The determinant of a square matrix equals the sum of the determinants of all possible elementwise products of the matrix with a permutation operator, e.g.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} a_{11} & & \\ - & a_{22} & - \\ - & - & a_{33} \end{vmatrix} + \begin{vmatrix} a_{11} & - & - \\ - & - & a_{23} \\ - & a_{32} & - \end{vmatrix}$$

$$+ \begin{vmatrix} - & a_{12} & - \\ a_{21} & - & - \\ - & - & a_{33} \end{vmatrix} + \begin{vmatrix} - & - & a_{13} \\ a_{21} & - & - \\ - & a_{32} & - \end{vmatrix}$$

$$+ \begin{vmatrix} - & a_{12} & - \\ - & - & a_{23} \\ a_{31} & - & - \end{vmatrix} + \begin{vmatrix} - & - & a_{13} \\ - & a_{22} & - \\ a_{31} & - & - \end{vmatrix}$$

5.5 Determinant and inversion by row-operations*

Recall the process for calculating an inverse matrix, which we discussed in Chapter III.

Each elimination step consists of:

(a) If necessary a row permutation, to ensure a non-zero diagonal pivotal element. This part of the operation changes the sign of the determinant of both the left hand side and the right-hand side matrix.

---

*For further reference on this topic see also Maurer [28]

(b) Multiplication of the pivotal row by the reciprocal of the pivotal element. This part of the operation multiplies the determinants of both the left-hand side matrix and the right hand side matrix by the reciprocal of the pivotal element.

(c) Addition of a multiple of the pivotal row to the other rows. This part of the operation leaves both determinants unchanged.

We start the inversion of a matrix A with a system $A\,\underline{x} = \underline{y}$ tabulated as

$$\begin{array}{cc} \underline{x} & \underline{y} \\ A & I \end{array}$$

and end the operation (if the inverse exists), with

$$\begin{array}{cc} \underline{x} & \underline{y} \\ I & A^{-1} \end{array}$$

Since the determinant of a unit matrix is always one, it follows that, except for the sign we obtain the determinant of A as the recursive product of the pivots.

Example

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & -4 & 5 \\ 0 & 6 & 0 \end{bmatrix} \quad \text{(from Section 5.1)} \quad |A| = -30$$

L				R			$\sum$
x_1	x_2	x_3	=	y_1	y_2	y_3	
(1)	2	0		1	-	-	4
3	-4	5		-	1	1	5
0	6	0		-	-	1	7

$|L| = -30$ $\qquad$ $|R| = 1$

x_1	x_2	x_3		y_1	y_2	y_3	$\sum$
1	2	0		1	-	-	4
-	(-10)	5		3	1	-	-7
-	6	0		0	-	1	7

$$|L| = \begin{vmatrix} -10 & 5 \\ 6 & 0 \end{vmatrix} \qquad |R| = 1$$

$$= -30,$$

unchanged because the first pivot was one.

x_1	x_2	x_3
1	-	1
-	1	$-\frac{1}{2}$
-	-	(3)

y_1	y_2	y_3
2/5	1/5	-
3/10	-1/10	-
-1 4/5	3/5	1

Σ
2 3/5
7/10
2 4/5

$L = 3$

$= -30 : -10$

$R = \begin{vmatrix} 2/5 & 1/5 \\ 3/10 & -1/10 \end{vmatrix}$ $*1$ $=$ $-1/10$

$=$ $1 : -10$

x_1	x_2	x_3
1	-	-
-	1	-
-	-	1

y_1	y_2	y_3
1	0	-1/3
0	0	1/6
-3/5	1/5	1/3

Σ
1 2/3
1 1/6
14/15

$|L| = 1$

$|R| = 1 \begin{vmatrix} 0 & 1/6 \\ 1/5 & 1/3 \end{vmatrix} + (-1/3) \begin{vmatrix} 0 & 0 \\ -3/5 & 1/5 \end{vmatrix}$

$= -1/30$

Exercise

Find the determinants of

$$A = \begin{vmatrix} - & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 9 \end{vmatrix}, \quad \text{and of } A = \begin{vmatrix} - & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix}$$

independently by two methods, i.e. by calculation according to its definition, and by inversion or attempt to invert. (Do not forget to swap sign when interchanging rows see section 3.10).

An obvious corrollary of the result obtained in this section is the following:

Any non-singular matrix may be inverted by row - operations, if necessary involving the interchanging of two rows of the matrix as indicated in section 3.10.

If an attempt to invert by row operations has to be abandoned, because no non-zero pivot may be found at all in the next column, neither on or below the diagonal, this proves the singularity of the matrix, the determinant being zero.

Furthermore, since all non-singular matrices can be inverted, it follows that all matrics with zero determinants are singular.

We have so far defined singularity as

$$\underline{y}' A = 0, \quad \underline{y}' \neq 0.$$

Since $|A| = |A'|$, it is now clear that $\underline{y}' A = 0$, $\underline{y}' \neq 0$; $A\underline{x} = 0$, $\underline{x} \neq 0$; and $|A| = 0$, are all equivalent definitions of singularity.

Exercise

Find the determinants of

$$A = \begin{vmatrix} -1 & 2 \\ 3 & -4 & 5 \\ -6 & 7 & -8 \end{vmatrix}, \quad A = \begin{vmatrix} -1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix}, \quad A = \begin{vmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \\ 7 & -8 & 9 \end{vmatrix}$$

independently by two methods

a) By calculating the determinants according to the definition of a determinant
b) By inverting or attempting to invert the matrices

Do not forget to swap the sign if row-permutation is needed (see section 3.10), and verify the correctness of any successfully calculated inverses by premultiplication i.e. application of (3.7.4) and (3.7.7).

5.6 The Calculation of Determinants

Two programmed procedures for the calculation of determinants are offered in this section. To calculate a determinant directly according to its definition is not a practical proposition. The two major complications are the following:

a) The rules for the search for and the signs of the various minors are somewhat complicated and would therefore lead to a somewhat complicated programme.
b) The matrices from which minors would have to be extracted do not as such exist, the elements of the larger matrix would first have to be copied into a minor-matrix of the appropriate order without the gaps for a missing row and column.

This would require additional space-reservation in the computer's memory. To calculate the determinant of a matrix

of let us say, 65 by 65, by that method one would need space-reservation for:

the 65 by 65 matrix itself
a 64 by 64 minor
a 63 by 63 minor of a principal minor etc...

The second of these complications is side-stepped by following Section 5.4 i.e. the determinant is evaluated as the sum of a series of determinants of elementwise products of the matrix with a permutation-operator. The first-mentioned complication is, in a sense, also present in this approach. It is not all that simple to generate "all possible" permutation operators and (the signs of) their determinants. Yet that is exactly what the following procedure does:

TEXT-LISTING OF THE PERMUTATION PROCEDURE:

```
'PROCEDURE' PERM(PO,N,K,SIGN,NEWSIGN,OPERATION);
'VALUE' K,SIGN; 'ARRAY' PO; 'INTEGER' N,K,SIGN,NEWSIGN;
'PROCEDURE' OPERATION;
'BEGIN' 'INTEGER' I,J;

    'COMMENT'
    PO IS THE PERMUTATION OPERATOR,
    AND IS ASSUMED TO BE OF ORDER N BY N

    IF PERM IS CALLED WITH THE PARAMETER K BEING ONE,
    THE WHOLE OPERATOR
    MATRIX IS FIRST EQUATED TO ZERO.
    EACH CALL OF PERM WILL PERMUTE ONE COLUMN OF PO, THAT IS,
    INVESTIGATE IF A UNIT -ENTRY IN ANY I TH ROW OF A COLUMN
    OF PO IS CONSISTENT WITH THE ENTRIES IN THAT I TH ROW,
    WHICH ARE ALREADY PLACED TO THE LEFTHAND-SIDE OF THE COLUMN
    WHICH IS CURRENTLY BEING PERMUTATED.
    AT THE END OF EACH 'TREE' OF PERMUTATIONS, THERE IS A CALL
    TO THE PROCEDURE OPERATION, I.E. THE OPERATION
    TO BE PERFORMED WITH THAT PARTICULAR PERMUTATION-OPERATOR.
    THE FULL SET OF PERMUTATION-OPERATORS IS GENERATED, BY
    THE DEVICE OF PERM CALLING ITSELF.
    IF SIGN WAS ASSIGNED THE INITIAL VALUE OF ONE, THEN,
    AT EACH EXIT-CALL TO OPERATION, SIGN WILL HAVE THE VALUE
    PLUS OR MINUS ONE,
    DEPENDING ON WHETHER THE NUMBER OF PERMUTATIONS WAS
    EVEN OR ODD;

   'IF' K > N 'THEN' 'BEGIN'
     OPERATION; 'GOTO' END OF PERM; 'END';


   'IF' K=1 'THEN'
   'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
   'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' PO[I,J]:=0;
```

```
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'FOR' J:=1 'STEP' 1 'UNTIL' K-1 'DO'
    'IF' PO[I,J] ≠ 0 'THEN' 'GOTO' ENDLOOP;
    PO[I,K]:=1;
    NEWSIGN:=SIGN;
    PERM(PO,N,K+1,NEWSIGN,NEWSIGN,OPERATION);

    CHANGE SIGN BECAUSE OF PERMUTATION:  SIGN := -SIGN;
    PO[I,K]:=0;
    ENDLOOP: 'END';

  END OF PERM: 'END';
```

As indicated by the 'COMMENT' this procedure caters for calling another procedure called operation, every time it has called back on itself with the index k - which indicates the column to be permuted - in excess of its order. It has then generated a particular permutation-operator, with its correct sign.

This sign should be flicked from plus to minus or vice versa at a particular permutation in the kth column without interference to its value to be transmitted to other minors when permuting columns further to the left again. For this reason this variable is not transmitted directly as a variable but "per value". The eventual result is stored in the variable NEWSIGN.

In the case at hand, this is the procedure TERM which is an internal procedure, subordinate to the procedure PERD which calculates the determinant by means of permutation-operators.

```
'PROCEDURE' PERD(A,N,D);
'ARRAY' A; 'INTEGER' N; 'REAL' D;
'BEGIN'
  'INTEGER' I,J,SIGN,NEWSIGN;

  'PROCEDURE' PERM(PO,N,K,SIGN,NEWSIGN,OPERATION);
  'VALUE' SIGN; 'ARRAY' PO; 'INTEGER' N,K,SIGN,NEWSIGN;
  'PROCEDURE' OPERATION; 'ALGOL';

  'PROCEDURE' TERM;
  'BEGIN' 'INTEGER' I,J; 'REAL' TE;
    'COMMENT'
    THE TERM TE, WHICH TERM ADDS TO A DETERMINANT IN THE PROCESS
    OF CALCULATION, IS THE RECURSIVE PRODUCT OF THOSE ELEMENTS
    OF THE MATRIX A, FOR WHICH THE PERMUTARION-OPERATOR HAS A
    UNITY-ELEMENT RATHER THAN A ZERO IN THE CORRESPONDING CELL,
    MULTIPLIED BY THE (NEW)SIGN INDICATED.;
```

```
      TE:=NEWSIGN;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
      'IF' PO[I,J]=1  'THEN' 'BEGIN'
        'IF' A[I,J]≠0 'THEN' TE:=TE*A[I,J]
        'ELSE' 'GOTO' END OF TERM; 'END';
      D:=D+TE;
      END OF TERM: 'END';

    'ARRAY' PO[1:N,1:N];

    START OF THE CONTROLLING MAIN BODY OF PERD:
    D:=0; SIGN:=1;
    PERM(PO,N,1,SIGN,NEWSIGN,TERM);

    END OF PERD: 'END';
```

This is a fairly complicated way of calculating a determinant.

Calculation of the determinant as a by-product of inversion on the lines of Section 5.5 is much simpler to programme. The following procedure is an adaptation of the inversion procedure which was offered in Section 3.13.

```
'PROCEDURE' INVD(A,M,N,D);
'VALUE' N; 'ARRAY' A; 'INTEGER' M,N; 'REAL' D;
'BEGIN' 'INTEGER' I,J,R,K; 'REAL' P,NUM;
  'COMMENT' DETERMINANT-CALCULATION WITH INVERSION;
  'IF' N=0 'THEN' 'BEGIN'
     N:=M;
     'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
       'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO' A[I,M+J]:=0;
      A[I,M+I]:=1; 'END'; 'END';

  INITIATE D:  D:=1;

  'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
     'COMMENT' FIRST FIND A NON-ZERO PIVOT;
     P:=0;
     'FOR' I:=R 'STEP' 1 'UNTIL' M 'DO' 'IF' ABS(A[I,R]) > ABS(P)
     'THEN' 'BEGIN' K:=I; P:=A[I,R] 'END';

     D:=D*P;
     'IF' ABS(P) < 0.000001 'THEN' 'BEGIN'
       SINGULAR:  'GOTO' END OF INVD; 'END';

     'IF' 'NOT' R=K 'THEN' 'BEGIN'
       INTERCHANGE ROWS AND CHANGE SIGN OF DETERMINANT:
       D:=-D;
       PERMUTE:
       'FOR' J:=R 'STEP' 1 'UNTIL' M+N 'DO' 'BEGIN'
         NUM:=A[R,J]; A[R,J]:=A[K,J]; A[K,J]:=NUM; 'END'; 'END';

     UPDATE:
     'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO' A[R,J]:=A[R,J]/P;
     'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M 'DO'
     'IF' 'NOT' A[I,R]=0
     'THEN'  'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO'
     A[I,J]:=A[I,J] - A[R,J]*A[I,R];  'END';

  END OF INVD:     'END';
```

The textual similarity between the INVD and INVE (See Section 3.13) is enhanced by the fact that the one was actually obtained by reproducing and partially amending the other.

The substantial differences, besides "COMMENT" and the labels are the following:

Firstly, there has to be a variable to indicate the determinant. Since the calculation of the determinant is one of the tasks of the procedure, this variable must be accessible outside the procedure. Therefore it has to be a procedure-parameter. Hence the additional parameter D.

Secondly there is the actual calculation of the determinant. Only three instructions relate to this, i.e. the setting of an initial value, D:=1, the evaluation of the recursive product of the pivots, i.e. D:=D*P, and D:=-D, which relates to the change in sign, in case two rows are interchanged or, as the relevant label has it, permuted.

The one other difference is that no alarm-message is printed when a matrix is found singular. This is because if someone uses INVD rather than INVE, one assumes he will verify the non-singularity himself, by investigating the value of the determinant. If he has no intention of doing that, he should use INVE rather than INVD.

If one desires the calculation of the determinant as such rather than as a by-product of inversion one could gain a slight gain in computational efficiency by taking the N=0 loop for putting a unit matrix as righthand side out of the procedure. In that case the calculation would be performed without any right-hand side reduced form calculation at all, if N=0 was supplied. However, a common feature of INVE and INVD is that the original matrix is replaced by some intermediate result. One is not likely to want calculation of a determinant and then do nothing more with the matrix.

The other procedure for calculating the determinant by permutation operators leaves the matrix itself unchanged.

5.7 Rank and the determinants of some structured matrices

The rank of a matrix is the order of the biggest square and non-singular block that can be found for any ordering of the rows and columns of the matrix.

Thus,

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 0 & 3 & 3 & 3 \\ 0 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

is of order 5 by 4, but its rank is only 2.

The blocks $\begin{bmatrix} 1 & 1 \\ 0 & 3 \end{bmatrix}$, $\begin{bmatrix} 2 & 2 \\ 0 & 3 \end{bmatrix}$, $\begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix}$, $\begin{bmatrix} 2 & 2 \\ 0 & 4 \end{bmatrix}$, $\begin{bmatrix} 0 & 3 \\ 5 & 5 \end{bmatrix}$ and $\begin{bmatrix} 0 & 4 \\ 5 & 5 \end{bmatrix}$

are square and non-singular. But all square blocks of order 3 or 4, e.g.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 0 & 3 & 3 \end{bmatrix}$$

contain proportional rows and identical columns.

Obviously the rank of a matrix is at most equal to its smallest order parameter. A matrix (or a block) of which the rank is equal to the smallest order parameter is said to be of <u>full rank</u>.

Thus, $A = \begin{bmatrix} 1 & -2 & 0 \\ -2 & 1 & 0 \end{bmatrix}$

is of full rank, i.e. its rank is 2, and it cannot be more than 2, because there are only two rows.

Since we cannot have a negative number of non-zero elements, the lowest possible rank is zero, in which case the matrix (or the block) contains nothing but zeros.

The concept of rank is useful, in particular with proofs concerning the non-singularity or singularity of partitioned square matrices.

Theorem

Let

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

be a square matrix with the diagonal blocks A_{11} and A_{22} being square, and $|A_{11}| \neq 0$.

Then

$$|A| = |A_{11}| \; | |A_{22} - A_{21} A_{11}^{-1} A_{12}||$$

Proof

Let us, for the moment assume $|A| \neq 0$

Consider the inversion of A by blocks, following the rules of section 3.11. The associated partitioned system is

$$A_{11} \underline{x}_1 + A_{21} \underline{x}_2 = \underline{y}_1$$

$$A_{21} \underline{x}_1 + A_{22} \underline{x}_2 = \underline{y}_2$$

We make the first block-inversion-step, as follows

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}_1$	$\underline{y}_2$
A_{11}	A_{12}		I	
A_{21}	A_{22}			I

$\underline{x}_1$	$\underline{x}_2$	=	$\underline{y}_1$	$\underline{y}_2$
I	$A_{11}^{-1} A_{21}$		A_{11}^{-1}	
	$A_{22} - A_{21} A_{11}^{-1} A_{12}$		$-A_{21} A_{11}^{-1}$	I

Now recall section 5.5.

Assuming that the inversion is along the diagonal (i.e. the matrix is ordered if needed, beforehand) we obtain the determinant of A, as the recursive product of its pivots.

These are the determinants of A_{11} (i.e. the recursive product of the pivots used in inverting A_{11}), multiplied by the recursive product of the pivots met in the second block-inversion step, i.e. $|A_{22} - A_{21} A_{11}^{-1} A_{12}|$. q.e.d. for $|A| \neq 0$.
Furthermore, if $|A_{22} - A_{21} A_{11}^{-1} A_{12}|$ is singular, the inversion fails, showing $|A| = 0$. q.e.d. for $|A| = 0$.
Which completes the proof.
q.e.d.

Similarly, the inversion fails if no non-singular block-pivot A_{11} can be found in the first place.

Therefore we have the following

Corollary

A square matrix of which any block-column is not of full-rank, is singular

Since A = A', we have the further corollary

A square matrix of which any block-row is not of full rank, is singular.

Example

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 5 \end{bmatrix}$$

the block-column $\begin{bmatrix} 2 & 3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

is of order 2 by 3, but its rank is only 1.

Therefore the whole matrix is singular.

For this small matrix, the fact is readily verified by developing $|A|$ by the minors of the first row

$$A = 1\begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix} - 2\begin{vmatrix} 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 0 & 0 \end{vmatrix} + 3\begin{vmatrix} 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 0 & 0 \end{vmatrix} - 4\begin{vmatrix} 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 0 & 0 \end{vmatrix} = 0.$$

The following applications are also worthwhile to note

If A is a block-triangular matrix with square diagonal blocks, its determinant is the recursive product of the determinants of its diagonal blocks.

Example

$$A = \begin{bmatrix} 1 & -2 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 \\ 2 & 3 & 1 & -2 & 0 \\ 4 & 5 & -2 & 1 & 0 \\ 6 & 7 & 8 & 9 & 1 \end{bmatrix}$$

$$|A| = \begin{vmatrix} 1 & -2 \\ -2 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & -2 \\ -2 & 1 \end{vmatrix} \qquad |1| = (-3) \times (-3) \times 1 = 9.$$

The following application to what we might call a (square) "bordered block diagonal matrix"

e.g.

$$A = \begin{bmatrix} A_{11} & & & A_{14} \\ & A_{22} & & A_{24} \\ & & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

is typical for a class of applications of the "full rank" corollary. If the number of columns of (any of)the top left-hand block exceeds the number of rows in the same block plus the number of rows in the bottom block-row the whole matrix is singular.

Thus if A_{11} has more columns than A_{11} and A_{41} together contain rows, the first block-column of the whole matrix is of less than full rank, showing the matrix to be singular.

Example

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 & 5 \\ 0 & 0 & 0 & 6 & 7 \\ 0 & 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 \end{bmatrix}$$

$A_{11} = [1 \quad 2 \quad 3]$ contains 3 columns, but

$A_{11} = [1 \quad 2 \quad 3]$ and $A_{31} = [12 \quad 13 \quad 14]$

together only contains 2 rows

Therefore $A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 12 & 13 & 14 \end{bmatrix}$

although of order 5 by 3, is of rank 2 only, and the whole matrix is singular.

Theorem

Let an m by n matrix A be of less than full rank.

Let A be partitionable as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad (5.7.2)$$

where the top lefthand block A_{11} is square of order m_1 by m_1, and non-singular, where m_1 is less than the rank of the full matrix A, and cannot be chosen any bigger.

Then
there exists a matrix B, order m_1 by $n-m_1$, for which the following relation holds:

$$\begin{bmatrix} A_{1,1} \\ \\ A_{2,1} \end{bmatrix} B = \begin{bmatrix} A_{1,2} \\ \\ A_{2,2} \end{bmatrix} \qquad (5.7.3)$$

Proof

Consider the equations system

$$\begin{aligned} A_{1,1}\,\underline{x}_1 + A_{1,2}\,\underline{x}_2 &= 0 \\ A_{2,1}\,\underline{x}_1 + A_{2,2}\,\underline{x}_2 &= 0 \end{aligned} \qquad (5.7.4)$$

Because A is not of rull rank, a solution to (5.7.4) with $x \neq 0$ must be assumed to exist.

From (5.7.4) we obtain

$$\begin{aligned} \underline{x}_1 \quad + \quad & A_{1,1}^{-1} A_{1,2} \underline{x}_2 \quad = \quad 0 \\ \left[A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}\right] & \underline{x}_2 \quad = \quad 0 \end{aligned} \qquad (5.7.5)$$

We must assume $\left[A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}\right]$ is a zero matrix.

The contra-assumption if a non-zero element in this block would imply that we could have allocated the corresponding equation to the top block row and the corresponding element of $\underline{x}_2$ to $\underline{x}_1$, identifying a further non-zero pivot by which a larger block $A_{1,1}$ could be inverted.

Therefore

$$A_{2,1} A_{1,1}^{-1} A_{1,2} = A_{2,2} \qquad (5.7.6)$$

We now readily identify

$$B = A_{1,1}^{-1} A_{1,2} \qquad (5.7.7)$$

and verify

$$\begin{aligned} A_{1,1} A_{1,1}^{-1} A_{1,2} &= A_{1,1} B = A_{1,2} \\ A_{2,1} A_{1,1}^{-1} A_{1,2} &= A_{2,1} B = A_{2,2} \end{aligned} \qquad (5.7.8)$$

confirming (5.7.3)

q.e.d.

The following is simply a different formulation of the same theorem:

If A is not of full rank and contains a block-column which is of full rank and also of the same rank as A, all other columns of A, not belonging to that block-column, can be expressed as linear combinations of the full-rank block-column. (The columns of B describe the combinations.)

Corollary

All rows of A, not themselves belonging to a largest full-rank block-row, can be expressed as combinations of the full-rank block-row.

Furthermore, if $A_{1,1}$ is just any square and non-singular block, rather than specifically the biggest possible square and non-singular block, we have the following generalization.

A residual non-inverted block $[A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}]$ is of rank $r-m_1$ where r is the rank of A. Note that this formulation of the theorem also applies to matrices which are of full rank, i.e. if A is of full rank, then $[A_{22} - A_{21} A_{11}^{-1} A_{12}]$ is also of full rank.

Theorem

Let a square and non-singular matrix A be partitioned as follows

$$A = \left[\begin{array}{c|c|c} A_{1,1} & A_{1,2} & \\ \hline A_{2,1} & A_{2,2} & A_{2,3} \end{array}\right] \qquad (5.7.9)$$

$$A_{3,1} \quad A_{3,2} \quad A_{3,3}$$

(i.e. A contains at least one zero element which has been put in the top righthand corner.)

Then we may require the partitioning to be such that

a) $A_{1,1}$ is square and non-singular,

b) Either

b_1) $A_{2,2}$ is of order zero by zero, the partitioning being in fact $A = \left[\begin{array}{c|c} A_{1,1} & \\ \hline A_{3,1} & A_{3,3} \end{array}\right]$

or

b_2) A_{22} is square and $[A_{2,1}, A_{2,2}]$ is of full rank.

Proof

The non-singularity of A implies that the top block-row is of full rank, hence a).

Concerning b), we note that b_1) is automatically implied if the non-zero part of the top block-row is square.

(We may not assume that there are more rows than columns in the non-zero part of the top block-row). Otherwise, the invertability of A implies the existence of a non-singular block-pivot

$$P_{2,2} = [A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}]$$

The existence of such a block-pivot also implies the invertability of

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} ,$$

hence the full rank of $[A_{2,1} \; A_{2,2}]$.

q.e.d.

Corollary

If a square matrix A is partitioned as

$$A = \left[\begin{array}{c|c} A_{1,1} & \\ \hline A_{2,1} & A_{2,2} \end{array}\right]$$

non-singularity of A, if present, implies that the rank of $A_{2,1}$ is at least equal to the excess of the number of columns of $A_{1,1}$ over the number of rows of $A_{1,1}$.

Exercise 5.7a

For $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix}$

(which is not of full rank).

* Establish the rank
* Find a block-column of full rank.
* Express the rest of the matrix by (5.7.3), having first calculated B.

Exercise 5.7b

For $B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & - & - & - \\ - & 6 & - & - \\ 7 & 8 & 9 & - \end{bmatrix}$ (which is non-singular)

* Find suitable orderings to present the matrix in block-triangular form (several possible orderings)

* Check the rank of the off-diagonal blocks against the last theorem in this section (for more than one ordering)

* Calculate $|B|$, using square diagonal blocks.

(The re-orderings may, or may not, lead to a change in the sign of the determinant.)

Exercise 5.7c

Which of the following matrices C is the singular one? Calculate the absolute value of the determinant of the non-singular one. (Explain which theorem you used to prove singularity.)

$$C = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & - & - & 6 \\ 10 & - & - & 12 \\ 13 & 14 & 15 & 16 \end{vmatrix} \qquad C = \begin{vmatrix} - & 1 & 2 & 3 \\ 4 & - & - & 5 \\ 6 & - & - & 7 \\ 8 & 9 & 10 & 11 \end{vmatrix}$$

Calculated determinants:

$$||B|| = \begin{vmatrix} 5 & - & - & - \\ - & 6 & - & - \\ - & 8 & 9 & - \\ 7 & 2 & 3 & 4 \end{vmatrix} = 5 \times 6 \times 9 \times 4 = 30 \times 36 = 1080$$

It needed 3 times interchanging adjoining rows to get the top row at the bottom, therefore $|B| = -1080$.

$$||C|| = \begin{vmatrix} 4 & 5 & - & - \\ - & 7 & - & - \\ - & 3 & 1 & 2 \\ 8 & 11 & 9 & 10 \end{vmatrix} = \begin{vmatrix} 6 & 7 \\ 4 & 5 \end{vmatrix} \cdot \begin{vmatrix} 9 & 10 \\ 1 & 2 \end{vmatrix} = (28-30)\cdot(10-18) = 16$$

5.8 The adjoint and its relation to the all-integer elimination method

Let a square matrix be partitioned, as follows

$$A = \begin{bmatrix} A_{1,1} & \underline{a}_{1,n} \\ \underline{a}'_{n,1} & a_{n,n} \end{bmatrix} \tag{5.8.1}$$

We assume $|A_{11}| \neq 0$, $|A| \neq 0$.

The partial inversion of A was already performed in the previous section for the general case of 2 square diagonal blocks.

The application of the result from section 5.7 for A_{22} of order 1, gives us a formula for the n^{th} pivot

$$p_n = a_{n,n} - \underline{a}_{n,1} A_{11}^{-1} \underline{a}_{1,n} \tag{5.8.2}$$

Again assuming $|A_{11}| \neq 0$ and $|A| \neq 0$, we readily develop a formula for the bottom righthand element of an inverse. If the inverse is indicated by the letter B, this formula is:

$$b_{n,n} = (a_{n,n} - \underline{a}'_{n,1} A_{11}^{-1} \underline{a}_{1,n})^{-1} \tag{5.8.3}$$

(The element of the inverse is the reciprocal of the last pivot).

Comparison of (5.7.1) and (5.8.3) allows us to express the element of the inverse in $|A|$ and $|A_{11}|$

$$b_{n,n} = \frac{|A_{1,1}|}{|A|} \tag{5.8.4}$$

Provided $|A| \neq 0$ is true, formula (5.8.4) is in fact valid for $|A_{11}| = 0$ as well as may be shown by finding a solution to the following system:

$$\begin{aligned} A_{1,1}\,\underline{x}_1 + \underline{a}_{1,n}\,x_n &= \underline{b}_1 = [0] \\ \underline{a}'_{n,1}\,\underline{x}_1 + a_{n,n}\,x_n &= b_n = 1 \end{aligned} \tag{5.8.5}$$

For $|A| \neq 0$, $|A_{1,1}| = 0$, we must assume $\underline{a}_{1,n} \neq 0$, as otherwise the top block row of A would not be of full rank.

However, if $A_{1,1}$ is singular, we may require

$$A_{1,1}\,\underline{x}_1 = A_{1,1}\,\underline{u}_1 \quad \lambda = 0,\ \underline{u}_1 \neq 0,\ \lambda \neq 0,\ \underline{x}_1 \neq 0 \qquad (5.8.6)$$

Furthermore $\underline{a}'_{n,1}\,\underline{u}_1 = 0$ would imply

$$A \begin{bmatrix} \underline{u}_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}, \text{ showing the singularity of A.}$$

We may therefore assume $\underline{a}'_{n,1}\,\underline{u}_1 \neq 0$.

We now assign the value 0 to x_n, and obtain a consistent solution for λ from the last equation in (5.8.5)

$$\underline{a}'_{n,1}\,\underline{u}_1\,\lambda = 1$$

is resolved as

$$\lambda = 1/(\underline{a}'_{n,1}\,\underline{u}_1) \qquad (5.8.7)$$

We find that

$$\underline{x} = \begin{bmatrix} \underline{u}_1 \\ 0 \end{bmatrix} \lambda \qquad (5.8.8)$$

is the solution vector corresponding to (5.8.5). However, this is the last column of the inverse, showing that

$$|A_{11}| = 0 \text{ implies } b_{n,n} = 0.$$

Hence (5.8.4) although initially developed on the assumption that $A_{1,1}$ is invertable, is generally valid.

Note that in (5.8.4) $|A_{1,1}|$ is the determinant of $[A_{1,1}]$, a top left-hand block. It is also the minor associated with $a_{n,n}$. While we developed (5.8.4) by reference to block inversion as practiced in the previous section, it is now more practical to revert back to the notation which was used for minors in the rest of this chapter i.e.

$$b_{n,n} = \frac{|A_{n,n}|}{|A|} \qquad (5.8.9)$$

In fact, (5.8.9) is valid for all the diagonal elements of the inverse: Interchanging the n^{th} (last) element of x with the k^{th} (any) element of x while performing the same operation on the columns of A is compensated by a similar reordering of the rows of A, thereby bringing the k^{th} element of both A and the

inverse back on the diagonal, now is the n,n cell. This operation does not affect the signs of either $|A|$ or $|A_{n,n}|$.

For off-diagonal elements, we have the usual complication of the sign, and the generalization of (5.8.9) is

$$b_{i,j} = \frac{|A_{j,1}|}{|A|} (-1)^{i+j} \qquad (5.8.10)$$

Note also that the interchanging of the indices i and j in (5.8.10) implies transposition. This is because the function of rows and columns in terms of tableau-interpretation interchange as a result of the inversion operation. In the structural system $A\underline{x} = \underline{y}$, the rows are associated with the elements of $\underline{y}$ (the equations), the columns with the elements of $\underline{x}$. In the inverse form $\underline{y} = A^{-1}\,\underline{x}$ this is the other way round.

Example

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 11 & -7 & 1 \\ -5 & 3 & 1 \end{bmatrix} \qquad \text{(from section 3.9)}$$

We wish (for example) to obtain the $b_{2,1}$ element of the inverse. To be able to apply (5.8.4) directly rather than (5.8.10), we put the second variable at the end and similarly the first equation.

Initial System:				Transformed System:			
$2x_1 + 5x_2 + x_3$	$=$	y_1		$11x_1 + 8x_3 - 7x_2$	$=$	y_2	
$11x_1 - 7x_2 + 8x_3$	$=$	y_2		$-5x_1 + x_3 + 3x_2$	$=$	y_3	
$-5x_1 + 3x_2 + x_3$	$=$	y_3		$2x_1 + x_3 + 5x_2$	$=$	y_1	

The element of the inverse is obviously the same for both systems, but we can now apply (5.8.4) to

$$A = \begin{bmatrix} 11 & 8 & -7 \\ -5 & 1 & 3 \\ 2 & 1 & 5 \end{bmatrix}$$

i.e.

$$b_{2,1} = \frac{\begin{vmatrix} 11 & 8 \\ 15 & 1 \end{vmatrix}}{\begin{vmatrix} 11 & 8 & -7 \\ -5 & 1 & 3 \\ 2 & 1 & 5 \end{vmatrix}} = \frac{51}{319} = 0.160$$

In the above expression the numerator $\begin{vmatrix} 11 & 8 \\ -5 & 1 \end{vmatrix}$ is the same as the corresponding minor from the original matrix A: the vectors to be reordered are not included in the minor.

The denominator, however, has changed sign because the interchanging of columns 2 and 3 causes the sign to change once, and the reordering of the rows requires two single permutations. We could have left the second equation in place, and the sign of $|A|$ would then remain the same. If that device were adopted, the numerator would change sign, the minor now becoming

$$\begin{vmatrix} -5 & 1 \\ 11 & 8 \end{vmatrix}$$

The result of this section as developed so far may now be summarized in one sentence:

The inverse is the transpose of the matrix of co-factors, divided by the determinant

The transposed matrix of co-factors is (i.e. the inverse multiplied by the determinant), is known as the adjoint.

We are now in a position to explain some of the finer points behind the all-integer method of calculating the inverse of an all-integer matrix, which was discussed in section 3.9. Basically, this algorithm revolves around the calculation of the adjoint of a block-pivot. If we assume inversion along the main diagonal, the typical calculation tableau of this algorithm is as follows

x_1	x_2	=	$\underline{y}_1$	$\underline{y}_2$
δ_k I	$\delta_k A_{11}^{-1} A_{12}$		$\delta_k A_{11}^{-1}$	
	$\delta_k [A_{22} - A_{21} A_{11}^{-1} A_{12}]$		$-\delta_k A_{21} A_{11}^{-1}$	δ_k I

where δ_k is the determinant of the block-pivot A_{11}. At the beginning of each k^{th} step, the non-pivotal rows are multiplied by $|A_{k,k}|$. From the second step onwards, the ratio of the entries in the non-pivotal rows, to the entries in the corresponding fractional tableau, immediately after elimination, is $|A_{k-1,k-1}| \cdot |A_{k,k}|$. Hence the subsequent division by the previous multiplier. (N.B.: Here $|A_{kk}|$ is the determinant of a block-pivot of order k, not a minor.)

Exercise

$$\text{For } A = \begin{bmatrix} 1 & -1 & 2 \\ 3 & - & 4 \\ 5 & -5 & 6 \end{bmatrix}$$

which is a non-singular matrix), perform the following calculations

1) Calculate all the co-factors ($=|A_{ij}| \quad (-1)^{i+j}$, see section 5.1)

2) Write the adjoint of $A_{11} = \begin{bmatrix} 1 & -1 \\ 3 & - \end{bmatrix}$, and of A itself

3) Calculate the determinant and the inverse, using (5.8.10).

4) Invert the matrix using the all-integer elimination method. Check the correctness of the inversion by verifying the unit-matrix product properly.

5.9 Commented text of the adjoint all-integer elimination procedure

The programmed procedure offered in this section is an adaption of the procedure INVD from section 5.6. The differences between the procedure IADJ and INVD may be summarised as follows

a) The calculation tableau is specified as an integer array. (INVD expects a real array)

b) The end result is the determinant, and either the adjoint (for N=0) or the righthand side pre-multiplied by the adjoint. These results are integer and to obtain the inverse or the reduced form, a division by the determinant is required.

c) No search for the absolute largest pivot in any column is performed, only for a non-zero entry. Calculations with integer numbers are exact and no problem of containing rounding errors arise.

On most machines this procedure would, compared to an inversion procedure in real numbers, save a factor two in storage space, and be somewhat quicker. And it is exact. Against these comparative advantages stands one obvious drawback:

Integer numbers are in practice limited to a fixed number of digits, and overflow may arise, if the matrix is large.

The text of the procedure is now listed, as follows:

```
'PROCEDURE' IADJ(A,M,N,D);
'VALUE' N; 'INTEGER' 'ARRAY' A; 'INTEGER' M,N,D;
'BEGIN' 'INTEGER' I,J,R,K,NUM,OVERSC;
  'COMMENT'
  INVERSION BY ELEMENTARY ROW-OPERATIONS,
  WITH CALCULATION OF THE DETERMINANT AS BY-PRODUCT,
  USING THE ALL-INTEGER METHOD.
  N.B. THE END RESULT NEEDS SCALING BY THE DETERMINANT, AS
  IN FACT THE ADJOINT RATHER THAN THE INVERSE IS CALCULATED.;

  'IF' N=0 'THEN' 'BEGIN'
     'COMMENT'
     FOR N EQUAL ZERO THE INVERSE IS ASKED.
     THEREFORE WE EQUATE N TO M AND THE RIGHTHANDSIDE MATRIX
     TO THE UNIT MATRIX;
     N:=M;
     'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
       'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO' A[I,M+J]:=0;
       A[I,M+I]:=1; 'END'; 'END';

  INITIATE OVERSCALE AND D:
  OVERSC := D := 1;

  'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
     'COMMENT' FIRST FIND A NON-ZERO PIVOT;
     'FOR' I:=R 'STEP' 1 'UNTIL' M 'DO' 'IF' A[I,R] # 0
     'THEN' 'BEGIN' K:=I; 'GOTO' END OF PIVOTSEARCH; 'END';

     'BEGIN'
       'COMMENT'  IF THIS LOOP IS GONE THROUGH THE MATRIX
       HAS BEEN FOUND SINGULAR;
       'GOTO' END OF IADJ; 'END';

     END OF PIVOTSEARCH:
     'IF' R#K 'THEN' 'BEGIN'
       'COMMENT'
       IF THE PIVOT WAS FOUND BELOW RATHER THAN ON THE MAIN
       DIAGONAL INTERCHANGE ROWS AND CHANGE SIGN OF DETERMINANT;
       D:=-D;
       PERMUTE:
       'FOR' J:=R 'STEP' 1 'UNTIL' M+N 'DO' 'BEGIN'
         NUM:=A[R,J]; A[R,J]:=A[K,J]; A[K,J]:=NUM; 'END'; 'END';
```

```
   UPDATE:
   'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO'
   'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M 'DO'
   A[I,J] := A[I,J]*A[R,R];

   'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M 'DO'
   'IF' A[I,R]≠0 'THEN'
   'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO'
   A[I,J]:=A[I,J] - A[R,J]*A[I,R];

   'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M 'DO'
   'FOR' J:=R+1 'STEP' 1 'UNTIL' M+N 'DO'
   A[I,J] := A[I,J]/OVERSC;
   OVERSC := A[R,R];

   'END';

'IF' D < 0 'THEN'
'FOR' J:=M+1 'STEP' 1 'UNTIL' M+N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' A[I,J] := -A[I,J];

D := A[M,M]*D;


END OF IADJ:    'END';
```

5.10 The determinant of the product of two square matrices

Theorem

If A and B are square matrices, of equal order, $|AB|$ may be evaluated as

$$|AB| = |A| \cdot |B| \qquad (5.10.1)$$

Proof

We first consider some special cases

Case 1a

$|A| = 0$; therefore $\underline{r}'A = 0$, for some $\underline{r} \neq 0$
$\underline{r}'AB = 0$, proves $|AB| = 0$, q.e.d. for case 1a

Case 1b

$|B| = 0$; therefore $B\underline{k} = 0$, for some $\underline{k} \neq 0$
$AB\underline{k} = 0$ proves $|AB| = 0$, q.e.d. for case 1b

Case 2a

A is partitionable as

$$A = \left[\begin{array}{c|c} 1 & \underline{a}'_{12} \\ \hline & I \end{array}\right]$$

In that case A is an operator serving to add the combination $\underline{a}'_{12}B_2$ of the rows of B_2, the block-row of B consisting of the non-leading rows of B, to the leading row, therefore we find (see section 5.3)

$$|A| = 1, \quad |AB| = |B|, \text{ q.e.d. for case 2a.}$$

Case 2b

B is partitionable as

$$B = \left[\begin{array}{c|c} 1 & \\ \hline \underline{b}_{21} & I \end{array}\right]$$

Now B is an operator, serving to add a combination of the non-leading columns of A to the leading column of A. The proof follows analogeous to case 2a. q.e.d. for case 2b.

Case 3a

A is partitionable as

$$A = \left[\begin{array}{c|c} 1 & \\ \hline \underline{a}_{21} & I \end{array}\right]$$

In that case, A is an operator, serving to add multiples of the leading row of B, to the other rows

$$\left|\left[\begin{array}{c|c} 1 & 0 \\ \hline \underline{a}_{21} & I \end{array}\right] \left[\begin{array}{c|c} b_{11} & \underline{b}'_{12} \\ \hline \underline{b}_{21} & B_{22} \end{array}\right]\right| = \left|\left[\begin{array}{c|c} \underline{b}_{11} & \underline{b}'_{12} \\ \hline \underline{b}_{21}+\underline{a}_{21}b_{11} & B_{22}+\underline{a}_{21}\underline{b}'_{12} \end{array}\right]\right|$$

$$= |B|$$

q.e.d. for case 3a.

Case 3b

B is partitionable

$$B = \left[\begin{array}{c|c} 1 & \underline{b}'_{21} \\ \hline & \\ & I \end{array}\right]$$

will now be obvious.

We now consider the non-trivial case that A and B are both non-singular and have no special structure.

In that case we may, if necessary after re-ordering the rows of A (and C correspondingly) and/or the columns of B (and C correspondingly), assume without loss of generality that the leading elements of both A and B are non-zero.

(If A had a zero column, or B a zero row no further proof was required).

$$|A\ B| = \left|\left[\begin{array}{c|c} a_{11} & \underline{a}'_{12} \\ \hline \underline{a}_{21} & A_{22} \end{array}\right] \left[\begin{array}{c|c} b_{11} & \underline{b}'_{12} \\ \hline \underline{b}_{21} & B_{22} \end{array}\right]\right|$$

is then evaluated as $|AB| =$

$$\left|\left[\begin{array}{c|c} 1 & \\ \hline -\underline{a}_{21}a_{11}^{-1} & I \end{array}\right] \left[\begin{array}{c|c} a_{11} & \underline{a}'_{12} \\ \hline \underline{a}_{21} & A_{22} \end{array}\right] \left[\begin{array}{c|c} b_{11} & \underline{b}'_{12} \\ \hline \underline{b}_{21} & B_{22} \end{array}\right] \left[\begin{array}{c|c} 1 & -b_{11}^{-1}\underline{b}'_{12} \\ \hline & I \end{array}\right]\right|$$

$$= \left|\left[\begin{array}{c|c} a_{11} & \underline{a}'_{12} \\ \hline & A^*_{22} \end{array}\right] \left[\begin{array}{c|c} b_{11} & \\ \hline \underline{b}_{22} & B^*_{22} \end{array}\right]\right| = \left|\left[\begin{array}{c|c} a_{11}b_{11} + \underline{a}'_{12}\underline{b}_{21} & \underline{a}'_{12}B^*_{22} \\ \hline A^*_{22}\underline{b}_{21} & A^*_{22}B^*_{22} \end{array}\right]\right| \tag{5.10.2}$$

where $A^*_{22} = A_{22} - \underline{a}_{21}a_{11}^{-1}\underline{a}'_{12}$ (5.10.3)

and $B^*_{22} = B_{22} - \underline{b}_{21}b_{11}^{-1}\underline{b}'_{12}$ (5.10.4)

In (5.10.2), the first second and third members are equal to each other (=|AB|) on account of the application of special cases 3a and 3b to the second member.

For $|A^*_{22}| = 0$, we find $|A| = 0$, (see section 5.7) and special case 1a is applicable, therefore no further proof is required for $|A^*_{22}| = 0$.

For $|A^*_{22}| \neq 0$, the fourth member of (5.10.2) is now evaluated (using special case 2a), as

$$|AB| = \left|\left[\begin{array}{c|c} 1 & -\underline{a}'_{12}A^{*-1}_{11} \\ \hline & I \end{array}\right] \left[\begin{array}{c|cc} a_{11}b_{11} + \underline{a}'_{12}\underline{b}_{21} & & \underline{a}'_{12}B^*_{22} \\ \hline A^*_{22}\underline{b}_{2} & A^*_{22}B^*_{22} & A^*_{22}B^*_{22} \end{array}\right]\right|$$

$$= \left|\left[\begin{array}{c|c} a_{11}b_{11} & \\ \hline A^*_{22}\underline{b}_{2} & A^*_{22}B^*_{22} \end{array}\right]\right| \tag{5.10.5}$$

The most righthand member of (5.10.5) is now evaluated by development by the top row as

$$|A\ B| = (a_{11}b_{11})|A^{*}_{22}\ B^{*}_{22}| \qquad (5.10.6)$$

Now assume that if A and B are of order n, the theorem is valid for matrices of orders up to n-1, (i.e. of orders 1,2, ... n).

If so, $|A\ B|$ may be evaulated by the righthand-side of (5.10.6) as

$$|A\ B| = (a_{11}b_{11})|A^{**}_{22}| \cdot |B^{*}_{22}| = (a_{11}|A^{*}_{22}|)(b_{11}B^{*}_{22}|)$$
$$= |A| \cdot |B| \qquad (5.10.7)$$

Since for square matrices of order 1, the theorem merely states the identity $ab \equiv a.b$, its general proof now follows by recursive induction.
q.e.d.

Example

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}$$

$$|A| = 4 - 6 = -2, \quad |B| = 4 + 6 = 10, \quad |A| \cdot |B| = -20$$
$$|AB| = \left|\begin{bmatrix} 7 & 6 \\ 15 & 10 \end{bmatrix}\right| = 70 - 90 = -20.$$

Exercise

Verify that $|A\ B\ C| = |A|\ |B|\ |C|$

$$\left|\begin{bmatrix} 3 & 2 \\ 4 & -1 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 2 & -1 \end{bmatrix}\right| = -363$$

by two separate calculations, i.e. by evaluating $|A|$, $|B|$, and $|C|$, and by calculating ABC and extracting its determinant. $|ABC|$

Part II

GRAPHS AND LINEAR PROGRAMMING

CHAPTER XII

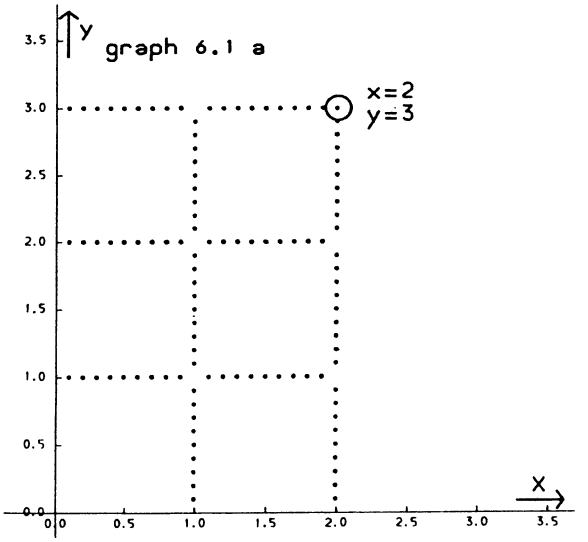
CHAPTER XIII

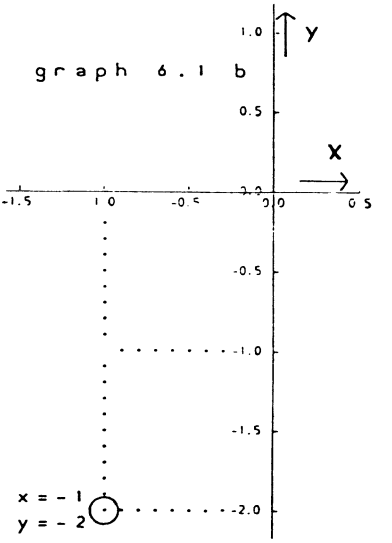
CHAPTER VI

VECTORS AND COORDINATE-SPACES

6.1 The coordinate-plane

The coordinate-plane is a useful visual aid when for one reason or another, numbers are ordered in pairs, i.e. for every number x, there is a corresponding number y. Many properties of two-dimensional coordinate-mappings (two-dimensional vectors), can be generalized to vectors of any order. Pairs of numbers are represented as points in the coordinate plane. It is convention to indicate one variable by the "right" direction and one variable by the "upward" direction. Thus x=2, y=3 is presented as a point which is 2 units of distance to the right and 3 units upwards from the origin.





In the above graphical mapping of x=2, y=3 the horizontal line is the x-axis, the vertical line is the y-axis as indicated by the arrows. All points on the x-axis have y=0 in common, and all points on the y-axis have x=0 in common. The point (0,0) (the intersection of the two axes) is called the origin. Since x is indicated by the right-hand direction, all points to the left of the y-axis have negative x-coordinates. Similarly, all points below the x-axis have negative y-coordinates, because y is indicated by the upward direction. Hence x=-1, y=-2 is in the bottom left-hand part of the coordinate plane. A

common notation for a two-dimensional vector is to use brackets, e.g. (2,3) means x=2, y=2 and (-1,-2) means x=-1, y=-2.

Exercise
(and preparation for later sections)

Make two series of mappings of pairs of points (x,y) in the x,y coordinate plane (best use proper graph paper).

Series 1:
(-3,6), (0,4), (3,2), (6,0).

Series 2:
(-6,14), (-5,7), (-4,2), (-3.42,0), (-3,-1), (-2,-2), (-1,-1), (-0.58,0), (0,2), (1,7), (2,14).

6.2 Functions and relations

A function is a relation, a "law" which tells us the value of one variable if we know the value of some other variable or variables. We would normally think in terms of an algebraically describable function e.g. $y = 7x^2 + 5$, $z = 2pq + 4$ etc. This is not strictly implied by the definition: "the temperature in Trafalgar Square" is a proper function of time. If we know the time we can establish the temperature in Trafalgar Square, provided someone takes the trouble of looking at a thermometer.

The one variable is the dependent variable or function value, the other variable(s) are the arguments of the function. Two or more variables may also be linked by a relation. A relation is a condition imposed on the values of two or more variables e.g. $x + y + z = 20$.

It is often, but not always, possible to formulate a relationship between some variables as one variable being a function of the others, (also referred to as explanatory variables).

$x + y + z = 20$ can be written as $x = 20 - y - z$, but $x^2 + y^2 = 4$ yields 2 values for x if we know the value of y.
$x=\pm\sqrt{4-y^2}$ is not a proper function.

We therefore say that $x + y + z = 20$ can be written as x being an explicit function of y and z. For $x^2 + y^2 = 4$ it is not possible to write x as an explicit function of y.

Relationships between only two variables can be mapped in the two-dimensional coordinate plane. This is so, irrespective of whether or not they can be written explicitly. The combination

of the points which satisfy the relation will normally constitute a line, and the shape of that line is characteristic for the relation.

6.3 How to draw the graph of a linear relation

The relation

$$a.x + b.y = c$$
$$(a \neq 0,\ b \neq 0)$$

can be written explicitly, either as x being a function of y, or as y being function of x.

The two explicit functions are:

$$x = c/a - b/a \,.\, y \qquad (6.3.2)$$

and

$$y = c/b - a/b \,.\, x \qquad (6.3.3)$$

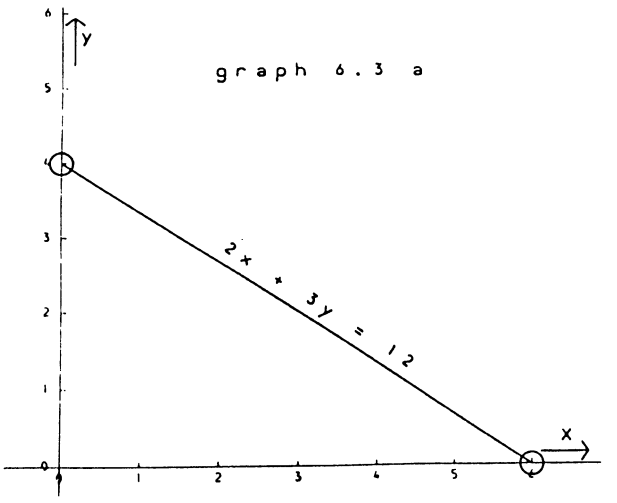
This establishes 2 points of the graph without further calculation. For x=0 (6.3.3) gives y=c/b, and for y=0 (6.3.2) gives x=c/a.

Example

$$2x + 3y = 12$$

This can be written explicitly as follows:

$$x = 6 - 1\tfrac{1}{2}y;\ y=0 \rightarrow x=6$$
$$y = 4 - 2/3x;\ x=0 \rightarrow y=4$$



The graph makes use of the fact that (as the name suggests) a linear relationship is represented in the coordinate plane by a straight line. This point will be discussed in more detail in the next section.

If a relation is already given in the explicit form, i.e.

$$y = a.x + b \qquad (6.3.4)$$

the graphical mapping is even simpler. The point x=0 with y=b is immediately obvious. The second point can be obtained by writing the relation in the form of x being a function of y, i.e.

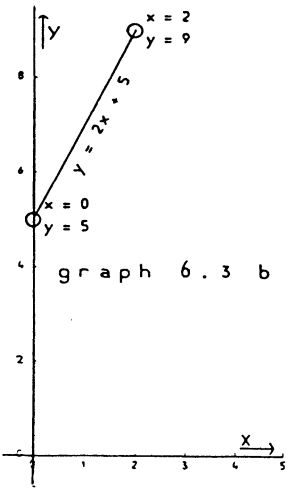
$$x = 1/a\ y - b/a \qquad (6.3.5)$$

But even simpler is the point x=1 with y=a+b, or, if a small scale is used, (for example)

$$x = 2 \text{ with } y = 2a + b$$

Example

$$y = 2x + 5$$



If a function is presented in the explicit form with the function-value on the vertical axis, the slope of the line is related to the coefficient for the explanatory variable. This coefficient is the tangent of the angle between the line and a horizontal line, e.g. the x-axis. Hence in the example, this angle is 60^{o}, since $\tan 60^{o}=2$.

6.4 Vectors, points and linear subspaces

Those who are willing to take the equivalence between two-dimensional graphical illustration, - which will be frequently

used in the rest of this book, and the general n-dimensional generalization of the illustrated properties for granted, may wish to skip the formal proofs in this section.

In the previous section we made use of the fact that if two points on a straight line satisfy the same linear relation, then all the points on the line satisfy that same relation. The algebraic equivalent of this property is the following:

Theorem

Let two vectors of order n,

$$\underline{x} = \underline{x}^{*} \text{ and } \underline{x} = \underline{x}^{**}$$

both satisfy the linear relation

$$\underline{a}'\underline{x} = b \tag{6.4.1}$$

where $\underline{a}'$ is a vector of known coefficients and b is a constant.

Then, any combination of these two vectors,

$$\underline{x}^{***} = p\,\underline{x}^{*} + (1-p)\underline{x}^{**} \tag{6.4.2}$$

(p any real scalar)

will also satisfy (6.4.1).

Proof

By assumption, (6.4.1) may be written for

$\underline{x} = \underline{x}^{*}$ as

$$\underline{a}'\,\underline{x}^{*} = b \tag{6.4.3}$$

and for $\underline{x} = \underline{x}^{**}$ as

$$\underline{a}'\,\underline{x}^{**} = b \tag{6.4.4}$$

From (6.4.3), we obtain

$$p\,\underline{a}'\,\underline{x}^{*} = p\,b \tag{6.4.5}$$

and from (6.4.4)

$$(1-p)\,\underline{a}'\,\underline{x}^{**} = (1-p)b \tag{6.4.6}$$

Together (6.4.5) and (6.4.6) yield, considering (6.4.2),

$$\underline{a}' \underline{x}^{***} = p\, \underline{a}' \underline{x}^{*} + (1-p)\, \underline{a}' \underline{x}^{**} = p.b. + (1-p).b = b \qquad (6.4.7)$$

q.e.d.

The main reason why the above proof (which is hardly more than a re-statement of the definition of a combination) cannot simply be applied to the line is that we do not normally define a line as the combination of points satisfying (6.4.1). Indeed, when applying geometry we rarely refer to any definition of a straight line. By Euclid's postulate, the shortest route between two points is the straight line. This defines a straight connecting line between two points.

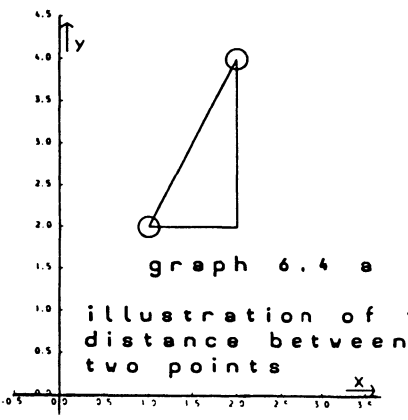
The concept of distance can be defined algebraically and measured geometrically. The distance between two vectors is the square root of the sum of the squares of the differences between their corresponding elements.

Hence we can write the distance between two vectors $\underline{x}^{*}$ and $\underline{x}^{**}$ as

$$\left|\underline{x}^{*} - \underline{x}^{*}\right| = \sum_{j=1}^{n} (x_j^{*} - x_j^{**})^2 \qquad (6.4.8)$$

Example

The distance between
x=1, y=2, and
x=2, y=4



graph 6.4 a
illustration of the distance between two points

The rectangular sides of the marked triangle are the differences between the coordinates of the two points, i.e. 2-1=1 in the horizontal x-coordinate direction and 4-2=2 in the vertical y-coordinate direction. Therefore, by Pythagoras' Theorem, the distance between the two points is

$$\sqrt{2^2 + 1^2} = \sqrt{5}$$

Theorem

Let (x^{*}, y^{*}); (x^{**}, y^{**}) and (x^{***}, y^{***}) be three points in the two-dimensional x,y coordinate plane. And let (x^{***}, y^{***}) be related to the other two points:

$$\left.\begin{aligned} x^{***} &= p \cdot x^{*} + (1-p)\, x^{**} \\ y^{***} &= p \cdot y^{*} + (1-p)\, y^{**} \end{aligned}\right\} \qquad (6.4.9)$$

$$(0 \leq p \leq 1)$$

Then

$$|(x^{***}, y^{***}) - (x^{*}, y^{*})| + |(x^{***}, y^{***}) - (x^{**}, y^{**})| = |(x^{*}, y^{*}) - (x^{**}, y^{**})| \qquad (6.4.10)$$

i.e. the shortest route from x^{*}, y^{*} to x^{**}, y^{**} as indicated by the righthand side of (6.4.10) is via x^{***}, y^{***} and one obtains the total distance by addition of the two separate distances.

Proof

Evaluate the first term in the left-hand side expression of (6.4.10) by the definition of a distance (6.4.8)

$$|(x^{***}, y^{***}) - (x^{*}, y^{*})| = \sqrt{(x^{***} - x^{*})^2 + (y^{***} - y^{*})^2} \qquad (6.4.11)$$

Substitution of the right-hand sides of (6.4.9) for x^{***} and y^{***} into (6.4.11) gives us

$$|(x^{***}, y^{***}) - (x^{*}, y^{*})| = \sqrt{(p.x^{*} + (1-p)x^{**})^2 + (p.y^{*} + (1-p)y^{**} - y^{*})^2}$$

$$= \pm(1-p)\sqrt{(x^{*} - x^{**})^2 + (y^{*} - y^{**})^2}$$

$$= (1-p)|(x^{*}, y^{*}) - (x^{**}, y^{**})| \qquad (6.4.12)$$

For $(0 \leq p \leq 1)$, the positive sign is applicable, otherwise the distance would become negative. The same method of evaluation, applied to the distance between (x^{***}, y^{***}) yields

$$|(x^{***}, y^{***}) - (x^{**}, y^{**}) = p|(x^{*}, y^{*}) - (x^{**}, y^{**})| \qquad (6.4.13)$$

Together (6.4.12) and (6.4.13) give the desired result, i.e.

$$|(x^{***}, y^{***})-(x^{*}, y^{*})|+|(x^{***},y^{***})-(x^{**},y^{**})|=$$
$$(1-p)\,|(x^{*}, y^{*})-(x^{**}, y^{**})|+p|(x^{*}, y^{*})-(x^{**}, y^{**})|=$$
$$|(x^{*}, y^{*})-(x^{**}, y^{**})| \qquad (6.4.14)$$

which is equivalent to (6.4.10.)
q.e.d.

This proof cannot be generalized directly to an n-dimensional coordinate space. This is because we took our definition of a straight line from Euclid's postulate.

The algebraic equivalent of a linear relation in 3-dimensional space is a flat plane, Euclid's postulate is applicable to a line. It is more practical to turn the proposition round.

In the n-dimensional $\underline{x}$ coordinate-space, the set of point-vectors $\underline{x}$ (of order n), which satisfy

$$\underline{a}'\,\underline{x} = b \qquad (6.4.1)$$

constitutes a Euclidean (or linear) subspace, of order n-1.

The elements of the n-dimensional row-vector $\underline{a}'$ and the constant b determine the position of the subspace within the whole of the n-dimensional space.

In the three-dimensional coordinate-space, a 2-dimensional linear subspace is more commonly referred to by its geometrical name: a flat plane. Similarly, in the two-dimensional coordinate space, a one-dimensional linear subspace is usually referred to by its geometrical name: a straight line.

With these definitions, we don't any more need to prove that 3 points in the two-dimensional coordinate plane, which satisfy a common linear relation are on a straight line. That's how we defined the straight line in the first place. Similarly, any set of three-dimensional vector-points satisfying the same linear equation, fit in the same flat plane.

The following result although slightly less intuitively obvious, can now be picked up as well:

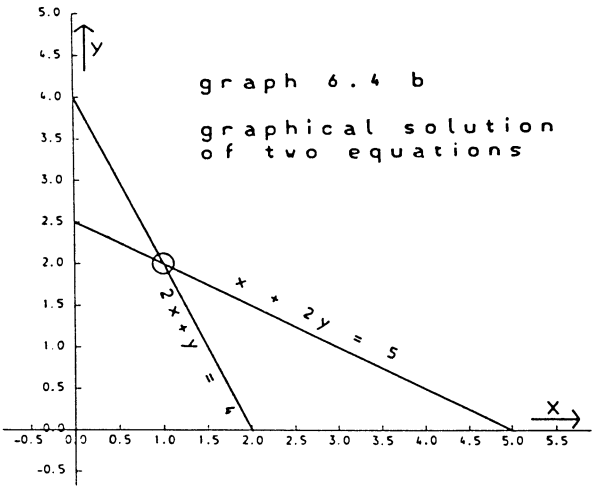
All points which satisfy two independent equations in 3-dimensional space, are on the same straight line.

That line is the intersection between the planes associated with the two equations. The line is a one-dimensional subspace in both of the two-dimensional subspaces defined by the two equations.

Similarly, the solution of a system of 2 equations and 2 unknowns, is the point of intersection of the two corresponding lines.

Example

$$\begin{aligned} x + 2y &= 5 \\ 2x + y &= 4 \end{aligned}$$



We find the intersection of the two lines at the point x = 1, y = 2. This is the solution of the system.

In 3-dimensional space, the solution of a system of 3 equations is found as the cornerpoint where the 3 planes intersect. If the system is singular, no unique intersection will exist. In the two-dimensional case, contradictory equations correspond to parallel lines, dependent equations to overlapping lines. By analogy to the linear subspace of order n-1, we may call the set of n-dimensional vectors, which satisfy k independent equations, an n-k dimensional Euclidean subspace. Clearly an n-dimensional vector which belongs to an n-k dimensional subspace i.e. is required to satisfy k independent linear equations, is fully determined, once we enumerate n-k elements

The remaining k elements may then be found by back-substitution.

6.5 Restrictions and convex sets

A restriction is a statement or property concerning vectors, which is defined in some coordinate-space. We would normally assume a restriction to be stated in the form of some function of the vectors in that coordinate-space to attain certain prescribed values.

Restrictions can be in the form of equations i.e. some function to attain a certain value and no other, or they can be in the form of inequalities, i.e. some function to be for example, greater than or equal to a certain number.

Thus $x^2 + 2y + z = 20$ is a restriction of equation-type and $x^2 + 2y + z \leq 20$ is a restriction of inequality-type in the x,y,z coordinate space.

We will normally assume a restriction to be non-trivial, that is, to define a characteristic of the vector not satisfied by all vectors in the relevant coordinate-space.

An example of a trivial restriction would be $(x - y)^2 \geq 0$. There are no pairs of numbers x and y, which don't satisfy this restriction.

Triviality may also arise relative to some other restrictions, i.e.

$$x \cdot y \geq 0$$

is not by itself trivial, but it is a trivial restriction if the restrictions $x \geq 0$, $y \geq 0$ are already present when $x \cdot y \geq 0$ is added to the list of restrictions.

A set of vectors is the collection of the vectors which satisfy some particular characteristic or characteristics. Normally those characteristics will be stated restrictions, equations and/or inequalities. A vector may be a (non-negative) combination of two or more other vectors.

x is a non-negative combination of $\underline{v}_1, \underline{v}_2 \cdots \underline{v}_k$ if

$$\underline{x} = p_1 \underline{v}_1 + p_2 \underline{v}_2 + \cdots p_k \underline{v}_k \qquad (6.5.1)$$

is true for some set of non-negative numbers

$p_1, p_2 \cdots p_k$, satisfying

$$p_1 + p_2 + \ldots\ldots p_k = 1 \qquad (6.5.2)$$

We made use of the concept of a combination in the previous section, for the special case of a two-dimensional vector. If for some set of vectors it is true that every non-negative combination of some vectors in the set is also part of the set, then such a set of vectors is named a convex set. (For this reason non-negative combinations are sometimes also indicated as convex combinations.)

An example of a non-convex set of vectors is the set of vectors whose elements are integer numbers.

This is shown by finding

$$\underline{x} = \begin{bmatrix}1\\2\end{bmatrix} \text{ as well as } \underline{x} = \begin{bmatrix}2\\1\end{bmatrix},$$

to belong to the set,

but then combination $\underline{x} = \frac{1}{2}\begin{bmatrix}1\\2\end{bmatrix} + \frac{1}{2}\begin{bmatrix}2\\1\end{bmatrix} = \begin{bmatrix}1\frac{1}{2}\\1\frac{1}{2}\end{bmatrix}$

does not belong to the set of vectors with integer elements

Theorem

The set of vectors satisfying

$$\underline{a}' \underline{x} \leq b \qquad (6.5.3)$$

is a convex set

Proof

Take some set of k vectors satisfying (6.5.3) i.e.

$$\underline{a}' \underline{x}_q \leq b \qquad (6.5.4)$$

for $q = 1, 2 \ldots k$

A non-negative combination of the $\underline{x}_q$ is defined by stating the values of the proportions of the combination to be derived from any $\underline{x}_q$, i.e. $p_1, p_2 \cdots p_k$

From (6.5.4) we derive

$$p_1 \underline{a}'\underline{x}_1 + p_2 \underline{a}' \underline{x}_2 + \ldots p_k \underline{a}' \underline{x}_k \leq (p_1 + p_2 + \ldots p_k)\, b = b \tag{6.5.5}$$

which proves that the combination (the left-hand side of (6.5.5)) satisfies (6.5.3)
q.e.d.

Theorem

If $\underline{x}$ satisfies

$$\underline{a}'\underline{x} \leq b \tag{6.5.3}$$

and for some other vector $\underline{y}$ the alternative relation

$$\underline{a}' \underline{y} \geq b \tag{6.5.6}$$

is true,
then, for some non-negative combination of $\underline{x}$ and $\underline{y}$, to be indicated as $\underline{z}$

$$\underline{a}' \underline{z} = b \tag{6.5.7}$$

is true.

Proof

State the actual values of

$$\underline{a}' \underline{x} = \Phi \tag{6.5.8}$$

and

$$\underline{a}' \underline{y} = \psi \tag{6.5.9}$$

By assumption $\Phi - b$ is a non-positive scalar, $\psi - b$ is a non-negative. Therefore a non-negative combination of them, a number in the interval $\Phi-b, \psi-b$, is zero

$$p_1(\Phi-b) + p_2(\psi-b) = 0 \tag{6.5.10}$$

or equivalently

$$p_1\Phi + p_2\psi = b \tag{6.5.11}$$

From (6.5.8) (6.5.9) and (6.5.11) we derive

$$p_1 \underline{a}' \underline{x} + p_2 \underline{a}' \underline{y} = \underline{a}' (p_1 \underline{x} + p_2 \underline{y}) = b \qquad (6.5.12)$$

Which identifies $\underline{z}$ as

$$\underline{z} = p_1 \underline{x} + p_2 \underline{y} \qquad (6.5.13)$$

showing the existence of such a vector q.e.d.

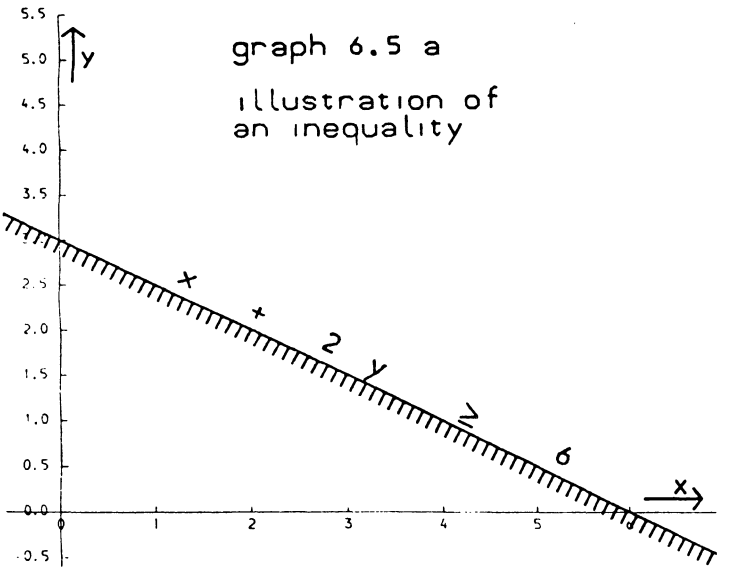
Obviously, if the supposed relationships (6.5.3) and (6.5.6) are put in the stronger < and > form, and the word "non-negative" is replaced by "positive non-zero" throughout the corresponding stronger theorem is also true.

The above theorems appear somewhat trivial, but they provide a strict proof for what is more or less intuitively obvious.

If we draw a line representing an equation, then this line will separate the coordinate space in two parts. On the one side of the equation the inequality of the $\geq$ type is satisfied, on the other the inequality of the $\leq$ type is

Example

$x + 2y \geq 6$



We first draw the graph of $x + 2y = 6$ (see section 6.3), and we then have to indicate at which side of the equation $x + 2y \geq 6$ is true, and at which side $x + 2y \leq 6$ is true. For $x = 0$, $y = 0$, the origin we find $0 + 0 \geq 6$ untrue.

Therefore the admissable side is the top right-hand side. We will indicate the non-admissable side by shading the line as a "wall" which inhibits access from the admissable to the inadmissable side.

Since there always is an "equal to" point between two points where the opposite signs i.e. $\geq$ and $\leq$ are true the relation $x + 2y \geq 6$ is untrue for all points below and to the left of the line. And, at least as important, we can generalise this property to the n-dimensional case.

We will therefore call the set of n-dimensional vectors satisfying a particular linear inequality e.g. (6.5.3) a half-space.

Any linear subspace of order n-1 splits the n-dimensional coordinate space, in which it is defined, in two half-spaces.

The set of vectors which satisfies a set of restrictions, either equations of inequalities, consist solely of vectors which satisfy each of them. Therefore any combination also satisfies each of them. This gives rise to the following

Corollary

The set of vectors which satisfy a particular combination of linear restrictions (equations and/or inequalities), is a convex set.

If we combine "convex" and "non-convex" restrictions we cannot, in strict logic say very much about the convexity or the non-convexity of the set of vectorpoints which satisfies them all.

Therefore, it appears prudent to assume that a set of vectors which is defined by a requirement that a number of restrictions are to be satisfied, is a convex set, only if each separate restriction also defines a convex set.

6.6 Graphical mapping of some non-linear functions and restrictions in two-dimensional space

Firstly, we consider the polynomial function

$$y = a_o + a_1 x + a_2 x^2 + \ldots a_p x^p \qquad (6.6.1)$$

There will be intersections with the horizontal x-coordinate axis at up to p points, these points being the real roots of the polynomial equation:

$$y = a_o + a_1 x + a_2 x^2 + \ldots a_p x^p = 0 \qquad (6.6.2)$$

If the equation has no real roots, the graph will not meet the horizontal axis.

There will be up to p-1 points where the graph is horizontal. These points of horizontal slope are the real roots of the polynomial equation:

$$\frac{dy}{dx} = a_1 + 2a_2 x + \ldots + pa_p x^{p-1} = 0 \qquad (6.6.3)$$

Similarly, there will be up to p-2 points where the curvature of the graph changes direction.

For a polynomial of second order the graph is a parabola, and the mapping is made somewhat easier by its symmetry property.

The quadratic function is

$$y = a_o + a_1 x + a_2 x^2 \qquad (6.6.4)$$

First we note that for p=2 (6.6.3) gives us a linear equation

$$\frac{dy}{dx} = a_1 + 2a_2 x = 0 \qquad (6.6.5)$$

i.e. for $a_2 \neq 0$ we find a horizontal point at

$$x = -\tfrac{1}{2} a_1/a_2 \qquad (6.6.6)$$

The qualification "for $a_2 \neq 0$" is trivial, since $a_2 = 0$ implies a linear function.

The corresponding value for y is found by evaluating (6.6.4) for $x = \frac{1}{2} a_1/a_2$

$$y = a_o - \tfrac{1}{2}a_1^2/a_2 + \tfrac{1}{4}a_1^2/a_2 = a_o - \tfrac{1}{4}a_1^2/a_2 \qquad (6.6.7)$$

The graph is symmetrical to a vertical axis going through the point of horizontal slope. This is shown by rewriting the function. From (6.6.4) we may obtain

$$y = a_2(x + \tfrac{1}{2}a_1/a_2)^2 + a_o - \tfrac{1}{4}a_1^2/a_2 \qquad (6.6.8)$$

Example

$$y = 2 + 4x + x^2$$

We find the point at which the graph is horizontal, by requiring, according to (6.6.3) and (6.6.5),

$$\frac{dy}{dx} = 4 + 2x = 0$$

$$\text{i.e. } x = -2$$

For this value of x, we evaluate y as

$$y = 2 - 4 \times 2 + 4 = -2$$

We now have found the horizontal point $x = -2$, $y = -2$. The graph will then be symmetric relative to the vertical line $x = -2$. For $x = 0$, we find $y = 2$. The symmetry property around the vertical line $x = -2$ then tells us that $x = -4$, $y = 2$ is also a point of the graph.

We find two more points by assigning a predetermined value to y. In this case $y = 0$ gives rise to real roots i.e.

$$x^2 + 4x + 2 = 0$$

has the roots

$$x = -2 \pm \tfrac{1}{2}\sqrt{16-8} = -2 \pm \sqrt{2}$$

should the zero value for y not supply real roots, a different pre-assigned value of y should be taken.

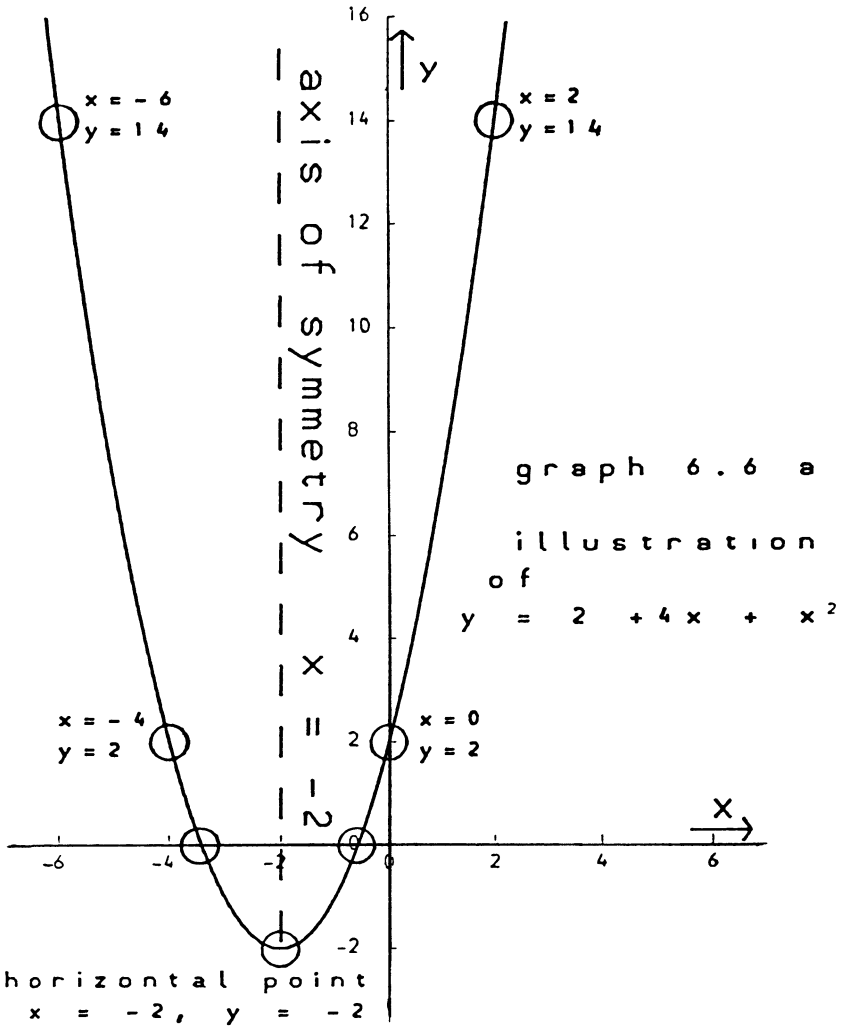
It is in any case desirable to establish a third pair of points on the graph. For $y = 14$ we find

$$14 = x^2 + 4x + 2$$

$$16 = x^2 + 4x + 4 = (x+2)^2$$

$$x + 2 = \pm 4 \rightarrow x = -2 \pm 4$$

It is now possible to draw the graph with reasonable accuracy and smoothness through these points.



graph 6.6 a illustration of $y = 2 + 4x + x^2$

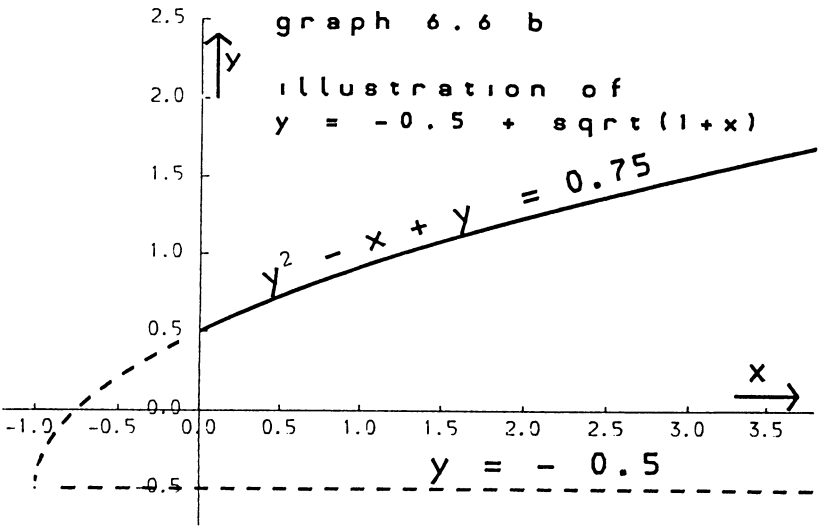
An immediate generalization of the quadratic function is the square root function.

$$y = a + \sqrt{x - b} \qquad (6.6.9)$$

$$(x \geq b)$$

The graph of this function is (the upper half of) a parabola with a horizontal axis of symmetry, $y = a$, and a vertical point at $x = a$, $x = b$. For $a < 0$, $b > 0$, this relationship is often useful to represent mildly curvelinear relations, which become flatter for higher values of the function-argument.

The function $y = -\frac{1}{2} + \sqrt{x + 1}$ is mapped in graph 6.6b



The "normal" presentation of the quadratic function may be obtained from (6.6.9) by bringing the constant term a over to the left-hand side, taking the square of both sides, and reordering thereafter

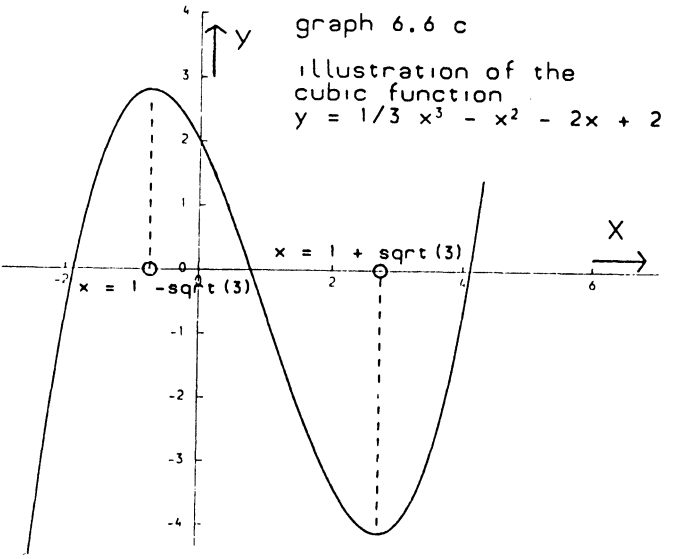
$$(y-a)^2 = y^2 - 2ay + a^2 = x-b,$$

or equivalently,

$$x = y^2 - 2ay + a^2 + b \qquad (6.6.10)$$

The one essential difference between (6.6.4) and (6.6.9) is that x, instead of y has become the variable which is a unique function of the other variable.

A polynomial function of the third degree is something like S-shaped, except that, if the function-value is indicated by the vertical direction, the S is turned on its side. The graph below gives a mapping of $y = 1/3x^3 - x^2 - 2x + 2$.



For sufficiently large values of x (large positive or very negative figures, the direction of the slope of the line is dominated by the term in x^3.

Thus, if the coefficient of x^3 is positive, the graph goes from bottom left to top right. In the middle, there is generally a dent and this may or may not go as far as a change in the direction of the slope.

If the cubic polynomial function itself is

$$y = a\,x^3 + b\,x^2 + c\,x + d,$$

the condition for the temporary vanishment of the slope is

$$\frac{dy}{dx} = 3\,a\,x^2 + 2\,b\,x + c = 0$$

In the example at hand, this equation is

$$\frac{dy}{dx} = x^2 - 2x - 2 = 0$$

$$\text{or} \quad x^2 - 2x + 1 = (x - 1)^2 = 3$$

This equation has two real roots,

$$x = 1 - \sqrt{3} = -0.41 \text{ and } x = 1 + \sqrt{3} = 2.41$$

In the interval between the roots, the slope has the opposite direction from the one indicated by the sign of the cubic term. For certain values of the coefficients the "dent in the middle" may not be enough to cause a reversal of the slope.

For example:

$$y = 1/3\, x^3 - x^2 + 2x + 1$$

would have a zero slope for

$$\frac{dy}{dx} = x^2 - 2x + 2 = 0$$

$$\text{or} \quad x^2 - 2x + 1 = (x + 1)^2 = -1$$

i.e. for no real value of x.

In such a case, the direction of the slope will be monotonously upwards (or downwards if the coefficient of x^3 is negative.)

We now discuss the mapping of

$$(x - a)^2 + (y - b)^2 = c \qquad (6.6.11)$$

$$(c > 0)$$

This is a circle with x = a, y = b, as origin, and a radius of c.

Example

$$(x - 3)^2 + (y - 4)^2 = 4,$$

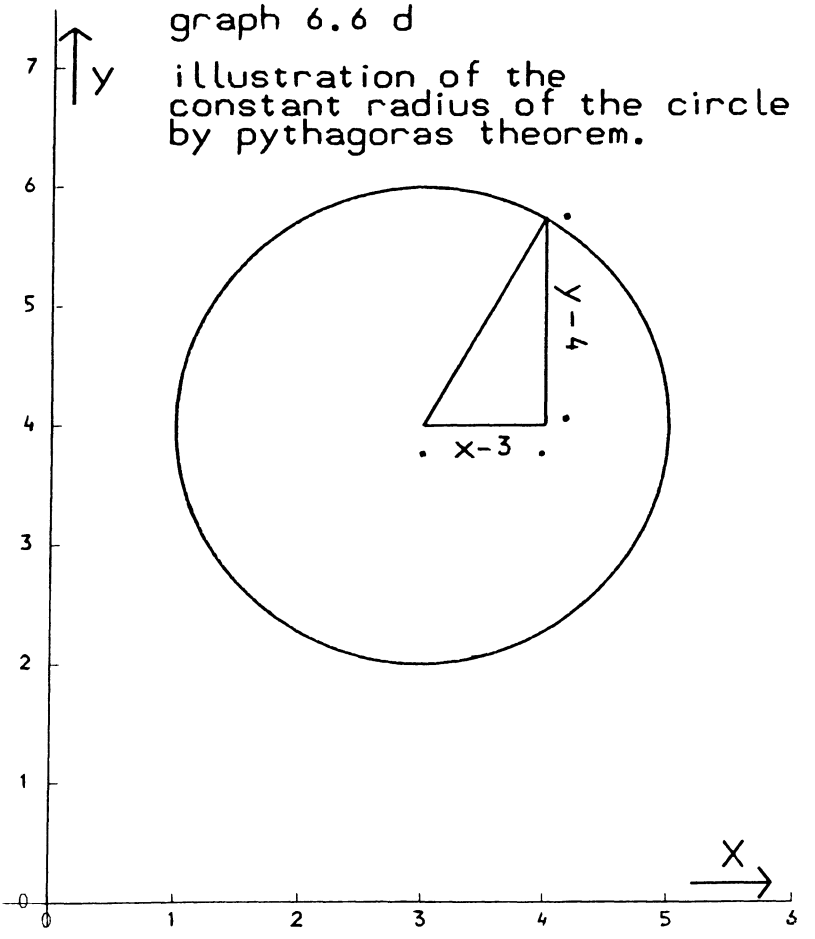
or equivalently

$$x^2 + y^2 - 6x - 8y + 21 = 0$$

That the graph of $(x - a)^2 + (y - b^2) = c$ is indeed a circle is shown by applying Pythagoras' Theorem.

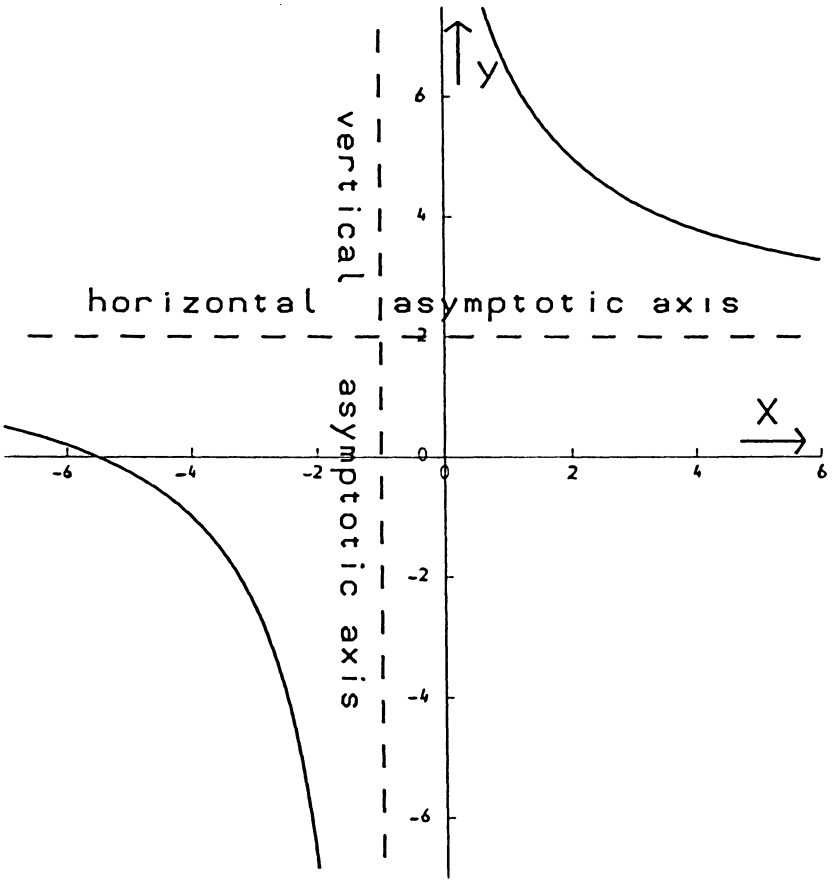
The horizontal and vertical sides of the triangle drawn in the graph are $x - a$ and $y - b$.

Therefore the sloping side is $\sqrt{(x - a)^2 + (y - b)^2}$. As this is the same at each point of the graph, i.e. $\sqrt{c}$ by (6.6.11), this distance is the same for all similar triangles. Constant length of the radius is the geometric definition of a circle.



graph 6.6 e

illustration of a
rectangular hyperbola



Finally we discuss the mapping of an implicit relationship

$$(x - a)(y - b) = c$$

The graph of this relationship is known as the rectangular hyperbola. It has two asymptotic axes, as may be shown by writing the relationship explicitly in terms of either variable.

$$x = a + c/(y - b)$$

For $c \neq 0$, x will not, for any finite value of y, attain the value $x = a$; but x will approach the value $x = a$ asymptotically if y moves towards $+\infty$, or towards $-\infty$, showing that $x = a$ is an asymptotic axis. The similar statement concerning $y = b$ will be obvious.

Typical for the hyperbola is that the graph comes in two branches. We will discuss this feature while assuming $c > 0$, (for $c = 0$, the graph reducesto the cross formed by the two asymptotic axes. The case $c < 0$ exists and is meaningful, but readily deduced from the case $c > 0$ by re-defining one of the two variables, reversing the sign of either x or y).

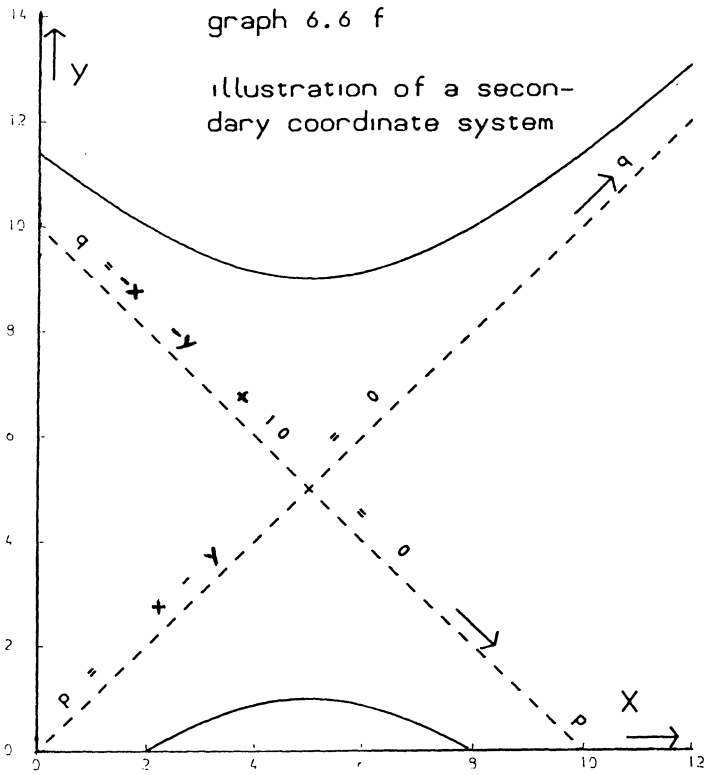
For $x > a$, $y > b$ we consider the positive branch in the top right-hand part of the coordinate space, for $x < a$, $y < b$ the factors $x - a$ and $y - b$ are both negative and their product is positive. The general shape of the graph may be seen in graph 6.6e, which contains the graphical illustration of $(x + 1)(y - 2) = 9$.

A considerable wider group of functions and relationships become manageable in terms of graphical mapping without undue effort if we introduce a secondary coordinate system.

This is normally related to a linear transformation, i.e. we introduce secondary variables p and q, which are linearly related to our originally specified variables x and y by the linear equations.

$$\left.\begin{aligned} p &= a\,x + b\,y + c \\ q &= d\,x + e\,y + f \end{aligned}\right) \qquad (6.6.12)$$

The use of this device is considerably simplified if the secondary coordinates are also associated with a rectangular set of axes and equality of units in both directions i.e. we simply rotate the graph paper, - but we may still need to change the unit of measurement - .



Example

$$(x-y)(10-x-y) = 16$$

We put

$$p = x - y$$
$$q = -x - y + 10$$

The secondary coordinate system now has its origin p = 0, q = 0 at x = 5, y = 5 and the p = 0 axis (q-direction) is the x = y line, the q = 0 axis (p-direction) is the x + y = 10 line.

We have no difficulty in drawing the rectangular hyperbola p.q = 16 in this new coordinate system, see graph 6.6f.

Exercise

Make graphical mappings of:

$x + y = 2$ $\qquad$ $x^2 + y^2 = 16$

$y = x + 2$ $\qquad$ $y = x^2 + x + 1$

$x + 2y = 6$ $\qquad$ $(x-1)^2 + (y-2)^2 = 9$

$x - y = 3$ $\qquad$ $x \cdot y = 16$

$(x-y+2)(x+y-5) = 36$

Use proper graph paper, one sheet for each graph. Then read an arbitrary point from the graph - not already used in drawing it -, and check that it approximately fits the equation which the graph represents.

6.7 Interior points, boundary points, outward points and extreme points

An interior point may be defined loosely as a point inside the space occupied by a set of vectors.

Take for example, the set of two-dimensional vectors satisfying

$$(x - 3)^2 + (y - 4)^2 \leq 4.$$

This is the inside, i.e. the surface of the circle in the previous section. Interior points are points which are actually inside rather than on the ring of the circle itself.

Definition

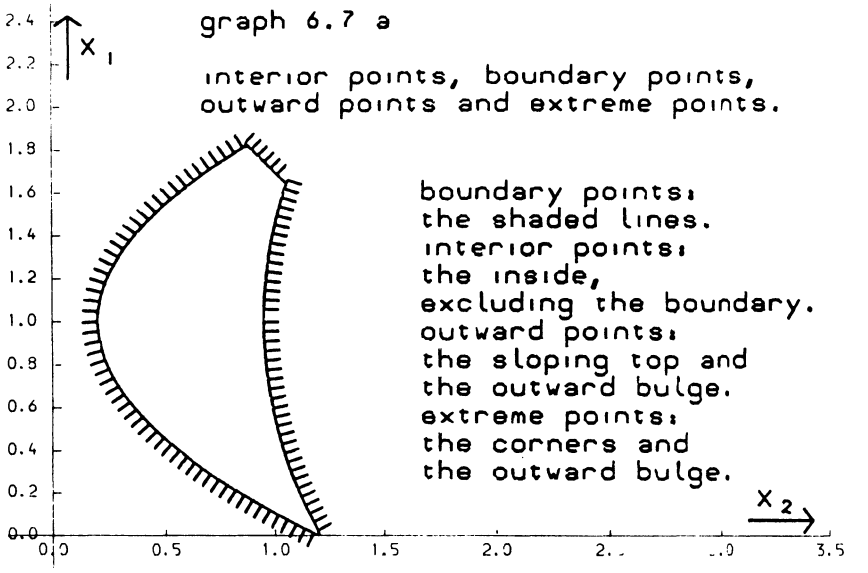
The vector $\underline{x}$ is an interior point of some set of vectors, if for every vector $\underline{z}$ in the same coordinate space, whether inside or outside the set, a combination of $\underline{x}$ and $\underline{z}$

$$\underline{v} = p \cdot \underline{z} + (1 - p)\,\underline{x} \qquad (6.7.1)$$

for some p in the interval

$$0 < p \leq 1 \qquad (6.7.2)$$

belongs to the same set as $\underline{x}$.



The requirement that p should be non-zero is the crucial point in this definition. The vector $\underline{z} - \underline{x}$ denotes an arbitrary direction i.e. towards "any" point in the coordinate space.

Some non-zero part of the line-segment from $\underline{x}$ to $\underline{z}$ must still be in the set. A vector which belong to a set, but is not an interior point of the set is a boundary point.

If a set of vectors is defined by a set of non-trivial restrictions,

$$f_i(\underline{x}) \geq 0 \qquad (6.7.3)$$

$$(i = 1, 2 \ldots m),$$

points for which one or more of these restrictions are binding will normally be boundary points. There are, however, some rather strange restrictions which provide counter-examples. This is because a restriction can be ineffective at certain points, without being trivial throughout the coordinate space.

For example, $x(y - z)^2 \geq 0$ is not a trivial restriction of the kind which we already excluded.

For $x = -1$, $y = 1$, $z = 2$, the restriction $x(y-z)^2 \geq 0$ is not satisfied. But in the $x \geq 0$ half-space this restriction is to all practical purposes trivial. For $x = 1$, $y = 1$, $z = 1$, the inequality $x(y - z)^2 \geq 0$ is exactly satisfied, yet this is an interior point.

To exclude complications of this kind, we must first define their salient characteristic.

A particular restriction

$$f_i(\underline{x}) \geq 0 \tag{6.7.3}$$

is indifferent at the point $\underline{x} = \underline{x}^*$

if all partial derivatives vanish at that point

$$\frac{\partial f_i}{\partial x_j} = 0 \tag{6.7.4}$$

for $j = 1, 2 \ldots . n$.

For linear restrictions these partial derivatives are simply the coefficients a_{ij} which occur in

$$f_i = \sum_{j=1}^{n} a_{ij} x_j + b_i \geq 0$$

It therefore follows that for a linear restriction which is non-trivial the issue of indifference at a particular point does not arise.

Theorem

Let

$$f_i(\underline{x}) \geq 0 \tag{6.7.3}$$

$$i = 1, 2, \ldots . m$$

define a set of vectors (points) in some n-dimensional coordinate space.

And let one restriction in (6.7.3), to be indicated as the h-restriction be exactly fulfilled at some point $\underline{x} = \underline{x}^*$, $f_h(x)$ not being indifferent at that point.

$$f_h(\underline{x}^*) = 0 \tag{6.7.5}$$

$$\frac{\partial f_h}{\partial x_j} \neq 0 \text{ for } \underline{x} = \underline{x}^* \quad \text{(some j)} \tag{6.7.6}$$

Then

$\underline{x} = \underline{x}^*$ is a boundary point of the set of vectors satisfying (6.7.3)

Proof

Consider a first-order linear approximation of

$f_h(\underline{x})$ at the point $\underline{x} = \underline{x}^*$

$$f_h(\underline{x}^* + d\underline{x}) = f_h(\underline{x}^*) + \sum_{j=1}^{n} \frac{\partial f_h}{\partial x_j} dx_j \qquad (6.7.7)$$

We may take minus the vector of first-order derivatives as the vector $\underline{z}$ in 6.7.1 and a very small increment indicated as dp as the proportionality factor p in (6.7.1).

Denoting the vector of partial differentials as $\underline{d}$, (6.7.7) is now evaluated for that particular direction as

$$f_h(\underline{x}^* + \underline{d}\ dp) = f_h(\underline{x}^*) - dp\ \underline{d}'\ \underline{d} \qquad (6.7.8)$$

The first term on the left-hand side of (6.7.8) is by assumption i.e. (6.7.5) zero, the second is negative, this contradicts (6.7.3) for the h-restriction.

Since an interior point is defined as staying inside the set when starting to move away from it, $\underline{x} = \underline{x}^*$ cannot be an interior point if a restriction is binding without indifference. q.e.d.

The reverse is also true and even more obvious:

If a set of vectors is defined by some inequalities, and at some point $\underline{x}$, $\underline{x}^*$ none of these inequalities is binding, then that point is an interior point.

The terms "boundary point" and "one or more restrictions binding" are therefore in practice interchangeable.

We will call $\underline{x} = \underline{x}^*$ an outward point of some set of vectors, if there is a non-zero direction-vector $\underline{a}' \neq 0$, for which

$$\underline{a}'\ \underline{x} \leq \underline{a}'\ \underline{x}^* \qquad (6.7.9)$$

is true for all $\underline{x}$ in the set.

The requirement that $\underline{a}'$ must not be a zero vector serves merely to exclude triviality. For $\underline{a}' = 0$ all vectors $\underline{x}$ would satisfy (6.7.9) for all sets of vectors and for all points $\underline{x}^*$ in them.

Outward points are also boundary points. This is because the direction-vector $\underline{z} - \underline{x}$ in (6.7.1) can be taken in any direction. Taking the column-prsentation of $\underline{a}'$ as the direction $\underline{z} - \underline{x}$ in (6.7.1), we may re-write (6.7.1) as

$$\underline{v} = \underline{x}^* + p\ \underline{a} \tag{6.7.10}$$

where the point towards which one is moving is found by comparing (6.7.10) and (6.7.1)

$$\underline{z} = \underline{x}^* + \underline{a} \tag{6.7.11}$$

Evaluation of (6.7.9) for the direction indicated by (6.7.10) yields

$$\underline{a}'\ \underline{x} = \underline{a}'\ \underline{x}^* + p\ \underline{a}'\ \underline{a} \geq \underline{a}'\ \underline{x}^* \tag{6.7.12}$$

thus contradicting (6.7.9)

The definition of an interior point is that we can move away from it over some distance indicated by the parameter p, and that includes moving in a particular direction indicated by the vector $\underline{a}'$ itself.

The reverse is however, not always true: boundary points are not always outward points (This is so, only in convex sets)

Since outward points are boundary points they are, by previous theorem, also points at which at least one restriction is binding.

CHAPTER VII

SOME BASIC LINEAR PROGRAMMING CONCEPTS

7.1 The linear programming problem

Linear programming consists of:

(a) Finding a solution to a system of linear inequalities, and
(b) Selecting from all solutions to such a system, a solution for which a certain linear function of the variables attains a maximum value.

A solution, satisfying (a), but not necessarily (b), is indicated as a feasible solution. An arbitrary vector of the required order which fails to satisfy the inequalities is named, somewhat euphemistically, a non-feasible solution. A solution, satisfying both (a) and (b), is named a feasible and optimal solution. The function to be maximised is the objective function or preference function.

There is of course a mirror problem to the one formulated above. One could search for a solution minimizing a certain linear function of the variables. It would lead to a second algorithm, differing only in sign from the one I want to discuss. Any minimization problem can be formulated in terms of maximizing, by inverting the sign of the objective function. We will only deal with maximization problems.

I will in fact impose one further restriction.* All inequalities should be in the form of a linear function of the variables being smaller than, or equal to a constant. Again inversion of the sign will turn the "larger than" type of inequality into the "smaller than" type.

A more general convention is that variables shall be restricted to non-negative values only. Historically, linear programming dealt with physical activities, and this convention was self-evident. Adherence to this convention also helps to simplify certain features of the Simplex Algorithm. In a later stage we will also discuss unconstrained activities. A variable,

---

* Where it concerns the signs of the coefficients, I follow the conventions adhered to by Charnes, Cooper and Henderson [5]

an activity is free, or unconstrained, or (as it will be referred to in code-listings in this book) of type absolute, if it is exempted from the convention of non-negativity.

7.2 The L.P - problem in matrix notation

Let A be a m by n matrix of known coefficients. Let $\underline{x}$ be an n by 1 column vector of unknown variables. Let $\underline{b}$ be a m by 1 column vector of known constants. Let $\underline{w}'$ be a 1 by n row vector of known coefficients. Let τ be a scalar variable. The L.P.-problem can then be formulated in matrix notation, as follows:

maximise $\tau = \underline{w}' \underline{x}$, (7.2.1)

subject to $A \underline{x} \leq \underline{b}$ (7.2.2)

$x_j \geq 0 \quad (j = 1, 2 \ldots . n)$

We now define an m by 1 column vector of slack-variables, named $\underline{s}$. The problem can then be formulated as an undetermined equations system:

maximise τ in:

$$\begin{aligned} A \underline{x} + \underline{s} &= \underline{b} \\ -\underline{w}' \underline{x} + \tau &= 0 \end{aligned}$$

$x_j \geq 0 \quad (j = 1, \ldots n)$

$s_i \geq 0 \quad (i = 1, \ldots m)$

The non-negativity restriction applies to the elements of the vectors, $\underline{x}$ and $\underline{s}$,. It does not apply to the value of the objective function. We will meet examples where the optimal value of τ is negative.

7.3 A numerical example

Consider a factory with 2 different machines. The capacities of these machines are known as 200 and 100. There are 3 production processes (products). The operation of 1 unit of these processes gives the manufacturer a value added of 2, 5 and 1 unit of money respectively. There is a competitive market, in which any output that is produced can be sold at the current price. The prices cannot be significantly influenced by the firm's small share in the market.

The objective function will then be:

$$\tau = 2 x_1 + 5 x_2 + x_3,$$

where x_1, x_2 and x_3 are the 3 production processes. The standard times for the different processes of both machines are known. They give rise to 2 inequalities plus the non-negativity requirements.

$$x_1 + x_2 + x_3 \leq 200$$

$$x_1 + 3x_2 < 100$$

$$(x_1, x_2, x_3 \geq 0)$$

This little problem will serve us in doing some exercises. It also is an example of an important class of practical applications of linear programming. It has one important simplifying feature. A feasible solution is already known. Just doing nothing, $x_1 = x_2 = x_3 = 0$, will satisfy the restrictions. It is of course not an optimal solution.

7.4 Graphical solution of a problem with 2 variables

When there are only 2 variables, an L.P.-problem can be solved by graphical methods. The practical relevance of this fact is not very great. Any realistic problem will involve more variables. And the Simplex Algorithm can be applied generally, with reasonable efficiency. But the exercise will serve us, by illustrating some basic concepts of linear programming.

Consider the problem:

minimise: $3x_1 - x_2$,

subject to:

$$x_1 + x_2 \leq 7$$

$$3x_1 + x_2 \geq 6$$

$$-x_1 + x_2 \leq 2$$

$$(x_1, x_2 \geq 0)$$

This problem, in its present form, does not satisfy our conventions of uniform notation. Therefore we re-state the problem, in a form which satisfies the conventions:

maximise: $\tau = -3x_1 + x_2$,

subject to:

$$x_1 + x_2 \leq 7$$

$$-3x_1 - x_2 \leq -6$$

$$-x_1 + x_2 \leq 2$$

$$(x_1, x_2 \geq 0)$$

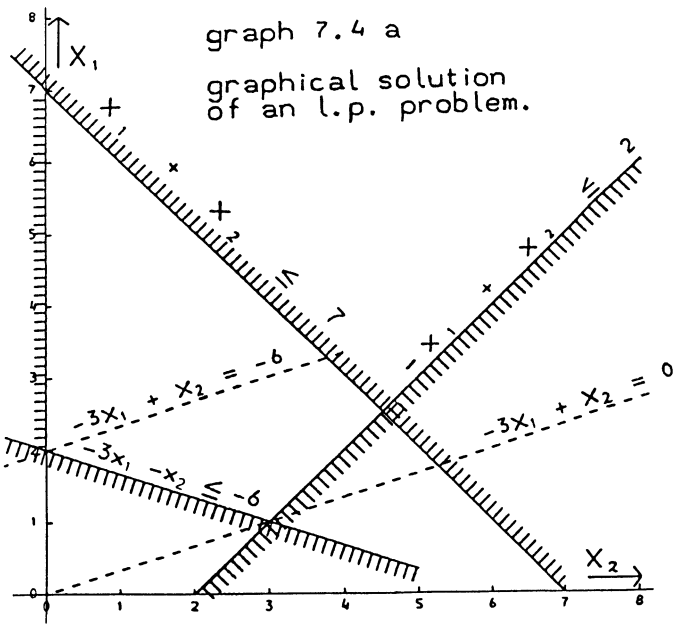
Alternatively, the problem can be formulated as an under-determined equations-system:

maximize τ in:

$$\left.\begin{array}{rrrrrrrrrl} x_1 & + & x_2 & + s_1 & & & & = & 7 & \\ -3x_1 & - & x_2 & & s_2 & & & = & -6 & \\ -x_1 & + & x_2 & & & + s_3 & & = & 2 & \\ 3x_1 & - & x_2 & & & & + \tau & = & 0 & \end{array}\right)$$

$$(x_1, x_2, s_1, s_2, s_3 \geq 0)$$

Because there are only two variables, we are able to map the restrictions in the two-dimensional x_1, x_2 co-ordinate plane, and solve the problem by visual inspection of the mapping. (See graph 7.4a below.)



The restrictions have been shaded on the non-admissible side, as far as they appeared to be relevant. A restriction which is satisfied by any solution satisfying the other restrictions in the system, is named a redundant restriction. In the present example, one of the tacit restrictions ($x_1 \geq 0$), is redundant. The 4 points (2,0), (7,0), ($2\frac{1}{2}$,$4\frac{1}{2}$) and (1,3) are the corners of a quadrangle. The area inside this quadrangle is the feasible area or feasible region. Note that the origin of the co-ordinate plane (the point $x_1=0$, $x_2=0$) is outside the feasible area. This could be seen from the tabular presentation of the inequalities. Once the restrictions are written in the standardized form, all the variables on the left-hand side of the $\leq$ sign and a constant on the right-hand side, the negative value of the constant in the second restriction shows that this inequality is not satisfied by $x_1 = x_2 = 0$. Apart from the restrictions, the graph shows 2 dashed lines for constant values of the objective function:

$$\tau = -3x_1 + x_2 = -6$$

and

$$\tau = -3x_1 + x_2 = 0$$

These two lines are parallel, as they should be. Any other line for another constant value of the objective function would run parallel to the ones in the graph. Clearly, no line for a positive non-zero value of τ goes through the feasible area. This pins $\tau = 0$ as the maximum, and $x_1 = 1$, $x_2 = 3$ as the optimal solution.

In the optimum, 2 restrictions are exactly fulfilled. Apart from the tacit restrictions, there is one restriction ($x_1 + x_2 \leq 7$) amply fulfilled.

The corresponding slack should then be non-zero:

$$s_1 = 7 - x_1 - x_3 = 3.$$

CHAPTER VIII

OUTLINE OF THE SIMPLEX ALGORITHM

8.1 The concept of a basic solution

The Simplex Method solves the Linear Programming problem by solving a series of linear equations systems. The initial system of inequalities is written as an equations-system, by adding slack variables. (See Section 7.2). The numerical example in 1.3 is then written as follows:

$$\begin{aligned} x_1 + x_2 + x_3 + s_1 &= 200 \\ x_1 + 3x_2 + s_2 &= 100 \\ -2x_1 - 5x_2 - x_3 + \tau &= 0 \end{aligned}$$

One equation should always be read as expressing one variable, always with a unity coefficient. That one variable is then expressed as a constant, plus a linear combination of the other variables. In the example, just re-written above, the first equation expresses s_1 as being 200 minus $x_1 + x_2 + x_3$. The understanding is, that in this initial solution x_1, x_2 and x_3 are in fact zero. That interpretation of a system of equations written in this particular form,* allows us to read a current solution-value from a tabulation of the system of equations, without further calculation. In the example, s_1 has the value 200. The coefficients of the other variable indicate how s_1 would change if any of the other variables would be given a non-zero value. This convention allows in each equation, a non-zero coefficient for only one non-zero variable. Conversely, a non-zero variable should have a unity coefficient in one equation, and zero coefficients in all other equations.

Or, more strictly speaking, a variable that is solved for, by the equations-system in question, is represented with a unity coefficient in one equation, and zero coefficients in the other equations. In the above example, s_1, s_2 and τ are the variables that are solved for. We are of course not assuming that x_1, x_2 and x_3 should necessarily be zero. Only our first current solution, considers non-zero values for s_1, s_2 and τ only.

---

* That form of presentation of a system of equations with a single unity-coefficient in each equation, is often called the canonical form. See W. Garvin [11], p.28.

The variables which are solved in the current solution are named the basic variables of that solution.

One variable is always solved for, but not normally listed as a basic variable: the value of the objective function.

The inference from this way of reading the equations-system is the following: The number of basic variables is always equal to the number of restrictions. During later iterations, other variables will be solved for, will become basic variables. But always when one variable enters the basis, another one is leaving the basis. Here the word "basis" is understood as meaning the list of names of the basic variables. This list of names determines the corresponding solution completely; it allows for solving their values by way of an ordinary linear equations-system. Obviously, a list of m non-zero variables, can be a list of basic variables, if and only if, the corresponding equations-system is non-singular.

Basic solutions are only a restricted class among all possible solutions. In a two-dimensional example (Section 7.4), this can be seen at a glance. The basic solutions of the problem, as far as they are feasible, correspond to the corners of the feasible area. Yet the Simplex Algorithm analyses no other but basic solutions. When the algorithm has been surveyed, it will become apparent that this restriction is justified in the general n-dimensional case as well.

Consider the system (7.2.1). We now partition this system as follows:

$$\begin{aligned} A_{11}\,\underline{x}_1 + A_{12}\,\underline{x}_2 &= \underline{s}_1 && && \underline{b}_1 \\ A_{21}\,\underline{x}_1 + A_{22}\,\underline{x}_2 &\quad + \underline{s}_2 && && \underline{b}_2 \\ -\underline{w}'_1\,\underline{x}_1 - \underline{w}'_2\,\underline{x}_2 &\quad + \tau = 0 \end{aligned} \qquad (8.1.1)$$

The equations and the variables are supposed to be re-arranged in such a way that the block-row $A'_1 = [A_{11}, A_{12}]$ now corresponds to the exactly fulfilled restrictions. The block-column consisting of A_{11} and A_{21} then refers to those of the significant variables that are basic in the current solution.

The composite vector consisting of $\underline{x}_1$ and $\underline{s}_2$ is the (column) vector of the basic variables.

The values of the basic variables are found by solving the linear equations-system:

$$\left.\begin{aligned} A_{11}\,\underline{x}_1 &= \underline{b}_1 \\ A_{21}\,\underline{x}_1 + \underline{s}_2 &= \underline{b}_2 \end{aligned}\right) \qquad (8.2.2)$$

This system is determinate, if and only if the composite ma-rix

$$\begin{bmatrix} A_{11} & \\ A_{21} & I \end{bmatrix}$$

is non-singular. This matrix is then often named the basis-matrix. If both groups of variables, elements of $\underline{x}$, and remaining slack-variables are present, the basis-matrix is a block-triangular matrix. One of its diagonal blocks is a unit matrix, and as such always square and non-singular. We saw in Section 5.7 that the determinant of a block-triangular matrix is the product of the determinants of its diagonal blocks. Then if the basis is to be non-singular, it is necessary and sufficient, that A_{11} is square and non-singular.

A_{11} is the pivot-matrix or block-pivot*

8.2 The Simplex Tableau

A Simplex Tableau represents an equation system, written in the standard "canonical" form, discussed in the preceding paragraph. Only, we do not write the names of the variable every time again.

The equations-system of Section 8.1 is represented by the following Simplex Tableau

TABLEAU 8.2 (EXAMPLE 7.2)
A SIMPLEX TABLEAU. (SET-UP TABLEAU, COMPLETE FORM)

NAME	!!	X 1	X 2	X 3	!	S 1	S 2	!	T	!!	VALUE
S 1	!!	1	1	1	!	1	-	!	-	!!	200
S 2	!!	1	3	-	!	-	1	!	-	!!	100
T	!!	-2	-5	-1	!	-	-	!	1	!!	-

*See also: A.R.G. Heesterman, Special Simplex Algorithm for Multi-Sector Problems [20], as well as Section 3.11 of this book.

On account of computer-checking of all Simplex tableaux in the manuscript the value of the objective function is referred to in tableaux as t, even where otherwise the use of a Greek letter would be more conventional.

The figures in the tableau are identical to those in Section 8.1. The "τ" column will remain a unit vector during all iterations.

8.3 Choosing the pivot column

If we write the objective function explicitly, it becomes again: $\tau = 0 + 3\ x_1 + 5\ x_2 + x_3$. We are looking for a new variable, with a view to increasing the value of the objective function. We should then look for a positive coefficient in the explicitly written objective function. This corresponds to a negative figure in the implicitly written function

$$0 = -3\ x_1 - 5\ x_2 - x_3 + \tau .$$

We will want to increase the value of the objective function as much as possible. Hence we normally take the most negative element in the objective function.

This rule is known as the principle of steepest ascent. It may happen that two or more figures are equally qualified as the most negative, because they are equal. We will assume the name with the lowest index to be taken in such cases. This is a device to make the choice unambiguous.

If we only want to reach the optimum, no matter after how many steps, the principle of steepest ascent can be dispensed with. Any negative element in the current preference function will do. The principle of steepest ascent is a guide to an efficient choice of a new basic variable. As such it is a somewhat arbitrary criterion, and there are other rules for selecting the incoming variable.

Once we have chosen a new basic variable, we have chosen a pivot column. This is the column in the tableau, corresponding to the new basic variable.

In our example, the rule of the steepest ascent gives us x_2 as the new basic variable. The pivot column will be marked in the tableau with an asterisk.

Also, following the suggestion in Section 3.5 we carry the sum-count of all the entries in each row as a "check" column, and perform the same operations on them.

TABLEAU 8.3 (EXAMPLE 7.2)
SIMPLEX TABLEAU, WITH MARKING OF THE PIVOTAL COLUMN.

NAME	X 1	X 2 *	X 3	S 1	S 2	T	VALUE	SUM CH
S 1	1	1	1	1	-	-	200	204
S 2	1	3	-	-	1	-	100	105
T	-2	-5	-1	-	-	1	-	-7

8.4 The choice of a pivot row

The pivot column will tell us how the (old) basic variables will be reduced, by introducing a non-zero value for the new basic variable, at a level of 1 unit. For instance, the s_2 - row should be read as $100 = x_1 + 3\,x_2 + s_2$. We now write s_2 explicitly, and suppress the coefficients of the other non-basic variables, which will remain zero-valued.

$$s_2 = 100 - 3\,x_2$$

If s_2 is to be the variable leaving the basis, it should be reduced to zero. That solves x_2 as

$$x_2 = 100 : 3 = 33\ 1/3$$

Considering s_1 as a candidate for leaving the basis, we would find x_2:

$$x_2 = 200 : 1 = 200.$$

If we were to increase x_2 to more than 33 1/3, s_2 would become negative. In particular for $x_2 = 200$ (or s_1 leaving the basis), we would have $s_2 = 100 - 3\,x_2 = 100 - 600 = -500$. This is no longer an admissable solution. We should not increase x_2 to more than 33 1/3.

This gives us the rule of the smallest quotient: The name of the variable to be eliminated out of the basis is found by dividing all (positive) values of the basic variables through their corresponding positive elements in the pivot column, and by selecting the smallest from these quotients.

In its present form, this rule is valid only when the current basic solution is feasible already. The question of how to attain a feasible solution, if the trivial solution is not feasible, will be discussed in Chapter IX.

We will indicate our choice in the tableau. The tableau will

now become:

TABLEAU 8.4 A (EXAMPLE 7.2)
SIMPLEX TABLEAU, WITH RING-MARKING OF THE PIVOT.

NAME	X 1	X 2 *	X 3	S 1	S 2	T	VALUE	SUM CH
S 1	1	1	1	1	-	-	200	204
S 2	1	③	-	-	1	-	100	105
T	-2	-5	-1	-	-	1	-	-7

The row of the tableau corresponding to the variable that is to be eliminated from the basis is named the pivot-row. The circle marks the intersection between the pivot-row and the pivot-column. The encircled element is named the pivot. Actually, we could have dispensed with a separate indication of the pivot-column. The indication of the pivot (with a circle), designates both pivot-row and pivot-column.

The question arises, if a search for the smallest quotient is always successful. Could one not meet cases where there are no positive entries in the pivotal column at all?

We could.

Example

Maximise x_1

subject to $x_1 \leq x_2 + 1 \quad (x_1, x_2 \geq 0)$.

We write the Simplex Tableau, and make one step.

TABLEAU 8.4 B (EXAMPLE 8.4)

AN UNBOUNDED PROBLEM DEVELOPS AFTER ONE STEP.
(TWO SIMPLEX TABLEAUX IN ONE REFERENCE TABLEAU)

NAME	X 1	X 2	S 1	T	VALUE
S 1	①	-1	1	-	1
T	-1	0	0	1	-

THE SECOND TABLEAU IS UNBOUNDED (IN X 2)

NAME	X 1	X 2 *	S 1	T	VALUE
X 1	1	-1	1	-	1
T	-	-1	1	1	1

The rule of the steepest ascent indicates x_2 as the next pivot-column, but the x_2 column does not provide a pivot row.

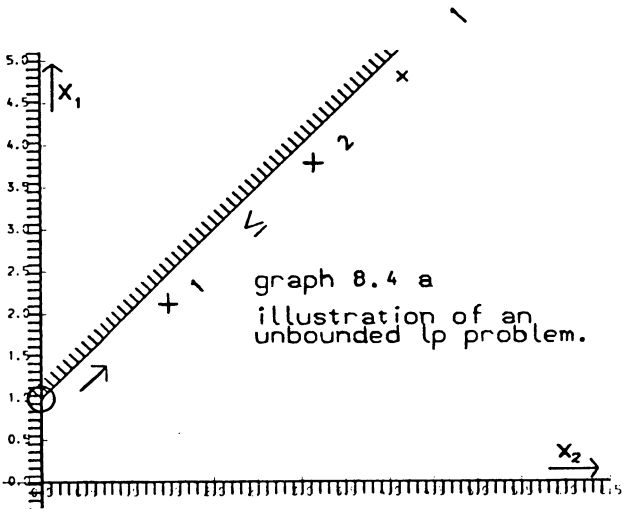
The reason is that this is an unbounded problem and the x_2 column is the unbounded* column.

Variable x_1, and as a consequence the objective function can attain any value above 1, and a matching value for x_2 can be found. x_1 = 1000000000000 with x_2 = 1000000000001 is a feasible solution, and there is no finite maximum. The general case is that unbounded columns increase the value of the objective function and the value of all basic variables (as far as they change at all). Hence they do not reduce any variable to zero and no finte upper limit for such variables exists. Since they increase the value of the objective function, no finite upper limit for the objective function exists either.

---

* Sometimes one will meet the term "unbounded in the preference direction". In the next chapter, we will consider LP problems where it is sometimes necessary to enter variables which will reduce the value of basic variables in order to find a feasible solution. However, a well-designed algorithm should ensure that such columns are entered, only if one can be sure that they will provide a pivot row. No such guarantee is possible with respect to columns which are unbounded in the preference direction because the LP problem itself may be unbounded.

In the case at hand, this is confirmed by a graphical mapping of the one restriction, in graph 8.4a below.



graph 8.4 a
illustration of an unbounded lp problem.

The one step from the origin to the point $x_1 = 1$, $x_2 = 0$ (marked 0) has been made, but further increase in x_1 can be obtained at infinitum, by moving along the binding restriction, in the direction of the arrow. Linear programming algorithms need a special "abnormal" exit-loop, to signal unbounded problems.

The LP problem is held to have no feasible and optimal i.e. finite maximum solution if the search for a pivot row fails.

8.5 The Simplex Step

Once a new list of (names of) basic variables is known, we will want to rewrite the equations-system, in order to have it again in its standard "canonical" form, with respect to the new basis. This is achieved by a procedure which is very similar to the one discussed in Section 3.3 for a block-equation.

First, the pivot-row is divided by the pivot. In our example, the s_2-row should then be divided by 3. Instead of

$$x_1 + 3\,x_2 + s_2 = 100$$

we will now write:

$$1/3\,x_1 + x_2 + 1/3\,s_2 = 33\ 1/3$$

In fact we only write the coefficients, the numbers

1/3, 1, 1/3, and 33 1/3

instead of

1, 3, 1, and 100.

The new row will now express x_2 as a constant (33 1/3), plus a linear function of the non-basic variables x_1 and s_2.

$$x_2 = -1/3\ x_1 - 1/3\ s_2 + 33\ 1/3$$

In all other equations, this last expression is now substituted for x_2. Or, more generally, the equation that is obtained after dividing the pivot-row by the pivot, is used to eliminate the new basic variable from all other equations. Arithmetically, this is accomplished by subtracting the pivot row so many times from all the other rows, as to cause vanishment of the coefficients in the pivot column.

The operation can also be expressed in terms of matrices and vectors. The "updating formula" is given here only for the "rest" of the tableau, i.e. excluding the pivot-row and the pivot-column itself.

$$T_{n+1} = T_n - \underline{k}\ \pi^{-1}\ \underline{r}' \qquad (8.5.1)$$

Here, T_n is the old tableau, T_{n+1} is the new one. Both matrices exclude the pivot-row. $\underline{k}$ is the pivot column (as found in the old tableau), $\underline{r}'$ is the pivot-row (as found in the old tableau), and π is the pivot. Obviously then $\pi^{-1}\underline{r}'$ is the new row, is obtained by dividing the pivot row by the pivot. The outward product of the old pivot-column and the new pivot-row, is subtracted from the remainder of the tableau. The pivot row itself is treated separately, it is divided by the pivot.

After the updating operation, our second Simplex Tableau will be:

TABLEAU 8.5 (EXAMPLE 7.2)
SIMPLEX TABLEAU, AFTER FIRST COMPLETE STEP.

NAME	X 1	X 2	X 3	S 1	S 2	T	VALUE	SUM CH
S 1	0.67	-	1	1	-0.33	-	166.67	169
X 2	0.33	1	-	-	0.33	-	33.33	35
T	-0.33	-	-1	-	1.67	1	166.67	168

The s_2-row has been divided by 3. The new transformed s_2-row (now named x_2-row), has been subtracted with a factor 1, from the s_1-row, and with a factor -5 from the objective function. (Or, alternatively, the transformed s_2-row) has been added with a factor 5 to the objective function.

Note the close analogy between the pivoting operation in linear programming and the elimination process in a block-equation, discussed in Chapters I and III.

Following section 1.1, the updating operation for the τ row may be made explicit as follows:

Multiply the new x_2-equation by 5, to obtain:

$$1\ 2/3\ x_1 + 5\ x_2 + 1\ 2/3\ s_2 = 166\ 2/3$$

The old τ-equation, to be added to it, is:

$$-2\ x_1 - 5\ x_2 - x_3 + \tau = 0$$

to obtain the new τ-equation

$$-1/3\ x_1 - x_3 + 1\ 2/3\ s_2 + \tau = 166\ 2/3$$

8.6 The attainment of the optimum

The "normal" end of the Simplex Algorithm occurs when a search for a new pivot column fails, because no negative elements occur in the current objective function. In our example, this will be the case after the second step.

After our first step, we will find -1 for x_3 to be the most negative element in the current preference function. In the x_3-column, there is only one non-zero positive element. Again, we will indicate our pivot-choice in the tableau with a circle.

TABLEAU 8.6 (EXAMPLE 7.2)
SIMPLEX TABLEAU, AFTER FIRST COMPLETE STEP, WITH NEW PIVOT MARKED.

NAME	X 1	X 2	X 3	S 1	S 2	T	VALUE	SUM CH
S 1	0.67	-	(1)	1	-0.33	-	166.67	169
X 2	0.33	1	-	-	0.33	-	33.33	35
T	-0.33	-	-1	-	1.67	1	166.67	168

CONTINUATION TABLEAU 8.6
THE NEXT TABLEAU IS THE OPTIMUM.

NAME	X 1	X 2	X 3	S 1	S 2	T	VALUE	SUM CH
X 3	0.67	-	1	1	-0.33	-	166.67	169
X 2	0.33	1	-	-	0.33	-	33.33	35
T	0.33	-	-	1	1.33	1	333.33	337

We now write the objective function. From the tableau we read:

$$1/3\ x_1 + s_1 + 1\ 1/3\ s_2 + \tau = 333\ 1/3$$

The equivalent explicit form of the objective function is:

$$\tau = 333\ 1/3 - 1/3\ x_1 - s_1 - 1\ 1/3\ s_2$$

In the tableau, there were only positive coefficients in the transformed objective function. As a result of this, all the coefficients of the variables in the explicit form of the objective function, are negative. All the non-basic variables would, if introduced into the basis, reduce the value of the objective function.

We conclude that we have reached the optimum.

The figures 1/3 for x_1, 1 for s_1, and 1 1/3 for s_2, occurring in the row "τ" in the tableau, are known as the shadowprices of these variables. In this case, they are the optimal shadowprices. The row may also be named the transformed objective function. The shadowprices are the elements of the transformed objective function. The non-negativity of the transformed objective function (of all the shadowprices), certifies the fact that an optimum has been reached. If all the shadowprices are positive non-zero, that is also the unique optimum, if there are zeros, several vectors may be co-equally optimal.

Exercise 8.6a

Write the implicit and the explicit forms of the x_3 and the x_2 equations, as corresponding to the optimum tableau 8.6.

Exercise 8.6b

The following LP problem is given

Maximise $\tau = 2\ x_1 - x_2$

Subject to $2\ x_1 + x_2 \leq 15$

$2\ x_2 \geq x_1 - 3$

$(x_1, x_2 \geq 0)$

Make a graphical mapping of this problem and identify the optimal and feasible solution by means of graphical analysis. Then solve the problem by the Simplex Method, while also marking each intermediate solution-point in the graph. (Answer-sheet at the end of the chapter).

8.7 The Simplex Tableau in matrix notation

Consider the system (8.1.1) in Section 1 of this chapter. A_{11} is assumed to be square and non-singular. We can then pre-multiply the first block-row of (8.1.1) by A_{11}^{-1}. The result is:

$$\underline{x}_1 + A_{11}^{-1} A_{12}\underline{x}_2 + A_{11}^{-1} \underline{s}_1 = A_{11}^{-1} \underline{b}_1 \qquad (8.7.1)$$

Or, solving for $\underline{x}_1$:

$$\underline{x}_1 = -A_{11}^{-1} A_{12}\underline{x}_2 - A_{11}^{-1} \underline{s}_1 + A_{11}^{-1} \underline{b}_1 \qquad (8.7.2)$$

With the help of this last expression we can eliminate $\underline{x}_1$ out of the other block-rows of (8.1.1). Together with the transformed first block-row, we obtain

$$\left.\begin{array}{l} \underline{x}_1 + A_{11}^{-1}A_{12}\underline{x}_2 + A_{11}^{-1}\underline{s}_1 = A_{11}^{-1}\underline{b}_1 \\ \left[A_{22} - A_{21}A_{11}^{-1}A_{12}\right]\underline{x}_2 - A_{21}A_{11}^{-1}\underline{s}_1 + \underline{s}_2 = \underline{b}_2 - A_{21}A_{11}^{-1}\underline{b}_1 \\ \left[-\underline{w}'_2 + \underline{w}'_1 A_{11}^{-1}A_{12}\right]\underline{x}_2 + \underline{w}'_1 A_{11}^{-1}\underline{s}_1 + \tau = \underline{w}'_1 A_{11}^{-1}\underline{b}_1 \end{array}\right\} \qquad (8.7.3)$$

The system (8.7.3) is the Simplex Tableau in matrix notation. This emerges when we write the names of the (vectors of) variables on top of the block-columns, instead of every time after the matrices. We did the same with the equations-system in figures.

name	$\underline{x}_1$	$\underline{x}_2$	$\underline{s}_1$	$\underline{s}_2$	τ	value
$\underline{x}_1$	I	$A_{11}^{-1}A_{12}$	A_{11}^{-1}			$A_{11}^{-1}\underline{b}_1$
$\underline{s}_2$		$A_{22}-A_{21}A_{11}^{-1}A_{12}$	$-A_{21}A_{11}^{-1}$	I		$\underline{b}_2-A_{21}A_{11}^{-1}\underline{b}_1$
τ		$-\underline{w}'_2+\underline{w}'_1A_{11}^{-1}A_{12}$	$\underline{w}'_1A_{11}^{-1}$		1	$\underline{w}'_1A_{11}^{-1}\underline{b}_1$

The corresponding initial tableau can be expressed in matrix notation as well:

name	$\underline{x}_1$	$\underline{x}_2$	$\underline{s}_1$	$\underline{s}_2$	τ	value
$\underline{s}_1$	A_{11}	A_{12}	I			$\underline{b}_1$
$\underline{s}_2$	A_{21}	A_{22}		I		$\underline{b}_2$
τ	$-\underline{w}_1'$	$-\underline{w}_2'$			1	0

This tableau gives the initial numerical information, and no other. Yet is refers to the current and in general non-trivial solution. This is so because rows and columns have been re-arranged to the partitioning by (8.1.1).

The block-columns "$\underline{x}_1$" and "$\underline{s}_2$" can be put together. They form the basis-matrix. We will normally indicate this matrix as B.

$$B = \begin{bmatrix} A_{11} & \\ A_{21} & I \end{bmatrix} \qquad (8.7.4)$$

The basis-matrix consists of the columns corresponding to the basic variables, and of the rows corresponding to all the restrictions. It is not normal practice to consider the objective function as one of the restrictions. However, the arithmetics of the Simplex method are the same for the objective function and for a non-binding restriction. The one difference is that the value of the objective function is always solved for. Even if it should become zero or negative, the objective function row never becomes pivot row. Otherwise one can consider the preference function as part of the basis, as the $m+1^{th}$ restriction. The value of the objective function should then be treated as one of the basic variables, except for search operations.

It will be observed that the inverse of the basis is to be found in any current Simplex Tableau, since

$$\begin{bmatrix} A_{11} & \\ \hline A_{21} & I \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} & \\ \hline -A_{21}A_{11}^{-1} & I \end{bmatrix}$$

8.8 The shortened Simplex Tableau

To write a set of unit vectors for the basic variables at each step, in each Simplex Tableau, is a bit tiresome. One could dispense with that. This is done in the shortened Simplex Tableau. When a column becomes pivot-column, we do not write a unit vector in its place. The empty place in the tableau is taken by the column associated with the just-eliminated variable.

The 3 tableaux of our numerical example can now be written as follows:

TABLEAU 8.8 (EXAMPLE 7.2)

TABULATION OF ALL THREE SHORTENED TABLEAUX.

SET-UP TABLEAU, WITH PIVOT MARKED.

NAME	X 1	X 2	X 3	VALUE	SUM +1
S 1	1	1	1	200	204
S 2	1	③	-	100	105
T	-2	-5	-1	-	-7

SECOND TABLEAU, WITH NEW PIVOT MARKED.

NAME	X 1	S 2	X 3	VALUE	SUM +1
S 1	0.67	-0.33	①	166.67	169
X 2	0.33	0.33	-	33.33	35
T	-0.33	1.67	-1	166.67	168

OPTIMAL TABLEAU.

NAME	X 1	S 2	S 1	VALUE	SUM +1
X 3	0.67	-0.33	1	166.67	169
X 2	0.33	0.33	-	33.33	35
T	0.33	1.33	1	333.33	337

The figures in these shortened tableaux are the same as in the full tableaux on the same example in Sections 8.2-8.5. Only the unit vectors have been suppressed. The corresponding equations-systems are of course the ones according to the full tableau.

The shortened tableau gives rise to a slightly different procedure for updating. There now is a special treatment of the pivot column. The new column can be obtained as the old pivot-column, divided by minus the pivot, and the pivotal element itself is replaced by its reciprocal. With the full tableau, the same result would have been obtained without this special rule; the unit vector would have served as an operator to obtain the result.

Note that this new rule for updating the pivot column is almost identical to the rule for updating the pivot row. The inversion of the sign i.e. division by minus the pivot is the only difference. The interpretation of a shortened tableau the same as for the corresponding explicit one, the tableau corresponds to the equations-system given by the corresponding "full" tableau. Note, that the "check" column now contains the sumcount of the corresponding row, increased by one.

The suppressed unit-vector has to be taken into account when one computes or verifies the "check" column.

The sumcount-property, as discussed in Sections 3.5 and 8.3 holds because a multiple of the pivot-row is added to other rows. This is only so in the full tableau, not in the shortened tableau.

8.9 The Rule of the highest step

We now consider a rule for choosing the pivot column, which is an alternative to the rule of the steepest ascent.

In section 8.3 it was assumed that we choose as a new basic variable, the one which gives the maximum increase in preference value, per unit of the new basic variable.

In general, the value of the new basic variable will not be just one unit. We may anticipate the choice of the pivot row and tentatively calculate for each column, the increase in the value of the objective function to be obtained from entering that particular column as new basic variable.

We may illustrate this new rule, for which I propose the name given in the title of this section, with the example from sections 8.1 and 8.2.

The shortened tableau for this problem is

TABLEAU 8.9 (EXAMPLE 7.2)

THE RULE OF THE HIGHEST STEP
FAVOURS LONG AS WELL AS STEEP STEPS.

NAME	X 1	X 2	X 3	VALUE
S 1	1	1	1	200
S 2	①	3	-	100
T	-2	-5	-1	-

If x_1 is chosen as pivot column, then s_2 will be the pivot-row, with an increase in the value of the objective function of 200.

If x_2 is chosen as pivot column, then s_2 will be the pivot row, with an increase in the value of the objective function of $166^2/3$.

If x_3 is chosen as pivot column, then s_1 will be the pivot row, and the value of the objective function will increase by 200.

The total increases for x_1 and x_3 are co-equally 200. We assume that the variable with the lowest index is then chosen. Hence x_1 is chosen in preference to x_2, on account of the greater length of the step, despite the steeper ascent for the x_2-variable.

In this small 2 by 3 example the number of steps required to reach the optimum, is the same as when the rule of the steepest ascent was used earlier on in this chapter. (Three in both cases).

But it seems reasonable to assume that, in general, the rule of the highest step will solve any given problem in, on the average fewer steps, than the rule of the steepest ascent would require. The computational effort per step is obviously greater for the rule of the highest step, on account of the more elaborate search operation.

8.10 Degeneracy

A Linear programming problem is <u>degenerate</u> (at a particular vertex), if several quotients between an element of the value column and a corresponding element in the pivotal column are non-negative, equal and co-equally the smallest.

TABLEAU 8.10 A

AN EXAMPLE OF INITIAL DEGENERACY.

NAME	!!	X 1	X 2	X 3	!!	VALUE
S 1	!!	1	1	1	!!	10
S 2	!!	1	-1	-	!!	0
S 3	!!	1	-	-2	!!	0
T	!!	-1	0	0	!!	-

The s_2-row ($x_1 \leq x_2$) and the s_3-row ($x_1 \leq 2x_3$) both provide an eligible pivot. Both quotients 0 : 1 = 0 for s_2 and 0 : 1 for s_3 are equal.

Degeneracy normally occurs with a largish number of zero entries in the trivial value column, but can also arise during later iterations. If several non-zero quotients are co-equally the smallest, the next tableau will contain at least one zero in the value column, as two variables are reduced to zero and only one is eliminated.

Example

TABLEAU 8.10 B

EQUAL ELEMENTS LEADING TO DEGENERACY.

NAME	!!	X 1	X 2	X 3	!!	VALUE
S 1	!!	1	1	1	!!	10
S 2	!!	(1)	-1	-1	!!	1
S 3	!!	1	-2	1	!!	1
T	!!	-1	0	0	!!	-

In this case it is assumed that the restriction with the lowest index is chosen and a zero will occur on the right-hand side in the next tableau

TABLEAU 8.10 C
NON-INITIAL DEGENRACY.

NAME	!!	S 3	X 2	X 3	!!	VALUE
S 1	!!	-1	2	2	!!	10
X 1	!!	1	-1	-1	!!	-
S 3	!!	-1	-1	2	!!	1
T	!!	1	-1	-1	!!	-

The form in which degeneracy usually occurs is the one of the first example. Multiple degeneracy of the trivial basis due to a large number of zeros in the value column is quite common.

Contrary to what is stated by some authors* degeneracy is a problem which gives - in my experience - rise to computational problems.

There are in fact two problems: convergency and accuracy. Of these only the first is usually covered in the literature. If there is no degeneracy the simplex Algorithm increases the value of the objective function by a positive amount at each step. Therefore a vertex which has once been a basic solution cannot be met again as this would imply a reduction in the value of the objective function.

Since the number of vertices is finite the algorithm must sooner or later end. This proof of convergence does not hold in the case of degeneracy. Well, I have met cases where degenerate problems were not solved within a reasonable time limit, whereas small changes in the value column, i.e. putting 0.0000001 instead of the exact zero did result in a solution within a normal jobtime limit.

The other problem concerns accuracy. Consider the following problem

TABLEAU 8.10 D
A DEGENERATE PROBLEM, SEE ALSO 8.10 E.

NAME	!!	X 1	X 2	X 3	F 1	F 2	F 3	!!	VALUE
S 1	!!	-0.99	0.02	0.04	(1)	-	-	!!	0
S 2	!!	0.00	-0.92	0.05	-	1	-	!!	0
S 3	!!	0.07	0.03	-0.97	-	-	1	!!	0
S 4	!!	0.42	0.37	0.49	-	-	-	!!	6.34
T	!!	0	0	0	-1	-1	-1	!!	-2

*
Thus R.C. Geary and J.E. Spencer in their "Elements of Linear programming" [14] p.55, refer to degeneracy as "a special case of trivial practical importance which usually receives a disproportionate attention in the textbooks."

The system represents an input-output allocation model* and it is desired to maximise total final output, of 3 national sectors of production subject to the appropriate interindustry relations, and a macro-economic factor supply limit. Typical of this class of systems is the square block in the top left-hand side dominated by a negative diagonal. Some of the off-diagonal elements in this block can be quite small.

Let us see what happens if the rules and procedures discussed so far are applied to this system. The first pivot has been marked already, but we reproduce below tableau number 4, where things become interesting.

TABLEAU 8.10 E
DEGENERACY, LEADING TO CHOICE OF A SMALL PIVOT.

NAME	!!	X 1	X 2	X 3	S 1	S 2	S 3	!!	VALUE
F 1	!!	-0.99	0.02	0.04	1	-	-	!!	0
F 2	!!	(0.00)	-0.92	0.05	-	1	-	!!	0
F 3	!!	0.07	0.03	-0.97	-	-	1	!!	0
S 4	!!	0.42	0.37	0.49	-	-	-	!!	6.34
T	!!	-0.91	-0.86	-0.87	1	1	1	!!	0

At this stage, x_1 is the pivot column. The diagonal element of -0.986 is not eligible on account of its negative sign. Let us assume that the search for the smallest quotient is done (going down along the x_1-column), by looking at candidate-pivots, only if they give rise to a smaller quotient than the previous one. Then f_2 is the next pivot-row variable, as the quotient for f_3 is not smaller. But obviously f_3 should have been taken, and in general the largest of equally eligible candidate-pivots should be selected.

A further point arises in connection with the use of floating point zeros. Some machines substitute the absolute smallest number which can be represented as a floating point number, for the number zero. In that case zero plus zero is greater than zero, and a number which is in any meaningful sense zero could become the pivot, despite the fact that the programmer excluded zero pivots.

I have seen the wrong "outcome" for a multi-degenerate input-output allocation model in two separate instances. The first case involved academic research supervision and a programme

---

*See also my book "Allocation Models and their use in economic planning" [18]

written by myself. The second case involved consulting work, and the programme was written by an employee of the client organization. In that second case the distortion of the specified linear programming problem went as far as finding a problem with a feasible trivial basis empty, i.e. not to have any feasible solution at all.

In both cases an amendment in the text of the programme, i.e. attending to degeneracy, produced the correct results. The most simple solution to this problem (used in both cases), is to replace a zero in the value column, by a very small positive number, e.g. 0.000000001. Thus, in our example, the search for the smallest quotient would consider 0.000000001 : 0.001 as the relevant quotient for the f_2-row, and 0.000000001/0.74 as the relevant quotient for the f_3-row. This means, in effect, that the largest of the available pivots (i.e. 0.074 in the above example), is chosen. (This type of change is known as perturbation of the originally degenerate problem).

The "time expired" degenerate linear programming problems which I have met, may, or may not have been cases of true cycling in degeneracy.

Loss of accuracy for the reasons indicated above, can also cause spurious steps. This is so, in particular if spuriously negative entries in the value column arise. The reason is that the methods for finding a feasible solution in the first place (to be discussed in the next chapter), are on the whole less efficient than the elementary optimizing algorithm. And spurious negative entries in the value column would cause a spurious activation of this so-called "Phase I" part of the programme.

One further aspect of the problem of degeneracy is computing efficiency, i.e. avoiding unnecessary steps, irrespective of the issue of cycling.

On balance, my recommendation is to modify the problem at the outset, and make it into a non-degenerate problem. Zeros in the value column are all replaced by small positive numbers. These small numbers are (for example)

$$e_i = 0.000\ 000\ 000\ 001 \quad \left(1 + \sum_{\text{if } a_{ij}<0}^{j} (-a_{ij})\right) \qquad (8.10.1)$$

i.e. one takes the number one, adds to it the absolute value of all the negative entries in the relevant row, and scales the whole lot down.

There are two main reasons for not simply taking the same number for each row:

a) In some systems there are columns with the same entry in a whole series of rows. If such a column becomes pivot-column against a pivot-row with a pseudo-zero in the value column, when all pseudo-zeros are equal one is back at multiple degeneracy.

b) The suggested variation tends to prefer pivot-rows with few or with only small negative entries to rows with many large negative entries. The effect of this is to avoid these negative entries to be added to the corresponding entries in the target row, thus reducing the chances of having to enter the corresponding columns as pivot columns at the next step.

It is appropriate at this point, to comment on the discussion degeneracy in the literature, and the methods recommended for its resolution. First consider the "naive" rule of taking the top one of any zero quotients, i.e. the first zero in the value column for which a corresponding entry in the pivotal column is positive. This is not a recommended method, simple and practicable in the majority of cases as it may be. It has been shown to lead to cycling, independently, by at least two people, using examples constructed to that purpose. These two examples are: Tucker's as referred to by Zoutendijk*, and Beale's as referred to by Van de Panne**. Both examples involve initial degeneracy with two zeros in the value column and both become readily solvable if the zeros are replaced by equal pseudo-zeros. The fact that pertubation with equal pseudo-zeros is effective in these examples obviously does not prove that such a method is always effective - nor is that method recommended here.

---

*Zoutendijk, G., Mathematical Programming Methods [42], section 3.5. No bibliographical reference to Tucker given, private correspondence assumed.

**Van de Panne, C., Methods of Linear and Quadratic Programming, [36], section 3.5. No bibliographical reference to Beale given.

The following is the method recommended by Charnes [4], [5] Ch VI, and methods which are in essence the same are still the ones recommended by the dominant tradition. Replace each constant b_i by

$$b_i^* = b_i + \varepsilon^i \qquad (8.10.2)$$

The dominant tradition has it that the actual figures in the tableau are left as they are, including exact zeros, and one then devises additional rules for selecting the pivot row assuming that the trivial basis has been perturbed according to (8.10.2).

What this effectively means may be illustrated by reference to tableau 8.10e. The choice of the leaving variable is in first instance restricted to f_2 and f_3 only. (f_1 is not eligible because the quotient is classified as -0 i.e. negative and s_4 is not eligible because the quotient is in any case not the smallest, being non-zero).

To choose between f_2 and f_3 as leaving variable, we now refer to the s_1 column as substitute value column. The s_1-column makes the biggest "pseudo-zero" contribution to the value column - if any non-zero entries in the s_1 column are available. In those rows in which the s_1-column contains non-zero entries, pseudo-zero entries in the value column are dominated by the ε entry in the original s_1-restriction.

Those "b_1^* dominated" entries could either be just the original ε entry in the s_1-row - if the restriction were still in the non-binding form with the slack-variable in the list of basic variables - or the updated s_1-column, multiplied by ε^1. (Compare 8.7.2 and 8.7.3).

In fact in the example at hand, no non-zero entries in the s_2 and s_3-row are available in the s_1-column and we are still left with an undecided choice between s_2 and s_3 as pivotal row. Therefore, not having resolved the choice of a pivotal row, we refer to the s_2-column as substitute value column and find as relevant quotients:

$(1 * \varepsilon^2)/0.00$ in the f_2-row

$(0 * \varepsilon^2)//0.07$ in the f_3-row, the latter quotient being rather smaller.

Charnes has proved that this method cannot lead to cycling. However, it leaves the door wide open for what has been indicated above as the main danger of degeneracy in practice, loss of numerical accuracy. The method comes in practice very near to choosing the last rather than the first of any available zeros. It differs sufficiently from that rule to guarantee non-cycling but not to guarantee meaningfully non-zero pivots! If the entries of 0.00 in the s_2/x_1 cell and 0.07 in the s_3/x_1 cell in tableau 8.10d are interchanged, the method of exponential perturbation would opt for the 0.00 entry (which is not actually a zero) as pivot in tableau 8.10e.

The choice of the pivot depends in no way on the actual figure of the candidate-pivot, it is enough that it is a positive figure. It would appear that while the relatively simple device of actually perturbing the trivial basis according to (8.10.1) is at the cost of a small loss of accuracy in the outcome, the exponential perturbation, whether actually performed or hypothetically assumed, is in practice prone to much greater inaccuracies, and the more so if actual perturbation is replaced by a rule which assumes hypothetical perturbation with a number ε approximating zero. (The actual perturbation would exclude pure rounding errors as pivots.) In principle, that gain in accuracy has to be balanced against the fact that the method recommended above is not supported by a proof that it will never lead to cycling.

8.11 Simplex Tableaux and Vector Spaces

Suppose we know for some LP problem the names of (some of) the binding optimal restrictions. We could in that case interpret the partitioning introduced earlier in this chapter (8.1.1) in a "non simplex" way. This partitioning was also used in section 8.7. We now write an equivalent of (8.7.3) while suppressing reference to $\underline{s}_1$ which vector is assumed to be zero.

$$\left.\begin{array}{lll} \underline{x}_1 + A_{11}^{-1} A_{12}\, \underline{x}_2 & = A_{11}^{-1}\, \underline{b}_1 &) \\ \left[A_{22} - A_{21} A_{11}^{-1} A_{12}\right] \underline{x}_2 + \underline{s}_2 & = \underline{b}_2 - A_{21} A_{11}^{-1}\, \underline{b}_1 &) \\ \left[-\underline{w}_2' + \underline{w}_1' A_{11}^{-1} A_{12}\right] \underline{x}_2 + \tau & = \underline{w}_1' A_{11}^{-1}\, \underline{b}_1 &) \end{array}\right. \quad (8.11.1)$$

Note that $\underline{x}_1$ is effectively the vector of slack-variables associated with the first block-row of (8.11.1), if the system is written in equality form. In the terminology of Chapter VI, we might say that a simplex tableau from which a (binding) $\underline{s}_1$ block-column has been deleted, describes the remaining restrictions in the $s_1 = 0$ subspace. If the final block-row

is binding, the optimal solution is equivalent to the optimal solution of the following residual LP problem:

Maximise

$$\lambda = \left[\underline{w}_2' - \underline{w}_1' A_{11}^{-1} A_{12}\right] \underline{x}_2 \qquad (8.11.2)$$

Subject to

$$A_{11}^{-1} A_{12} \underline{x}_2 \leq A_{11}^{-1} \underline{b}_1 \qquad (8.11.3)$$

$$\left[A_{22} - A_{21} A_{11}^{-1} A_{12}\right] \underline{x}_2 \leq \underline{b}_2 - A_{21} A_{11}^{-1} \underline{b}_1 \qquad (8.11.4)$$

Comparison of (8.11.2) to (8.11.4) with (8.7.3) shows that the new (reduced) problem is related to the old one as follows:

Firstly, λ is the old objective function less the current solution value in (8.7.3).

$$\lambda = \tau - \underline{w}_1' A_{11}^{-1} \underline{b}_1 \qquad (8.11.5)$$

Comparison of (8.11.3) with the first block-row of (8.11.1) confirms that $\underline{x}_1$ is the vector of slack-variables in (8.11.3). If we prefer the more usual ordering with the slacks at the end, the first block-row of (8.11.3) could be written as

$$A_{11}^{-1} A_{12} \underline{x}_2 + \underline{x}_1 = A_{11}^{-1} \underline{b}_1 \qquad (8.11.6)$$

The vector of slack-variables in (8.11.4) is the old $\underline{s}_2$ vector.

The "non-simplex" interpretation arises because (8.11.2) to (8.11.4) is treated as an LP problem in its own right and we do not necessarily assume that all elements of $\underline{x}_1$ and none of $\underline{x}_2$ are in the collection of basic variables. We don't know whether the partitioning between $\underline{x}_1$ and $\underline{x}_2$ is the optimal one. We may have to assume that the partitioning of the tableau in two block-columns is done so as to find a square and non-singular block-pivot A_{11}.

Restrictions (8.11.3) and (8.11.4) list the restrictions of the LP problem in an $n-m_1$ dimensional co-ordinate space, the $\underline{x}_2$-space. The set of points satisfied by (8.11.3) and (8.11.4) is equivalent to the intersection of the $\underline{s}_1 = 0$ subspace with the set of points satisfying the originally stated restrictions in (8.1.1).

The choice of any particular sub-vector $\underline{x}_1$ as well as the non-negativity of the corresponding trivial solution, $\underline{x}_1 = A_{11}^{-1} \underline{b}$, is irrelevant provided the associated block-pivot A_{11} is non-singular. Alternatively, we can say that the original problem is stated in the m+n dimensional $\underline{x}$, $\underline{s}$ coordinate space, and the reduced problem is stated in the $m+n-m_1$ dimensional $\underline{x}_1$, $\underline{x}_2$, $\underline{s}_2$ co-ordinate space. We are now in a position to formulate and prove a theorem which we have - in a sense - already used. The simplex Algorithm investigates basic solutions where the number of binding restrictions equals the number of solved variables. We will show that if there is an optimal and feasible solution there is an optimal and feasible solution among the collection of simplex (basic) solutions.

Theorem

Let $x = \underline{x}^*$ be an optimal and feasible solution to the LP problem

Maximise

$$\tau = \underline{w}' \underline{x} \qquad (7.2.1)$$

Subject to

$$A \underline{x} \leq 0 \qquad (7.2.2)$$

$$\underline{x} \geq 0,$$

Then we may require the number of binding restrictions in that optimal solution or else in some co-equally optimal solution $\underline{x} = \underline{x}^{**}$, to be not less than the number of non-zero valued variables. We may also require the existence of a non-singular block-pivot as defined in (8.1.1).

Proof

First we consider the case of a trivial objective function, i.e. we assume $\underline{w}' = 0$. In that case we are free to declare any vector $\underline{x}$ which satisfies the stated restrictions (7.2.2), as being an optimal solution. That includes the optimal solution to a secondary LP problem which we obtain by substituting a non-trivial objective function for the trivial one. One could for example take (minus) the sum of all variables and maximand, i.e. replace the zero vector $\underline{w}'$ by a summation vector, which is not a zero vector. Therefore, if we obtain a proof of the theorem for LP problems with non-trivial objective functions, this includes the case of a trivial objective function. We may thus assume that the objective-function is not trivial.

For LP problems with non-trivial objective functions, the optimum solution satisfies the definition of an outward point as given in Section 6.7.

It was shown in Section 6.7 that, irrespective of the linearity or non-linearity of the side-conditions at least one side-condition is binding at an outward point. (The proof in section 6.7 depended only on the linearity of the maximand). In Chapter VI trivial restrictions were excluded by assumption. We do not now make that restrictive assumption, at least not as a restrictive assumption.

In the case of a linear restriction, triviality (defined in Chapter VI as not excluding any vector in the co-ordinate-space), is possible only if for that particular i-restriction.

$$\underline{a}_i' = 0 \qquad (8.11.7)$$

is true.

For $b_i > 0$ such a restriction is trivial. We assume that we never consider such a restriction as binding and we will in fact still be able to prove the theorem. No difference in the set of feasible points arises if we discard it. For $b_i < 0$ no meaningful problem has been stated. A restriction which states that zero is less than a negative constant cannot be satisfied at all. $b_i < 0$ will not be true for $\underline{a}_i' = 0$. We may therefore assume, without loss of generality that all restrictions are non-trivial.

Therefore the proof which we carried in Section 6.7 for an outward point is applicable and at least one non-trivial restriction may be required to be binding in the optimum. In general, we cannot identify binding restrictions in isolation from the list of basic variables. But let us for the sake of argument assume that we could identify at least one binding restriction of the optimum solution of an L.P. problem. (We know there is at least one). We can make use of this result in two cases in two different ways.

In the special case of only one variable, the proof is complete, as we have shown that at least one restriction is binding. In the more general case of n variables ($n > 1$), the one binding restriction defines a linear (Euclidean) subspace and the optimal solution is part of that subspace of n-1 dimensions. (Or m+n-1 dimensions if we could the slacks as co-ordinate-directions).

Since the one binding restriction is non-trivial we can always find a variable which provides a 1 by 1 non-singular block-pivot A_{11}. Finding a non-trivial binding restriction is enough to show that a partitioning by (8.1.1) exists. We therefore have a well-specified residual problem, if we delete reference to the slack of the one binding restriction from the originally specified problem.

Example (of a residual problem, proof itself continued below)

Take the numerical example in section 7.4 Maximise $\tau = 3x_1 - x_2$

Subject to

$$x_1 + x_2 \leq 7$$

$$-3x_1 - x_2 \leq -6$$

$$-x_1 + x_2 \leq 2$$

Once we know that the first restriction $x_1 + x_2 \leq 7$ is binding in the optimal solution we may pivot (for example) x_2 against the slack of the first restriction.

TABLEAU 8.11 A
A SIMPLEX STEP MAY LEAD TO A RESIDUAL PROBLEM. (TABLEAU 8.11 B)

NAME	X 1	X 2	VALUE
S 1	1	(1)	7
S 2	-3	-1	-6
S 3	-1	1	2
T	-3	1	-

and the up-dated tableau is (see section 8.8 for the up-dating rules in a shortened tableau)

TABLEAU 8.11 B

THE RESIDUAL PROBLEM, INDICATED IN TABLEAU 8.11 A.

NAME	!!	X 1	S 1	!!	VALUE
X 2	!!	1	1	!!	7
S 2	!!	-2	1	!!	1
S 3	!!	-2	-1	!!	-5
T	!!	-4	-1	!!	-7

DITTO, RE-INTERPRETED.

NAME	!!	X 1	!!	VAL.
S 1	!!	1	!!	7
S 2	!!	-2	!!	1
S 3	!!	-2	!!	-5
T	!!	-4	!!	-7

The x_2, s_1 pivot is not a very well-advised pivot. Without the hindsight that $x_1 + x_2 \leq 7$ is binding in the optimal solution it is in fact a particularly ill-advised pivot because it replaces on violated restriction (the second) by another (the third). But it serves to state the residual problem. If $x_1 + x_2 \leq 7$ is binding in the optimum then we find the optimal x_1 from the residual problem.

Maximise

$$\lambda = 4\, x_1$$

Subject to

$$x_1 \leq 7$$

$$-2x_1 \leq 1$$

$$-2x_1 \leq 5$$

$$(x_1 \geq 0)$$

Continuation of the proof

The extremum property that at least one restriction is binding is applicable to this residual problem as well as to the original problem. In the particular example, this completes the proof, as only one variable is left and we only need to find the most binding restriction. In the general case, finding another non-trivial restriction binding in the reduced problem allows us to find a pivot in that row and hence a non-singular block-pivot of which the order is one greater than at the previous partitioning. (The determinant of a matrix has the same absolute value as the recursive product of the pivots which may be used to invert it, see section 5.5). Therefore if there is an optimal solution there is another non-trivial binding restriction, as long as there is an unused co-ordinate-direction.

The number of explicit co-ordinate directions in each residual problem is one less than in the corresponding full problem. If we identify an explicit restriction, i.e. the non-negativity of some s_i becomes binding, we eliminate the explicit reference to a variable by means of an elimination step, some x_j becomes a slack in (8.11.3). Since the statement of an L.P. problem does not require a positive righthand-side, this is true, even if a negative pivot were used to eliminate the slack-variable s_i. If we identify a binding "tacit" restriction we merely drop further reference to the zero-valued variable.

The end of this hypothetical process of identifying binding restrictions, is no variables to change being left.

Therefore the number of restrictions identified in the course of the process, and the order of the block-pivot is the initially available number of variables. The actual number of exactly fulfilled restrictions cannot be less.
q.e.d.

The number of binding restrictions is not less than (rather than always equal to) the number of co-ordinate directions because of two footnotes to the proof.

Firstly, we assumed that if we met a trivial restriction in some subspace, we would discard it. This may be an additional binding optimal restriction. It may even be an additional non-trivial restriction. It does not follow that a restriction which is trivial in some subspace is also trivial in the fully specified problem.

Example

Maximise $\tau = x_1$

subject to

$$x_1 + x_2 \leq 5$$

$$-x_1 - x_2 \leq -5 \quad (\text{i.e. } x_1 + x_2 \geq 5)$$

The optimal solution to this problem is given in the following tableau

TABLEAU 8.11 C

ILLUSTRATION OF
SUBSPACE-TRIVIALITY.

NAME	!!	S1	X2	!!	VALUE
X 1	!!	1	1	!!	5
S 2	!!	1	-	!!	-
T	!!	1	1	!!	5

The second restriction is trivial in the $s_1 = 0$ subspace. It makes no difference to the solution (the first restriction is binding on the optimal solution anyway) but the second restriction is not a trivial restriction. It excludes, for example, the origin from the set of feasible vectors x_1, x_2. Having discarded the s_2-restriction as trivial in the $s_1=0$ subspace, we identify the non-negativity of x_2 as the second binding restriction. Secondly there may be several equally binding restrictions on the last variable, e.g.

$$x \leq 2, \quad 2x \leq 4, \quad 3x \leq 6, \text{ etc.}$$

Again the restrictions may be dependent in some subspace but they might nevertheless be independent restrictions in the fully specified problem.

The assumed possibility of identifying successive binding restrictions proves that, (because there is at least one non-trivial binding restriction in any L.P. problem) the optimal solution is a vertex, a basic solution. As an algorithm this approach has one fatal flow: we cannot in fact identify an individual binding optimal restriction without first solving the problem as *a* whole. But the simplex method which investigates only basic solutions, will get us there.

There are some further carefully formulated passages in the theorem, i.e. "we may require" and "or in one of equal value". There are indeed L.P. problems for which "non simplex" solutions (with fewer binding restrictions) are co-equally optimal with Simplex solutions. This possibility arises when the objective function is trivial in some sub-space.

Example

Maximise

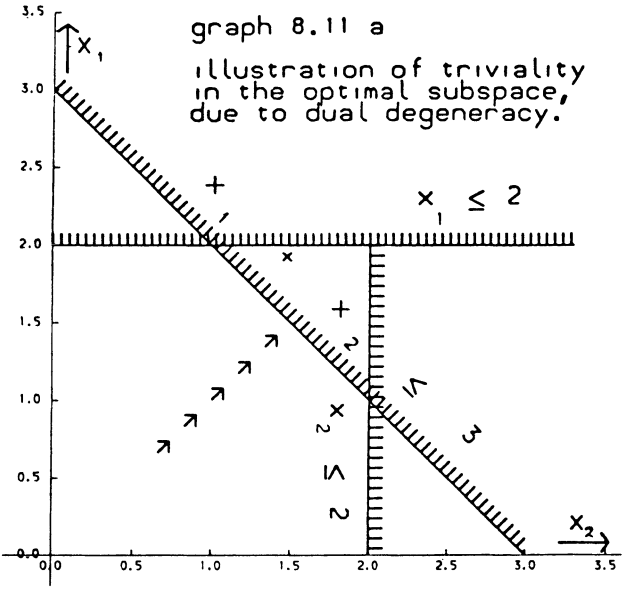
$\tau = x_1 + x_2$

Subject to

$x_1 \leq 2$

$x_2 \leq 2$

$x_1 + x_2 \leq 3$



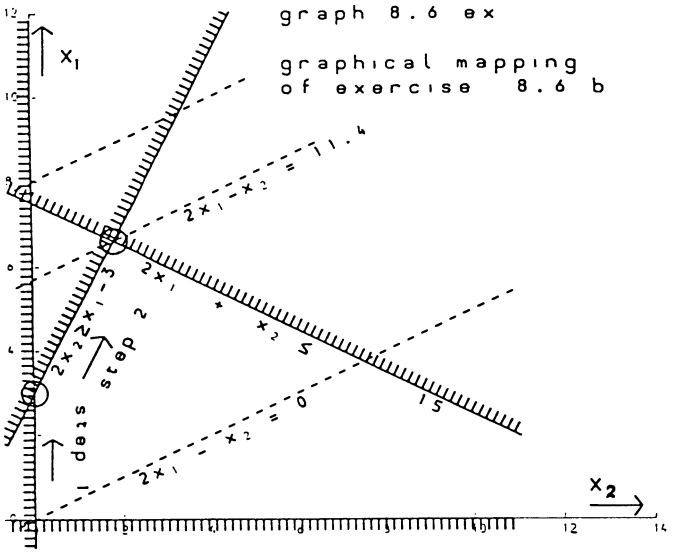
Two vertices (Simplex solutions) are co-equally optimal at $\tau = 3$. Any point on the facet between them is also co-equally optimal. This situation arose because in the $(s_3 = x_1 + x_2 = 3)$ subspace, the objective function is trivial.

The fact that $x_1 = 1.1$, $x_2 = 1.9$ is optimal, yet only one restriction is binding does not invalidate the theorem. It is the case of a trivial objective function referred to in the proof. That objective function is trivial in the $s_3 = 0$ subspace, not elsewhere in the x_1, x_2 co-ordinate space.

N.B.

A summary of this proof may be found in section 14.8, where the same theorem is stated and proved for the more general non-linear (and non-convex) case.

ANSWER-SHEET 8.6 EX



THE SET-UP TABLEAU:

NAME	X 1	X 2	S 1	S 2	T	VALUE
S 1	2	1	1	-	-	15
S 2	①	-2	-	1	-	3
T	-2	1	-	-	1	-

THE RULE OF THE STEEPEST ASCENT INDICATES X1 AS (ONLY ELIGEABLE) INCOMING VARIABLE. THE RATIOS BETWEEN THE VALUE COLUMN AND THE X1 COLUMN ARE 15/2 = 7.5 FOR S1 AND 3/1 = 3.00 FOR S2. THE RULE OF THE SMALLEST QUOTIENT THEREFORE INDICATES S2 AS THE LEAVING VARIABLE.

THE VERTEX FOLLOWING STEP 1:

NAME	X 1	X 2	S 1	S 2	T	VALUE
S 1	-	⑤	1	-2	-	9
X 1	1	-2	-	1	-	3
T	-	-3	-	2	1	6

THE RULE OF THE STEEPEST ASCENT NOW INDICATES X2 AS THE (ONLY ELIGEABLE) INCOMING VARIABLE.

THE RATIO 3/-2 = -1.50 FOR X1 AS LEAVING VARIABLE IS NOT ELIGEABLE, WE THEREFORE CHOOSE S1 AS LEAVING VARIABLE.

THE VERTEX FOLLOWING STEP 2:

NAME	X 1	X 2	S 1	S 2	T	VALUE
X 2	-	1	0.20	-0.40	-	1.80
X 1	1	-	0.40	0.20	-	6.60
T	-	-	0.60	0.80	1	11.40

NOW WE DON'T FIND A NEGATIVE ENTRY IN THE OBJECTIVE FUNCTION ROW. WE HAVE FOUND THE OPTIMUM.

CHAPTER IX

THE SEARCH FOR A FEASIBLE SOLUTION

9.1 The case of one unfulfilled restriction

In Section 7.1 we defined the L.P.-problem as:

(a) finding a feasible solution,

(b) finding, among all feasible solutions, the optimum.

This definition is slightly shorter than the one formally given in Section 7.1. This is so, because we now use some terms, which we had not yet defined in Section /.1. The substance is the same.

It is standard practice to tackle - if needed - (a) first. The operation is known as "Phase I", leaving optimization as "Phase II".

The Simplex Algorithm gave us as yet only a solution to (b), on the assumption that we had one to (a). The reason for this curious order is this:

The search for a feasible solution is done with help of the maximizing algorithm.

The method of solving the "Phase I" problem, offered in this book, is probably best described as the use of a substitute objective function. It should, however, be stressed that the method of artificial variables (summarized briefly in Section 9.5), is in fact more commonly used, or at least reported in the literature. In the case of just one violated restriction, the substitute objective function is, in effect the slack of that one violated restriction.

Consider the partitioned system:

$$\begin{aligned} C\,\underline{x} + \underline{t} &= \underline{c} \\ \underline{d}'\,\underline{x} + \mu &= \delta \\ -\underline{w}'\,\underline{x} + \delta &= 0 \end{aligned}$$

We have split off one row from A, partitioning A into a remainder matrix C, and the row-vector $\underline{d}'$. The vectors $\underline{s}$ and $\underline{b}$ are correspondingly split into scalars μ and δ, and the remainder $\underline{t}$ and $\underline{c}$.

We assume $c_i \geq 0$ ($i = 1, 2, \ldots m-1$), but $\delta < 0$. The trivial solution satisfies the first m-1 restrictions, but not the last one. The problem of finding a feasible solution will then be solved by finding the maximum value of $\mu = -\underline{d}' \underline{x} + \delta$. We do not always really need the maximum value of μ. But if a non-negative solution is at all possible, the maximum will be a non-negative solution. We therefore start with maximizing $-\underline{d}' x$, subject to $C \underline{x} \leq \underline{c}$.

For example, consider the problem used in Section 7.4 to illustrate the possibility of graphical solution.

The initial Simplex tableau for that problem (now suppressing the τ and "check" columns) is:

TABLEAU 9.1 A
SET-UP TABLEAU OF A LINEAR PROGRAMMING PROBLEM, WITH AN INFEASIBLE STARTING SOLUTION.

NAME	!!	X 1	X 2	!	S 1	S 2	S 3	!!	VALUE
S 1	!!	1	1	!	1	-	-	!!	7
S 2	!!	-3	-1	!	-	1	-	!!	-6
S 3	!!	-1	1	!	-	-	1	!!	2
T	!!	3	-1	!	-	-	-	!!	-

We treat the s_2-row as a substitute for the objection function. For the time being we try to increase s_2 instead of τ. The most negative element in the s_2-row is provided by the x_1-column. Hence x_1 will be the first pivot-column. For the moment we will assume, that the s_2-row (now being the objective function), is excluded as pivot-row. The real objective function is of course not eligible as a pivot-row either. This leaves us with the choice of s_1 and s_2 as candidates for becoming pivot-row. Here s_1 produces the only pair of positive numbers, and will be our pivot row. We will indicate this in the tableau, which now is:

TABLEAU 9.1 B
SET-UP TABLEAU WITH AN INFEASIBLE STARTING SOLUTION, AND PIVOT MARKED.

NAME	!!	X 1	X 2	!	S 1	S 2	S 3	!!	VALUE
S 1	!!	①	1	!	1	-	-	!!	7
S 2	!!	-3	-1	!	-	1	-	!!	-6
S 3	!!	-1	1	!	-	-	1	!!	2
T	!!	3	-1	!	-	-	-	!!	-

We now carry out the indicated step. In graph 7.4a this step corresponds to a movement along the x_1-axis, until the top left-hand corner of the quadrangle is reached. The new Simplex tableau will be:

TABLEAU 9.1 C
UPDATED TABLEAU OF AN LP PROBLEM, WITH INFEASIBLE STARTING SOLUTION, AND A NOW AMPLY FULFILLED RESTRICTION.

NAME	!!	X 1	X 2	!	S 1	S 2	S 3	!!	VALUE
X 1	!!	1	1	!	1	-	-	!!	7
S 2	!!	-	2	!	3	1	-	!!	15
S 3	!!	-	2	!	1	-	1	!!	9
T	!!	-	-4	!	-3	-	-	!!	-21

This tableau represents a feasible solution, witness the non-negativity of the "value" column. The further search for the optimum can then be carried out with the methods described in Chapter VIII. There are, however, two remarks to be made in this stage.

Firstly, the choice of s_1 as the pivot-row is in fact not quite correct. It does lead to the desired result. But cases may arise, where the extension of the rule for choosing a pivot row, obtained in Chapter VIII to non-feasible problems, could lead to trouble. The problem will be discussed in the next paragraph.

Secondly, one should realize, that the search for a feasible solution, does not always end with finding one.

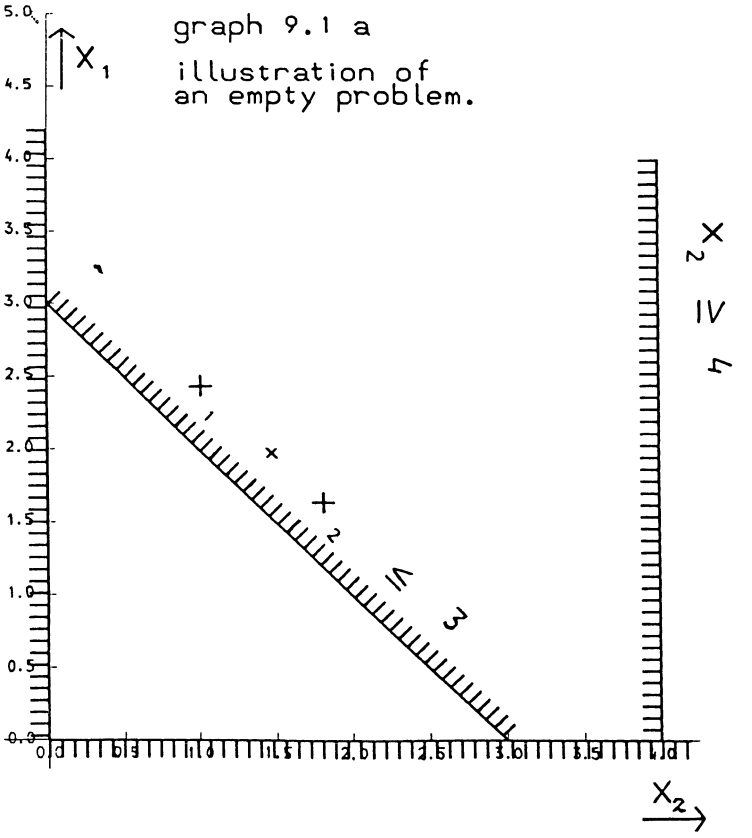
If we have the system:

maximise x_1, subject to:

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ - x_2 &\leq -4 \end{aligned}$$

We will start with maximizing x_2. But clearly, the maximum of x_2 will be obtained with $x_2 = 3$. Yet no value of x_2, smaller than 4, will satisfy the second restriction. The system has no feasible solution.

An L.P.-problem, to which no feasible solution can be found, is known as an empty problem. Since the present problem again involves only two variables, we can illustrate it graphically. Again the shaded side of the restrictions indicates the non-admissable side (See graph 9.1a).



There is no place in the graph, that is at the non-shaded side of all the restrictions. The restrictions contradict each other. This reveals the emptyness of the problem graphically.

The fact that the problem is an empty one, can also be demonstrated by the Simplex Algorithm. This takes only one step; two Simplex tableaux.

These Simplex tableaux will be:

TABLEAUX 9.1 D AND 9.1 E

RECOGNITION OF AN EMPTY PROBLEM
BY THE SIMPLEX METHOD.

NAME	X 1	X 2	S 1	S 2	VALUE
S 1	1	①	1	-	3
S 2	-	-1	-	1	-4
T	-1	0	-	-	-

NAME	X 1	X 2	S 1	S 2	VALUE
X 2	1	1	1	-	3
S 2	1	-	1	1	-1
T	-1	-	0	-	0

The s_2-row should now be read:

$$x_1 + s_1 + s_2 = -1$$

This is an entirely non-negative combination of the non-negative variables x_1, s_1 and s_2. Such a linear combination can never be equal to something negative. It is an "impossible" restriction. Whenever we find in any row, a negative element in the "value" column, and positive elements (or zeros) in all other columns, this certifies the problem to be empty. This situation should then be listed as an indication of the end of the algorithm.

When we are searching for a negative element in a row, with a view to finding a new pivot column, and do not find one, this will always certify the end of the algorithm. If the row is the objective function, the inference is that we have reached the optimum. If the row is not the target-row, it must be a row describing one of the basic variables or a combination of them in some versions of the Simplex algorithm. The value of the

corresponding variable must be negative. Then the problem has been found empty. The situation will never arise with a row describing a positive variable. We do not perform such search operations in rows describing positive variables.

9.2 Again: The choice of the pivot row

In Section 8.3, we discussed the choice of the pivot row. We then assumed the existence of a feasible solution. The rule then said that we should divide the (positive) values of the basic variables by the positive elements in the pivot column, and select the smallest among these quotients. In this form, the rule of the smallest quotient excludes the choices of negative pivots. This rule will no longer do. We must assume the normal result of the feasibilization-algorithm to be: unfulfilled restrictions becoming exactly fulfilled, by becoming pivot row.

Consider the L.P.-problem, that was first discussed in Section 7.4, as an example of graphical solution. We solved a feasible solution to this problem, by extending the old rule of the smallest quotient, to this non-feasible problem. (See the preceding Section 1 of this chapter). But from graph 7.4a it will be clear that $x_1 = 2$ would have been sufficient, while we increased x_1 in fact until $x_1 = 7$. The choice of s_2 itself would have been more efficient. The step we should have made, is given below in the corresponding Simplex tableaux. (Tableaux 9.2a and 9.2b). Tableaux 9.2b and 9.1c both indicate feasible solutions, but 9.2b indicates the higher solution value .

TABLEAUX 9.2 A AND 9.2 B
SELECT THE SMALLEST QUOTIENT, WITH A NEGATIVE PIVOT.

NAME	!!	X 1	X 2	!	S 1	S 2	S 3	!!	VALUE
S 1	!!	1	1	!	1	-	-	!!	7
S 2	!!	(-3)	-1	!	-	1	-	!!	-6
S 3	!!	-1	1	!	-	-	1	!!	2
T	!!	3	-1	!	-	-	-	!!	-

NAME	!!	X 1	X 2	!	S 1	S 2	S 3	!!	VALUE
S 1	!!	-	0.67	!	1	0.33	-	!!	5
X 1	!!	1	0.33	!	-	-0.33	-	!!	2
S 3	!!	-	1.33	!	-	-0.33	1	!!	4
T	!!	-	-2	!	-	1	-	!!	-6

This is not just a problem of efficiency, of finding the optimum in the shortest possible number of steps.

For consider the following L.P.-problem

TABLEAU 9.2 C

TO ENTER X1 INTO THE BASIS AT ALL, WE NEED A NEGATIVE PIVOT.

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
S 1	!!	(-3)	-1	!	1	-	!!	-6
S 2	!!	-1	1	!	-	1	!!	2
T	!!	3	-1	!	-	-	!!	-

This is basically the same problem as the preceding one. The first restriction, which was not binding on the optimum, has been omitted. As a consequence of this the second and third restrictions have been renamed first and second. Like the preceding one, this problem has a normal optimal solution, again $x_1 = 1$ and $x_2 = 3$. However, once we take x_1 as our pivot column, we are forced to take a negative pivot, as there are no positive numbers in the x_1-column.

We now formulate the new rule for choosing the pivot row, as follows:

For each non-zero element of the pivot-column, establish whether the corresponding element in the "value" column has the same sign (zeros to be counted as positive). For all pairs of negative values of basic variables and negative elements of the pivot column, find the largest quotient of the variable divided by the candidate-pivot. For all pairs of positive numbers, and this one pair of negative numbers, find the smallest quotient of the value of the variable divided by the candidate-pivot. It will be observed, that all the quotients are positive.

The following examples may illustrate this new rule. We first consider the example in Section 8.3. This is a feasible problem, hence no pairs of negative numbers are available. In that case the new rule is identical to the old one. Then there is the example of Section 7.4, discussed earlier in this section. There is one pair of positive numbers (s_1), and one pair of negative numbers (s_2). The s_3-row is not eligible as a pivot-row, as this is a pair with unequal signs. If we took s_3 as the new pivot-row, x_1 would be pivoted into the basis with the negative value of -2.

The largest of the quotients of negative numbers is $-6 : -3 = 2$ for s_2, and it is the only quotient in this group. The

smallest of the quotients of positive numbers is 7 : 1 = 7 for s_1, again in fact the only one. From these two quotients the smallest must be chosen, and this is -6: -3 = 2 for s_2.

Then there is the example of the empty problem, discussed in this chapter (Section 9.1):

TABLEAU 9.2 D

A SMALLER QUOTIENT OF TWO POSITIVE NUMBERS HAS PRIORITY OVER A BIGGER QUOTIENT OF TWO NEGATIVE NUMBERS.

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
S 1	!!	1	(1)	!	1	-	!!	3
S 2	!!	-	-1	!	-	1	!!	-4
T	!!	-1	0	!	0	0	!!	-

We search for the most negative element in the s_2-row, and find x_2 as our pivot-column. We then consider the quotient 3 : 1 = 3 for s_1, which is the smallest (the only one) among quotients of positive numbers. And we consider -4 : -1 = 4 for s_2, being the largest (the only one) among quotients of negative numbers. From these two quotients, 3 for s_1 is the smaller one, and is chosen.

And consider the example in this section (see tableau 9.2c)

In this example, the "search" for the smallest/biggest quotient is again trivial on account of the presence of only one quotient of the appropriate sign.

We look for the most negative element in the s_1-row, find x_1 as the pivot column, and there are no quotients of positive numbers available, while there is one quotient of negative numbers. This situation is typical for feasibilization-problems. We have a row with a negative element in the "value" column, and we look for the most negative element in that row. Therefore, if we have found a pivot column, there must be at least one pair of negative numbers, namely in the unfulfilled restriction that gave us the pivot column. In the example at hand, we find s_1 as our pivot row.

The next example will help us to see the essential efficiency-advantage of the new rule. Consider the L.P.-problem as given in:

TABLEAUX 9.2 E AND 9.2 F

CHOICE OF THE BIGGER OF TWO QUOTIENTS OF NEGATIVE NUMBERS: FEASIBLE IN ONE STEP.

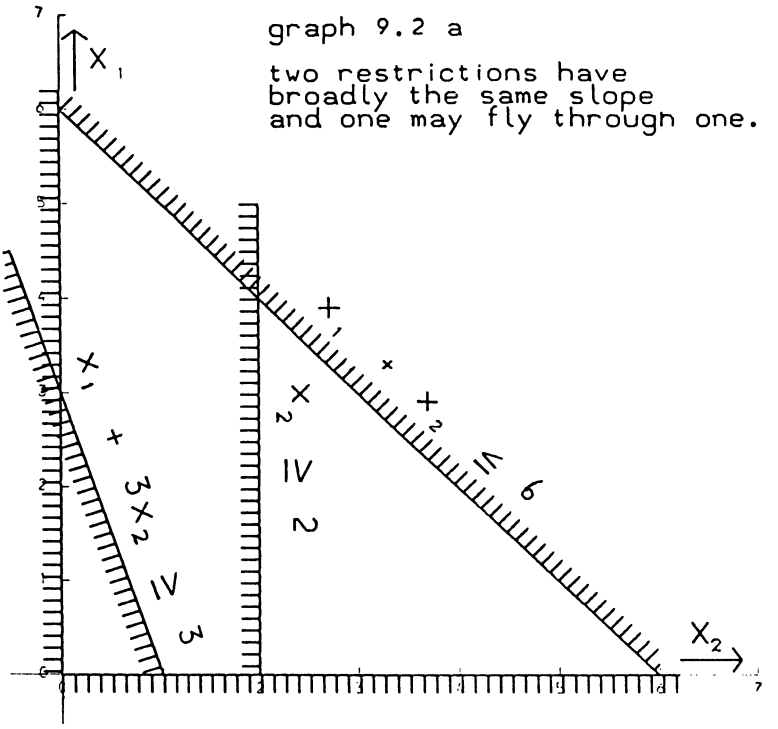
NAME	X 1	X 2	S 1	S 2	S 3	VALUE
S 1	1	1	1	-	-	6
S 2	-1	-3	-	1	-	-3
S 3	-	(-1)	-	-	1	-2
T	-1	1	-	-	-	-

NAME	X 1	X 2	S 1	S 2	S 3	VALUE
S 1	1	-	1	-	1	4
S 2	-1	-	-	1	-3	3
X 2	-	1	-	-	-1	2
T	-1	-	-	-	1	-2

Suppose we start with maximising s_2. We find -3 for x_2 as the most negative elements in the s_2-row.

There are now <u>two</u> pairs of negative numbers; -3: -3 = 1 for s_2 and -2 : -1 = 2 for s_3, of which s_3 provides the largest quotient of negative numbers. There is one quotient of positive numbers, 6 : 1 = 6 for s_1. Of the two quotients, 6 for s_1 and 2 for s_3, the latter one is the smaller. Therefore we choose s_3 as our pivot row. The next solution will then be $x_2 = 2$, with $s_1 = 4$ and $s_2 = +3$. We have satisfied two violated restrictions in one step.

As this again is a problem of only two variables, it can be illustrated graphically, which has been done in graph 9.2a.



From the graph, we observe that the proposed step will in fact bypass the second restriction, turning it from a violated into an amply fulfilled restriction.

In fact, if it were not because of this efficiency-aspect, we should have taken a more simple rule.

The "conservative" rule of taking always the smallest of all available positive quotients (including quotients of negative numbers), would always solve the problem. And it would have the advantage of being simple.

Under this "conservative" rule, a feasible solution would be reached in two steps, instead of in one. The corresponding Simplex tableaux would then be:

TABLEAUX 9.2 G, H, AND I

CHOICE OF THE SMALLER OF TWO QUOTIENTS OF NEGATIVE NUMBERS: FEASIBLE IN TWO STEPS.

NAME	X 1	X 2	S 1	S 2	S 3	VALUE
S 1	1	1	1	-	-	6
S 2	-1	(-3)	-	1	-	-3
S 3	-	-1	-	-	1	-2
T	-1	1	-	-	-	-

NAME	X 1	X 2	S 1	S 2	S 3	VALUE
S 1	0.67	-	1	0.33	-	5
X 2	0.33	1	-	-0.33	-	1
S 3	0.33	-	-	(-0.33)	1	-1
T	-1.33	-	-	0.33	-	-1

NAME	X 1	X 2	S 1	S 2	S 3	VALUE
S 1	1	-	1	-	1	4
X 2	-	1	-	-	-1	2
S 2	-1	-	-	1	-3	3
T	-1	-	-	-	1	-2

It will be observed that, apart from the interchanging of the position of the rows "x_2" and "s_2", this tableau is exactly the same, as the one that would have been obtained in one step, with the help of the more complicated rule.

There is, apart from its simplicity, yet another argument in favour of the "conservative" rule. Like the ordinary rule of the smallest quotients, the "conservative" rule tends to avoid small pivots. The more complicated "efficient" rule on the other hand, will to a certain extent favour the choice of small pivots. For consider the following problem:

TABLEAUX 9.2 J AND 9.2 K

CHOICE OF THE BIGGER OF TWO QUOTIENTS OF NEGATIVE NUMBERS: A VERY SMALL PIVOT RESULTS.

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
S 1	!!	(-0.01)	-1	!	1	-	!!	-1
S 2	!!	-1	1	!	-	1	!!	-1
T	!!	1	1	!	-	-	!!	-

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
X 1	!!	1	100	!-	100	-	!!	100
S 2	!!	-	101	!-	100	1	!!	99
T	!!	-	-99	!	100	-	!!	-100

If we start this problem with trying to get s_1 non-negative, no particular problem occurs. In that case x_2 becomes the pivot-column. And then only s_1 can become the pivot-row. But if we start with trying to get s_2 non-negative, some strange figures may occur. Then x_1 becomes the pivot column. We will then have to choose between two pairs of negative numbers:

$$-1.00 : -0.01 = 100 \text{ for } s_1 \text{ and } -1.00 : -1.00 = 1$$

for s_2. Choosing the larger of the two quotients then will give us the number 0.01 as pivot. Then any rounding errors in the s_1-row are multiplied by 100. And it is not at all certain that such a pivot was really needed. In fact, with the "conservative" rule, a feasible solution would be attained in two steps, without the occurring of such small pivots. The next two tableaux would then be:

TABLEAUX 9.2 L AND 9.2 M

CHOICE OF THE SMALLER OF TWO QUOTIENTS OF NEGATIVE NUMBERS: A BIGGER PIVOT RESULTS.

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
S 1	!!	-0.01	-1	!	1	-	!!	-1
S 2	!!	(-1)	1	!	-	1	!!	-1
T	!!	1	1	!	-	-	!!	-

NAME	!!	X 1	X 2	!	S 1	S 2	!!	VALUE
S 1	!!	-	-1.01	!	1	-0.01	!!	-0.99
X 1	!!	1	-1	!	-	-1	!!	1
T	!!	-	2	!	-	1	!!	-1

The problem can be overcome by modifying the "efficient" rule. It will now be:

For each non-zero positive element in the pivot-column, and a corresponding positive (or zero) value of the basic variable, find the smallest quotient of the value of the basic variable, divided by the corresponding element of the pivot column. Compute quotients of negative values of basic variables, divided by the corresponding elements of the pivot-column (candidate pivots) as far as the latter are negative and in absolute value greater than a specific small non-zero number. If one of these quotients should be larger than the smallest quotient of positive numbers, the row which originally gave the smallest quotient of positive numbers, is the pivot row. If all eligible quotients of negative numbers have been computed, while none of them is larger than the smallest quotient of positive numbers, the row in which the largest quotient of negative numbers was found, becomes the pivot row and the problem is feasible in one step. If no quotient at all has been found in this way, whether of negative numbers, or from positive numbers, find among the negative and non-zero elements of the pivot-column, corresponding to negative values of basic variables, the absolutely largest (= most negative) element in the pivot column; the corresponding row will be the pivot row. Note that we would not have entered such a column, unless a pair of negative numbers were available.

This seems quite a mouthful, because it should once be formulated with some precision. It can also be said a lot shorter:

Find the smallest quotient of positive numbers. Find out if any quotients of negative numbers are larger, but reject too small negative pivots. If a quotient of negative numbers is found to be larger, the quotient of positive numbers provides the pivot row. Otherwise choose the largest of the quotients of negative numbers. If no pivot at all is found in this way, we will be forced to accept a small negative pivot; take the largest (=most negative one) available.

The L.P. code which is offered in Section 12.3 applies this rule with some modifications; its bias for small pivots being one of the main reasons for these modifications.

When the current solution is not feasible, preference is given to columns which indicate an increase in the value of the specified objective function as well as in the value of a substitute objective function. In such "preferred" columns, negative pivots are not accepted at all. In other columns, representing variables which increase a substitute objective

function (the sum of the slacks of all violated restrictions), at the cost of a reduction in the value of the specified objective function, the criterion of the dual ratio is superimposed on that of row-selection. A discussion of that criterion is, however delayed to section 11.3.

Exercise

Starting with tableau 8.11a, solve the example-problem from section 7.4 several times, once for each pivot-selection rule discussed in this section. Use shortened tableaux with a sum+1 check column throughout. Check the correctness of your calculations by

a) Charting x_1 and x_2 in graph 7.4a, finding them to match vertices

b) Comparing with the full tableaux, as far as available in print

9.3 The choice between a number of violated restrictions

Suppose in the initial solution a whole series of restrictions is not satisfied. For instance, let s_3, s_{12}, s_{13}, s_{24}, s_{26} and s_{32} have negative slacks. In the preceding paragraph, we assumed that we chose our pivot column by finding the most negative number in the row, corresponding to a violated restriction, or the objective function. In what row should we look, if there are many negative values? One obvious approach would of course be: start with s_3, then take s_{12}, etc. We tackle the different restrictions in the order of their index number. If we start with s_3, we may find after one step, that we came to a solution satisfying s_{13}, by incident, but still have a negative element in the s_3-row. The collection is then s_3, s_{12}, s_{24}, s_{26} and s_{32}. We then again take the most negative element in the s_3-row, to find a pivot-column. After the second step, we may have satisfied s_3, but also incidentally s_{12} and s_{26}. Since we never take a quotient, larger than the smallest quotient of positive numbers, there are never new names added to the list of violated restrictions. Our collection is then s_{24} and s_{32}, we now take s_{24} as our next substitute for the target-row; etcetera.

At each step, we look for the negative variable with the lowest index-number. In that row, we look for the most negative number. If we do not find such a negative number, we know that the problem has no feasible solution. Otherwise we will always end up with finding a feasible solution.

This procedure is not necessarily an efficient one. Consider the following example:

TABLEAU 9.3 A

THE EFFICIENT STEP DOES NOT CORRESPOND TO THE STEEPEST ASCENT IN ANY SEPERATE RESTRICTION.

NAME	!!	X 1	X 2	X 3	X 4	X 5	!!	VALUE
S 1	!!	-2	-	-	-	-1	!!	-2
S 2	!!	-	-2	-	-	-1	!!	-2
S 3	!!	-	-	-2	-	-1	!!	-2
S 4	!!	-	-	-	-2	-1	!!	-2
S 5	!!	1	1	1	1	(-1)	!!	-3
T	!!	1	1	1	1	1	!!	-

In this problem, $x_5 = 3$, with s_5 as the corresponding exactly fulfilled restriction, is a feasible solution. This can be seen from the corresponding shortened Simplex Tableau:

TABLEAU 9.3 B

OPTIMAL AND FEASIBLE IN ONE STEP, AFTER STARTING WITH FIVE VIOLATED RESTRICTIONS.

NAME	!!	X 1	X 2	X 3	X 4	S 5	!!	VALUE
S 1	!!	-3	-1	-1	-1	-1	!!	1
S 2	!!	-1	-3	-1	-1	-1	!!	1
S 3	!!	-1	-1	-3	-1	-1	!!	1
S 4	!!	-1	-1	-1	-3	-1	!!	1
X 5	!!	-1	-1	-1	-1	-1	!!	3
T	!!	2	2	2	2	1	!!	-3

Now if the problem is started with maximizing s_5, this feasible solution is reached in one step; it is the optimum as well. But if we start with s_1, s_2, etc., we will have to run through six tableaux, before we reach a feasible solution. To some extent, this is just the result of the fact that the Simplex method is an iterative method; the properties of the optimum are unknown, before the optimum is actually solved. But there is one systematic aspect. If we start with maximizing s_1, we will take x_1 as pivot column. In doing so, we make s_5 more negative. This is so, because in the x_1-column, the element in the s_5-row is positive. Apparently "good" columns are columns with negative,

or at least non-positive elements in all rows representing violated restrictions. This cannot be made into a general rule for finding the pivot column. The normal case will be that all columns have both negative and positive signs. Could we perhaps make it the rule, that we should take the column with the largest number of negative elements in the rows that represent the violated restrictions? We could not. For consider the following example, where one step has been made according to this rule (shortened tableaux).

TABLEAUX 9.3 C AND 9.3 D

INCONSISTENT SUBSTITUTE OBJECTIVE FUNCTION, CHANGING AT EACH STEP: THE RESULT IS CYCLING

NAME	!!	X 1	X 2	X 3	X 4	X 5	!!	VALUE
S 1	!!	(1)	-	-	-	-	!!	1
S 2	!!	-1	-1	-	-	-	!!	-2
S 3	!!	-1	-	-1	-	-	!!	-2
S 4	!!	1	-	-	-1	-	!!	-1
S 5	!!	1	-	-	-	-1	!!	-1
T	!!	-	1	1	1	1	!!	-

NAME	!!	S 1	X 2	X 3	X 4	X 5	!!	VALUE
X 1	!!	(1)	-	-	-	-	!!	1
S 2	!!	1	-1	-	-	-	!!	-1
S 3	!!	1	-	-1	-	-	!!	-1
S 4	!!	-1	-	-	-1	-	!!	-2
S 5	!!	-1	-	-	-	-1	!!	-2
T	!!	-	1	1	1	1	!!	-

Again taking the column with the largest number of negative elements in rows 2-5, reproduces the initial solution, and leads to cycling. We really need the approach discussed in Section 1 of this chapter. As long as the list of the violated restrictions is the same, a substitute objective function should be increased at each step. And as long as the list of the violated restrictions is unchanged, this should be the same function at each following step. But there is a compromise.

We can maximize the sum of all negative slacks. Our substitute objective function will then simply be the sum of all unfulfilled restrictions. On the average, this rule has a higher probability of choosing "good" columns, relative to attacking the violated restrictions in the order of their indices.

Another reasonably efficient procedure is to apply the principle of steepest ascent to the selection of the restriction to be eliminated, but only intermittently. We choose a "badname", indicating the most negative valued slack-variable. Then we go on increasing ("maximizing") this negative slack-variable, until it is eliminated. This is more or less the procedure we first suggested in Section 9.1. The rule for selecting the pivot-row is then as follows:

First, consider the "badname" row itself as pivot-row. In the "badname" row, there is always a positive and non-zero quotient, between the negative entry in the value column (indicating the violated "badname" restriction) and the negative entry in the pivot column (selected as the most negative entry in that row).

Refuse all other negative pivots. Apply the rule of the smallest quotient between the one quotient of negative numbers already selected, and such quotients of positive numbers as may be found.

In our example (above) these rules would result in the same step as for the "efficient" rule. We would take s_5 as our (first) "badname" and provisionally select the s_5, x_5 element as pivot. Since there are no positive entries in the x_5 column at all, the question of selection of the smallest quotient does not arise.

This rule is, on the whole likely to require more steps than the use of all violated restrictions as substitute objective function, but it has the advantage of being simpler. (It also provides a better safeguard against loss of accuracy due to the choice of a very small pivot. A negative entry in the "badname" row, selected as the most negative in that row, is not likely to be very small in absolute value.

9.4 Pivot-selection in Phase I: the general case

By way of example, we will now tackle a "general" linear programming problem. With the word "general" we mean that all the problems discussed so far, are present in this example. The initial tableau is neither optimal, nor feasible.

We will use shortened tableaux and apply the "efficient" rules for selecting pivots. The problem is given in tableau 9.4a on the next page.

TABLEAU 9.4 A

TABLEAU WITH SUBSTITUTE OBJECTIVE FUNCTION:
THE GENERAL CASE.

NAME	!!	X 1	X 2	X 3	X 4	!!	VALUE
S 1	!!	1	-2	③	-1	!!	3
S 2	!!	-2	3	-	-	!!	2
S 3	!!	-3	1	-2	3	!!	-1
S 4	!!	3	-2	1	-3	!!	-1
S 5	!!	-2	1	-3	-2	!!	-6
T	!!	3	-1	-2	1	!!	-
SI	!!	-2	-	-4	-2	!!	-8

The row si is the substitute objective function row, si standing for sum of infeasibilities. This row is obtained by an addition of the violated restrictions, in this case s_3, s_4 and s_5. The rule of the steepest ascent, applied on the substitute objective function, will indicate x_3 as the pivot column; the number -4 being the most negative element in the substitute function. The x_3-column is also the only "preferred" column, indicated by both the specified and the substitute objective function.

Among the pairs of numbers between the value-column and the (x_3) pivot column, three pairs have equal sign. The corresponding quotients are 3 : 3 = 1 for s_1, -1 : -2 = $\frac{1}{2}$ for s_3, and -6 : -3 = 2 for s_5. The other pairs have unequal signs, hence the corresponding rows are not to be considered as pivot row in this stage. The largest quotient of negative numbers is found to be -6 : -3 = 2 for s_5. The smallest quotient of positive numbers is 3 : 3 = 1 for s_1. Of these two quotients, the one for s_1 is the smaller one. The s_1-row will be the pivot-row.

The tableau is updated in the usual way, except for the substitute objection function. This function is re-specified itself, every time when one or more violated restrictions cease to be violated. It is therefore more convenient to simply add the relevant rows.

Our second tableau will now be:

TABLEAU 9.4 B

WE HIT THE OTHER SIDE OF THE FEASIBLE SPACE AREA: NO MORE FLYING THROUGH.

NAME	!!	X 1	X 2	S 1	X 4	!!	VALUE
X 3	!!	0.33	-0.67	0.33	-0.33	!!	1
S 2	!!	-2	3	-	-	!!	2
S 3	!!	-2.33	-0.33	0.67	2.33	!!	1
S 4	!!	2.67	-1.33	-0.33	-2.67	!!	-2
S 5	!!	-1	-1	1	-3	!!	-3
T	!!	3.67	-2.33	0.67	0.33	!!	2

From the tableau, it will be observed, that s_3 has dropped from the list of the violated restrictions. The slack-variable s_3 is still in the basis, but now with a positive value. We could of course have followed the conservative rule, and have taken s_3 as the pivot-row, instead of s_1 as we did. The corresponding value for x_3, the new basic variable, would have been $\frac{1}{2}$, if s_3 had been our pivot row. What have we gained by increasing x_3 still more, until $x_3 = 1$? This can be seen from the previous tableau. On inspection of the x_3 column in that tableau we see that such further increase of x_3, has increased s_5 with $\frac{1}{2} \times 3$ units, and has reduced s_4 with $\frac{1}{2} \times 1$ unit. Then the sum of the negative slacks s_4 and s_5 has increased 1 units, by using the "efficient" rule for choosing the pivot row instead of the "conservative" one. This "extra" is of course a per chance possibility. But it is not quite incidentally. The pivot column x_3 was chosen because -4 was the most negative element in the substitute objective function. This was the sum of such elements in the x_3 column, as corresponded to the negative slacks. Because it was an in absolute value relatively large number, this sum did not change sign after one of the variables dropped from the list of negative slacks. The remainder list still provided a negative sum. The other "extra" possibility of the "efficient" rule is the dropping out of more than one name from the list of negative variables. But this did not materialize.

9.5 The method of artificial variables

This is the method most commonly advocated in literature on the subject. For each restriction, not satisfied by the trivial solution, one defines an artificial activity. This artificial variable in fact is minus the slack of the violated restriction. This variable declares the restriction to be satisfied, but at the price of a huge penalty-element in the objective function.

Alternatively, one first solves the problem of minimizing the sum of the artificial* variables.

This defines a related problem. The related problem is a problem, of which we may for some reason assume that it will have a solution similar to the solution of the original problem. This assumption is in the present case based on the "penalty" elements in the objective function. Any solution to the related problem, with one of the artificial variables being one of the basic variables, will have a huge negative value for the objective function.

When the original problem has no feasible solution (is empty), the artificial variables cannot be eliminated out of the basis of the related problem. Otherwise, the optimum of the related problem will be the optimum of the original problem.

Let us now see, how an L.P.-problem is solved with the method of artificial variables. Consider again the example given in Section 2 of this chapter. The initial tableau for this problem in "ordinary" extended form is already given as tableau 9.2e.

The related problem then is the one given below in tableau 9.5a

TABLEAU 9.5 A

TABLEAU WITH ARTIFICIAL VARIABLES, TO BE EXCHANGED AGAINST NEGATIVE SLACKS.

NAME	!!	X 1	X 2	A 1	A 2	S 1	S 2	S 3	!!	VALUE
S 1	!!	1	1	-	-	1	-	-	!!	6
S 2	!!	-1	-3	(-1)	-	-	1	-	!!	-3
S 3	!!	-	-1	-	(-1)	-	-	1	!!	-2
T	!!	-1	1	100	100	-	-	-	!!	-

The artificial variables are now pivoted against the corresponding slacks. To all practical purposes, this amounts to changing the signs of the rows, representing the violated restrictions. The artificial variables are the negative counterparts of the slacks; they become basic variables, with

---

*cf W. W. Garvin "An Introduction to Linear Programming" [11] pp.39-46 or G.B. Dantzig "Linear Programming and Extensions" [8] pp.94-103.

positive values, instead of the slacks with negative values. After two steps, this automatically results in a feasible solution to the related problem:

TABLEAU 9.5 B

ARTIFICIALLY FEASIBLE STARTING SOLUTION.

NAME	!!	X 1	X 2	A 1	A 2	S 1	S 2	S 3	!!	VALUE
S 1	!!	1	1	-	-	1	-	-	!!	6
A 1	!!	1	(3)	1	-	-	-1	-	!!	3
A 2	!!	-	1	-	1	-	-	-1	!!	2
T	!!	-101	-399	-	-	-	100	100	!!	-500

If the original problem has a feasible solution at all (if it is not empty), the optimum of the related problem will be such, that the artificial variables are not in the basis. Their elimination is achieved by the normal process of simplex operations. In the present example there will be two more tableaux, before a_2 and a_3 will have been eliminated.

TABLEAUX 9.5 C AND D

TWO FURTHER TABLEAUX OF THE SAME PROBLEM.

NAME	!!	X 1	X 2	A 1	A 2	S 1	S 2	S 3	!!	VALUE
S 1	!!	0.67	-	-0.33	-	1	0.33	-	!!	5
X 2	!!	0.33	1	0.33	-	-	-0.33	-	!!	1
A 2	!!	-0.33	-	-0.33	1	-	(0.33)	-1	!!	1
T	!!	32	-	133	-	-	-33	100	!!	-101

NAME	!!	X 1	X 2	A 1	A 2	S 1	S 2	S 3	!!	VALUE
S 1	!!	1	-	-	-1.00	1	-	1.00	!!	4
X 2	!!	-	1	-	1	-	-	-1	!!	2
S 2	!!	-1	-	-1	3	-	1	-3	!!	3
T	!!	-1	-	100	99	-	-	1	!!	-2

Tableau 9.5d corresponds to a feasible solution of the original problem.

If we have tried to obtain a feasible solution to the original problem, by maximising the sum of all negative slacks, while using the "conservative" rule of pivot-row selection we would have obtained the same tableau except for the absence of the

artificial columns. Once the right-hand side is positive the columns which represent artificial variables can be suppressed anyway. If we had used the "efficient" rule for choosing the pivot row, the same tableau would have been obtained in one step. (x_2 as pivot column, against s_3 as pivot row). If we had followed the "conservative" rule of taking the smallest among all positive quotients, we would, to all practical purposes, also have made the same steps. The difference would be that we would not have described them as eliminating artificial variables, but as eliminating negative-valued slacks out of the basis. The pivoting operations would have been numerically the same, except that we would have met positive quotients of negative numbers. The artificial variables have in fact served as operators, for adding the violated restrictions, blown up by a factor 1000, to the objective function. However, it is not too difficult to find an example (e.g. the one in section 9.4), for which several versions of the substitute objective function method will take advantage of the possibility to "fly through" several violated restrictions in one step. With artificial variables this is not possible.

9.6 Non-updating of the substitute objective function

It is not in practice necessary to computationally treat a substitute objective function as a function at all. In particular if the substitute preference function is the sum of the infeasibilities, the substitute preference coefficients may be computed ad hoc by adding the coefficients of the violated restrictions.

Example

Minimise	$3x_1 + 4x_2$	$(= \text{maximise } -3x_1 - 4x_2)$
	$x_1 + x_2 \geq 10$	$(-x_1 - x_2 \leq -10)$
	$x_1 \geq 2x_2 + 1$	$(-x_1 + 2x_2 \leq -1)$
	$x_2 \geq 2$	$(-x_2 \leq -2)$
	$x_1 \leq 8$	
	$(x_1, x_2 \geq 0)$	

The set-up tableau with the first pivot and the second tableau are given here with a substitute objective function row.

TABLEAUX 9.6 A AND 9.6 B

THE SUM-OF-INFEASIBILITIES-ROW IS OBTAINED BY ADDING ALL THE ROWS REFERRING TO VIOLATED RESTRICTIONS.

IN 9.6 A: 1, 2, AND 3.

NAME	X 1	X 2	VALUE
S 1	-1	-1	-10
S 2	-1	2	-1
S 3	-	-1	-2
S 4	①	-	8
T	3	4	-
SI	-2	-	-13

IN 9.6 B: 1 AND 3.

NAME	S 4	X 2	VALUE
S 1	1	-1	-2
S 2	1	2	7
S 3	-	-1	-2
X 1	1	-	8
T	-3	4	-24
SI	1	-2	-4

In the initial tableau the substitute objective function is $si = s_1 + s_2 + s_3$, and we computationally obtain it by adding the corresponding rows of the tableau. If we now look at the second tableau we note that the substitute objective function now is $si = s_1 + s_3$, s_1 and s_3 being the negative-valued slacks, the s_2-restriction being now satisfied. From the s_1 and s_3 rows of the tableau we read the expressions for s_1 and s_3, i.e.

$$s_4 - x_2 + s_1 = -2 \rightarrow s_1 = -s_4 + x_2 - 2$$

$$- x_2 + s_3 = -2 \rightarrow s_3 = x_2 - 2$$

$$si = s_1 + s_3 = -s_4 + 2x_2 - 4$$

Putting this again in implicit form, we obtain (the substitute objective function row).

$$s_4 - 2x_2 + si = -4$$

It is obvious that this row is computationally most easily obtained as the sum of the s_1 and the s_3 rows as they are already in the tableau. Furthermore, if one were to update the old substitute objective function, the result would not be valid, insofar as s_2 is concerned. The reason is that the collection of violated slacks has changed, $si = s_1 + s_2 + s_3$ is no longer the correct substitute objective function. We have no reason to increase the value of s_2, now that this slack has ceased to be negative.

Exercise 9.6a

The following L.P. problem is given:

Minimise $x_1 + x_2$

subject to $3x_1 + 2x_2 \geq 2$

$x_1 \geq x_2 + 1$

$x_2 \geq 2$ $\qquad (x_1, x_2 \geq 0)$

The following should be done:

1. Make a graphical mapping and find the optimal solution by means of graphical analysis.

2. Solve the problem by means of a version of the Simplex Method, using

   (a) full explicit tableaux, including unit vectors. Also write the corresponding systems of equations, with explicitly written variables (carry a sum-count column);

   (b) condensed (shortened) tableaux.

Check that algebraically calculated solution vectors match vertices in the drawn graph.

CHAPTER X

MIXED SYSTEMS, UPPER AND LOWER BOUNDS

10.1 Variables without sign restriction

So far, we have assumed that the so-called "tacit" restrictions $x_1 \geq 0$ (all j) apply.

There is no good reason, other than convenience why this restriction should always apply. Problems in which some variables are "free" variables, or variables "of type absolute", occur and can be solved.

Some textbooks* recommend to represent "free" variables by defining plus or minus the same variable as technically two variables. For example, a problem in two variables where x_1 is not restricted in sign might be written as:

Maximise $\quad x_{1p} - x_{1n} + x_2$

Subject to $\quad 2x_{1p} - 2x_{2n} + x_2 \leq 10$

$$x_2 \leq 20$$

$$-x_{1p} + x_{1n} \leq 2$$

$$x_{1p} - x_{1n} \leq 1$$

$$(x_{1p}, x_n, x_2 \geq 0)$$

We have "split" x_1 into two variables, x_{1p} being the positive side of x_1, and x_{1n} the negative side of x_1.

There is however, no need for this cumbersome approach. It is enough to require that x_1 is a basic variable irrespective of its sign. As x_1 does not have to be positive it can be pivoted into the current solution, if so desired with a negative pivot. And, once x_1 is represented in the Simplex tableau by a row, it is except from the search operations.

---

*cf. W. W. Garvin [11] p.4

The following tableaux may now be developed.

TABLEAU 10.1 A

SET-UP TABLEAU WITH A FREE
VARIABLE, NO SPECIAL FEATURES.

NAME	X 1	X 2	VALUE
S 1	2	1	10
S 2	-	1	20
S 3	-1	-	2
S 4	(1)	-	1
T	-1	-1	-

Both s_3 and s_4 are acceptable as pivot-rows. Since $x_2 = 1$ gives rise to a higher solution value than $x_2 = -2$, we might take the s_4-row. (The code offered on section 12.3 chooses the lowest absolute critical ratio).

TABLEAU 10.1 B

THE FREE VARIABLE X1 HAS
NOW BECOME A BASIC VARIABLE.

	NAME	X 2	S 4	VALUE	
	S 1	(1)	-2	8	
	S 2	1	-	20	
	S 3	-	1	3	
*	X 1	-	1	1	*NOT TO BECOME PIVOT ROW
	T	-1	1	1	

We now enter x_2 into the current solution, by the normal rules of steepest ascent and smallest quotient, and write the third tableau:

TABLEAU 10.1 C

X2 IS NOW ALSO IN THE BASIS.

NAME	!	S 1	S 4	!!	VALUE	
X 2	!	1	-2	!!	8	
S 2	!	-1	2	!!	12	
S 3	!	-	(1)	!!	3	
* X 1	!	-	1	!!	1	*NOT TO BECOME PIVOT ROW.
T	!	1	-1	!!	9	

This time the rule of the steepest ascent gives us s_4 as pivot-column. Excluding x_1 from the search-operation, we find s_3 as the pivot-row and make the step

TABLEAU 10.1 D

S4 HAS DRIVEN OUT S3, NOT X1.

NAME	!	S 1	S 3	!!	VALUE	
X 2	!	1	2	!!	14	
S 2	!	-1	-2	!!	6	
S 4	!	-	1	!!	3	
* X 1	!	-	-1	!!	-2	*ALLOWED TO BECOME NEGATIVE
T	!	1	1	!!	12	

And we find $x_2 = 14$, $x_1 = -2$ as the optimal solution.

Note that it is in this context meaningful to have a negative lower limit on a free variable e.g. $x_1 \geq -2$, written as $-x_1 \leq 2$. There is, however, a more efficient way of handling restrictions of that type (see section 10.4).

10.2 Equations

The most practical way to integrate equations into the Simplex Algorithm is first to make sure that they become binding inequalities, and then to exclude their slack-variables/shadow prices from the list of eligible pivot columns. In terms of

interpretation such "slack variables" are then in fact artificial variables but the difference in computational procedure is not all that much.

One method of ensuring the elimination of the "slack-variables" of equations is to present them as violated restrictions, and then to apply the "conservative" rule in the search operation for a pivot row, i.e. to accept negative pivots in rows representing equations.

Example

Maximise $\tau = x_1$

Subject to $x_1 + 2x_2 + x_3 = 10$

$x_2 \geq x_3$

$x_2 \leq \frac{1}{2}x_3 + 3$

The initial Simplex tableau is now written as follows:

TABLEAU 10.2 A

PRESENT AN EQUATION IN THE FORM OF A VIOLATED INEQUALITY.

NAME	X 1	X 2	X 3		VALUE
S 1	(-1)	-2	-1	=	-12
S 2	-	-1	1	≤	0
S 3	-	1	-0.50	≤	3
T	-1	0	0		-

The = sign in the s_1-row is so far an intention, not a met requirement: The trivial basis is $x_1 = x_2 = x_3 = 0$ and this is not equal to -12.

Quite apart from the equation-status of the s_1-row, the efficient way to solve this problem is by activating the "preferred" x_1-column.

The next tableau therefore is (after re-ordering):

TABLEAU 10.2 B

OPTIMAL TABLEAU, WITH A NEGATIVE-VALUED SHADOWPRICE OF AN EQUATION.

NAME	!	X 2	X 3	S 1 *	!!	VALUE
X 1	!	2	1	-1	!!	12
S 2	!	-1	1	-	!!	0
S 3	!	1	-0.50	-	!!	3
T	!	2	1	-1 *	!!	12

S1 NOT TO BECOME INCOMING VARIABLE.

This tableau is in optimal form, despite the fact that the updated form of the objective function still contains a negative element in the s_1-column.

The objective function row, represents the equation:

$$2x_2 + x_3 - s_1 + \tau = 12$$

or, written in explicit form:

$$\tau = 12 - 2x_2 - x_3 + s_1.$$

Thus, we could increase the value of the objective function i.e. x_1 by one unit for each unit of s_1. But since x_1 already has the value of 12, $x_1 + 2x_2 + x_3$ would become greater than 12 if we did so, contrary to the first equation-requirement. Shadow prices of equations are allowed to be negative-valued in the optimal solution.

10.3 Upper bounds and the two value columns

It is obviously possible to list an upper bound on a variable as a restriction.

In the example in section 10.1 there was an upper bound of 20 on x_1, i.e. we listed the requirement $x_1 \leq 20$ as a restriction.

Restrictions of this type occur fairly often, and some problems could be substantially reduced in size, if all variables were considered to have an upper bound. One could then carry two value columns, one giving the distance of the variables from

zero, i.e. their value, and the other from a specified upper bound.

Example

Maximise $\tau = 2x_1 + x_2 + x_3$

$x_1 \leq 2x_2 + x_3$

$x_1 \leq 10, \quad x_2 \leq 20, \quad x_3 \leq 30$

$x_1 + x_2 - x_3 \leq 20 \qquad (x_1, x_2, x_3 \geq 0)$

We will indicate the slack-variables of the upper limit restrictions with the letter b, b_j being the distance between x_j and its upper limit. We need to store explicitly only the x_j or the b_j-column, not both at the same time. (While referring to upper limits we shall refrain from using matrix notation, as confusion with the elements of the vector $\underline{b}$ might otherwise arise.)

A suitable place to write the upper limits in the tableau, is below the columns which relate to the variables in question, in the $(m+2)^{nd}$ row. The set-up tableau now becomes:

TABLEAU 10.3 A

SET-UP TABLEAU WITH UPPER LIMITS.

NAME	X 1	X 2	X 3	VALUE	DIST.
S 1	(1)	-2	-1	0	1000
S 2	1	1	-1	20	1000
T	-2	-1	-1	-	X
BOUND	10	20	30	X	X

The column entitled "Dist" gives the distances between the values of the basic variables and their upper limits - As no upper limits for slack-variables have been specified, this column so far consists solely of dummy-entries. In the interest of uniform tableau-manipulation, a "fancyhigh" number of 1000 has been entered, but the algorithm does not include upper limits on slack-variables in the search operations.

The rule for the choice of the pivot row is modified. Eligible quotients are ratios between the entries in the value column divided by the corresponding element in the pivot-column as

well as the entries in the distance column and minus any negative elements in the pivot column, (as far as they refer to elements of $\underline{x}$). If the selection of the incoming variable is by the steepest ascent, we first make an ordinary step, developing the following tableau:

TABLEAU 10.3 B

THE SAME PROBLEM AFTER ONE ORDINARY STEP.

NAME	!	S 1	X 2	X 3	!!	VALUE	DIST.
X 1	!	1	[-2]	-1	!!	0	10
S 2	!	-1	3	-	!!	20	1000
T	!	2	-5	-3	!!	0	X
BOUND	!	1000	20	30	!!	X	X

The entry of 10 in the "distance" column for x_1 is the originally specified upper bound of 10, less the value at which x_1 is brought into the basis - zero in this case -. In the upper bounds row we copy in the new pivot column the sum of the entries in the value column and the distance column. (For the s_1 slack-variable, this calculation is not really relevant, as we do not recognize an upper limit on a slack-variable.)

We now find x_2 as incoming variable.

The normal rule of the smallest quotient amended by the inclusion of the absolute value of negative quotients with the upper bounds column, indicates that the upper bound on x_1 is the pivot row. (The pivot has been marked with a square rather than a circle, to avoid confusion with "normal" pivots).

We first write the b_1-row in the tableau. The row which describes the upper bound on x_1 differs from the x_1-row, in the sign, and in the entry in the value column. Anything which increases x_1 decreases the distance from the upper bound on x_1, and vice versa. The signs in the x_1-row have therefore been reversed, except for the "value" and "distance" entries, which have been interchanged. We thereby obtain tableau 10.3c:

TABLEAU 10.3 C

FINDING AN UPPER LIMIT BINDING, WE WRITE THE CORRESPONDING ROW EXPLICITLY.

NAME	!	S 1	X 2	X 3	!!	VALUE	DIST.
B 1	!	1	(2)	1	!!	10	0
S 2	!	-1	3	-	!!	20	1000
T	!	2	-5	-3	!!	0	X
BOUND	!	1000	20	30	!!	X	X

The pivot can now be marked in the "normal" way and we make the step, to obtain Tableau 10.3d

TABLEAU 10.3 D

TABLEAU WITH A BINDING UPPER LIMIT

NAME	!	S 1	B 1	X 3	!!	VALUE	DIST.
X 2	!	-0.50	0.50	0.50	!!	5	15
S 2	!	(0.50)	-1.50	-1.50	!!	5	1015
T	!	-0.50	2.50	-0.50	!!	25	X
BOUND	!	1000	10	30	!!	X	X

The entries in the "distance" column are, in terms of interpretation, the slacks of the upper limit restrictions on the corresponding variables. Amply fulfilled upper limit restrictions are not stored, except when they were specified as "ordinary" restrictions.

We need a somewhat different rule for updating the distances column. The sum of the value of a variable itself, plus the distance from the upper bound should not change. Therefore, when an entry in the value column is increased the corresponding distance from the upper bound is reduced, and vice versa. Hence we reduced, by normal updating rules the s_2-entry from 20 to 5; we therefore increase the corresponding "distance" entry by 15. Thus their sum stays 1020 as it originally was.

The value- and distance column entries in the pivotal row (b_1/x_2 in the example), are dealt slightly different. The 5 for x_2 itself is the critical ratio (as usual). The 15 in the distance column is the difference between the original upper bound of 20, and this figure of 5.

Generally the upper limit on a new incoming variable is obtained by subtracting the value of the incoming variable from its upper limit, as stored in non-updated form.

We may now develop a number of successive tableaux, in which there is no direct reference to a variable which is at its upper limit. If we bring in s_1 as the next incoming variable the rule of the smallest quotient is applied between $15/\frac{1}{2} = 30$ for the upper limit on x_2 (where the denominator is $\frac{1}{2}$ instead of $-\frac{1}{2}$ as the variable which might be driven out is not x_2 but the slack of the upper limit on x_2, and

$5/\frac{1}{2} = 10$ for s_2.

This gives s_2 as the leaving variable, and the next tableau is tableau 10.3e.

TABLEAU 10.3 E

X2 AND ITS LIMIT-DISTANCE STILL ADD TO 20.

NAME	!	S 2	B 1	X 3	!!	VALUE	DIST.
X 2	!	1	-1	[-1]	!!	10	10
S 1	!	2	-3	-3	!!	10	990
T	!	1	1	-2	!!	30	X
BOUND	!	1020	10	30	!!	X	X

The 1020 for s_2 in the upper bounds row is the sum of the entries in the two value columns in the previous tableau 10.3d. For a slack-variable, this is not actually a relevant calculation, but it illustrates the method of calculating the upper limit on a leaving variable.

Now x_3 is the incoming variable. There is in fact only one quotient to which the rule of the smallest quotient may be applied, viz. 10/1 for the upper limit on x_2. (990/3 for the "upper limit" of s_1 does not qualify, because we do not recognise upper limits on slack-variables.)

At this point, we will dispense with further repetition of illustration of this particular type of step, and leave the completion of this example on the lines indicated, to the reader.

However, one point which warrants further discussion concerns the updating by vector-operations only. If a variable drives out its own upper limit distance, the coefficients for the non-basic variables are unaffected. This is on account of the block-

diagonal structure of the block-pivot

$$A_{11} = \left[\begin{array}{c|c} A^*_{11} & \underline{a}_{1j} \\ \hline & 1 \end{array}\right] \qquad (10.3.1)$$

where A^*_{11} is the old block-pivot, before the upper limit became binding, A_{11} being the new block-pivot; $\underline{a}_{1j}$ then is the column of the old A_{12} block which describes the relation between the incoming variable x_j and the old basic variables $\underline{x}^*_j$.

A current tableau extract, relating to these variables is

$$T^* = \begin{array}{c} \\ x^*_1 \\ \\ b_j \end{array} \left[\begin{array}{c|c} A^{*-1}_{11} & A^{*-1}_{11}\,\underline{a}_{1j} \\ \hline & 1 \end{array}\right] \qquad (10.3.2)$$

whereas the inverse of the new block-pivot is:

$$A^{-1}_{11} = \left[\begin{array}{c|c} A^{*-1}_{11} & -A^{*-1}_{11}\,\underline{a}_{1j} \\ \hline & 1 \end{array}\right] \qquad (10.3.3)$$

(see also section 3.11 for the rules of block-inversion).

Obviously, the reverse is true as well, if an upper limit distance b_j becomes incoming variable, and the leaving variable is the corresponding variable itself which moves from its upper limit to zero, the coefficients of the non-basic variables are not affected either.

We illustrate this possibility by solving the same problem again, this time emplying the rule of the highest step as pivot selection rule.

It so happens that this leads to a choice of pivots where all three variables move right to their upper limits. Reference to the similar problem with the explicitly written restrictions illustrates the fact that the upper limit itself is an eligible quotient. This is illustrated in tableau 10.3f.

TABLEAU 10.3 F

THE SAME PROBLEM, WITH
EXPLICITLY WRITTEN LIMITS.

NAME	X 1	X 2	X 3	VALUE
S 1	1	-2	-1	0
S 2	1	1	-1	20
S 3	1	-	-	10
S 4	-	1	-	20
S 5	-	-	(1)	30
T	-2	-1	-1	-

In the set-up tableau itself, there are four eligible pivots, viz:

s_1/x_1 with $\Delta\tau=0$ (actually selected when the rule of the steepest ascent is applied), in the x_1-column,

s_2/x_2 and s_4/x_2 with co-equal ratios of $x_2 = 20$ for each, with $\Delta\tau=20$ in the x_2 column, and

s_5/x_3 with $\Delta\tau=30$ as the only elgible quotient in the x_3 column.

When working with a tableau with a separate upper limits row, we therefore have a further modification of the rule of the smallest quotient.

If no quotient is found, or if the smallest quotient as found exceeds the value of the upper limit on the variable itself, the value of the incoming variable at the next vertex is the upper limit on the incoming variable itself.

This is the case with the pivot which is actually selected: b_3/x_3. As the unity coefficient is not stored, we put the usual marking on the value of the upper limit itself. We now summarize the solution of the problem in the tableau series 10.2g to 10.2j.

TABLEAUX 10.3 G, H AND I

SOLUTION OF THE SAME PROBLEM BY THE
RULE OF THE HIGHEST STEP:
VECTOR-UPDATING ONLY IS REQUIRED.

NAME	X 1	X 2	X 3	VALUE	DIST.
S 1	1	-2	-1	0	1000
S 2	1	1	-1	20	1000
T	-2	-1	-1	-	X
BOUND	10	20	(30)	X	X

NAME	X 1	X 2	B 3	VALUE	DIST.
S 1	1	-2	1	30	970
S 2	1	1	1	50	970
T	-2	-1	1	30	X
BOUND	(10)	20	30	X	X

NAME	B 1	X 2	B 3	VALUE	DIST.
S 1	-1	-2	1	20	980
S 2	-1	1	1	40	980
T	2	-1	1	50	X
BOUND	10	(20)	30	X	X

NAME	B 1	B 2	B 3	VALUE	DIST.
S 1	-1	2	1	60	940
S 2	-1	-1	1	20	1000
T	2	1	1	70	X
BOUND	10	20	30	X	X

To derive from the set-up tableau (10.3g = 10.3a), the successor tableau 10.3h, we adjust the signs of all the entries in the x_3 column (but not of the value of the upper limit!), adjust the two value columns, and leave the rest of the tableau unchanged.

The same applies for the x_1/b_1 column at the next succession, and for the x_2/b_2 column at the transition to the optimal tableau.

Exercise 10.3a

Solve the example problem of this section three times, twice using the "normal" set-up tableau 10.3f and similar successor tableaux, and once using the rule of the steepest ascent, once using the rule of the highest step, the third time by completing the tableau-series started with tableau 10.3A to 10.3e, selecting the incoming variable on the basis of the rule of the steepest ascent.

10.4 Lower bounds: a question of problem formulation

A lower bound on a variable x_j is a minimum value which x_j is required to attain

$$x_j \geq m_j \qquad (10.4.1)$$

Here m_j is the minimum value of x_j which is an exogenous number. We assume that, if such a restriction is specified for $m_j < 0$, it overrides the non-negativity requirement on x_j: there is no point in listing it otherwise.

The most efficient way to handle restrictions of this type, is by re-formulating the problem. We denote the slack-variable of (10.4.1) as

$$y_j = x_j - m_j \qquad (10.4.2)$$

The re-formulation of the problem then consists in substituting $y_j + m_j$ for x_j at the outset, and $x_j - m_j$ for y_j after solving the re-formulated problem. The shadowprice of y_j obviously is that of the lower bound on x_j. We give two examples, one for $m_j > 0$, one for $m_j < 0$:

Maximise $\tau = x_1 + x_2$

Subject to $x_1 + 2x_2 \leq 10$, $(x_1 \geq 0,\ x_2 \geq 2)$

Hence $y_2 = x_2 - 2$, and the LP algorithm is applied to the re-formulated problem

Maximise $\tau = x_1 + y_2 - 2$

Subject to $x_1 + 2y_2 \leq 6 \quad (x_1 \geq 0, y_1 \geq 0)$

That problem is then solved (we find $x_1 = 6$, $y_2 = 0$), and the solution of the original problem is $x_1 = 6$, $x_2 = 2$.

Example for $m_j < 0$

Maximise $\tau = - x_1 + x_2$

Subject to $2x_1 + x_2 \leq 6 \quad (x_1 \geq - 5, x_2 \geq 0)$

$- x_1 + 2x_2 \leq 2$

$y_1 = x_1 + 5$, the re-formulated problem is (substituting $y_1 - 5$ for x_1):

Maximise $\tau = - y_1 + x_2 + 5$

Subject to $2y_1 + x_2 \leq 16$

$-y_1 + 2x_2 \leq - 3$

The solution of this problem is $y_1 = 7$, $x_2 = 2$, therefore $x_1 = 2$, $x_2 = 2$.

To implement the lowerbound facility computationally we need to impose some conventions with respect to storage. It is here assumed that the actual figures m_j are initially stored as an additionally $(m + 3)^d$ row of the tableau, i.e. the last example is written as

TABLEAU 10.4 A
SPECIFIED PROBLEM WITH LIMITS

NAME	X 1	X 2	VALUE	DIS
S 1	2	1	6	X
S 2	-1	2	2	X
T	1	-1	-	X
UB	100	100	X	X
LB	-5	0	X	X

TABLEAU 10.4 B
RE-INTERPRETED PROBLEM

NAME	Y 1	X 2	VALUE	DIS
S 1	2	1	16	X
S 2	-1	2	-3	X
T	1	-1	5	X
UB	105	100	X	X
LB	-5	0	X	X

where the number 100 simply represents a "very high" non-meaningful upperbound.

The operation of re-formulating the problem is sufficiently simple to make it uneconomic to programme it as a separate procedure.

The following lines may be incorporated in any programme instead

```
'FOR'  J := 1  'STEP'  1  'UNTIL N 'DO'

'FOR'  I := 1  'STEP'  1  'UNTIL M+1 'DO'

T[1,N+1]    := T [1, N+1] - T [I,J] * T [M+3, J] ;

'FOR' J := 1 'STEP' 1 'UNTIL' N 'DO' T[M+2, J] := T[M+2 J]
                                                  - T[M+3,J];
```

A re-conversion at the end is not really needed, it is a question of interpreting the outcome.

To obtain a result which relates to the originally specified problem we do not need to change the optimal tableau at all. It is sufficient to interpret the result in terms of the originally specified problem. (See also tableau 10.4c).

TABLEAU 10.4 C

INTERPRETATION OF AN OPTIMUM WITH A LOWER LIMIT

NAME	!	X 2	S 2	!!	VALUE	DIST	
Y 1	!	-2	-1	!!	3	102	(MEANS X 1 = -2)
S 1	!	5	2	!!	10	X	
T	!	1	1	!!	-3	X	
UB	!	100	X	!!	X	X	
LB	!	-5	0	!!	X	X	

The procedure REPO listed below (a result reporting procedure) does just that, and its text needs little discussion here. One point which is however, worth a mention at this point is the possibility to use large negative lower limits to create, in effect free variables. This is possible, but not advisable. Non-zero lower bounds, negative or positive should only be specified if they are meaningfully intended and do not involve non-meaningful outsize numbers. The transformed problem with

the y-variables starts with the trivial basis and if wide-out lower limits are supplied, this solution corresponds to wide-out negative values of the corresponding elements of $\underline{x}$ in the initially specified problem. The trivial solution of the transformed problem will then normally contain large positive as well as large negative entries in the value column i.e. setting these variables at their far-away negative values results in a solution which is not anywhere near the feasible space area.

To start that far away from the feasible space area has two main disadvantages, viz:

1) It requires the activation of the relatively inefficient Phase I part of the algorithm.

2) Manipulation with large numbers may result in loss of numerical precision e.g. if only 6 significant digits are stored 1000 000 + 0.25 is not distinguishable from 1000 000.

The text of the reporting procedure contains a number of references to the machine implementation of the L.P. algorithm. In particular the coding conventions i.e. the recognition of slack-variables and upper limit distances, as distinct from ordinary variables may be unclear to the reader until he has read chapter XII.

The main reason for listing it here rather than in chapter XII is to further emphasize the fact that calculation of the actual values of x_j by means of (10.4.2) is actually part of what is basically a result-output procedure rather than a part of the algorithm.

The text is now listed, as follows:

```
'PROCEDURE' REPO(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL;
'BEGIN' 'INTEGER' I,J,R,K,NS;

  NEWLINE(1);
  WRITETEXT('('SOLUTION REPORT')');
  NEWLINE(3);
  WRITETEXT('('REPORT%ON%THE%ELEMENTS%OF%X')');
  NEWLINE(2);
  WRITETEXT('('FREE%VARIABLES')');
  NEWLINE(1);
```

```
'FOR' J:=1 'STEP' 1 'UNTIL' NAV 'DO' 'BEGIN'
  NEWLINE(1); WRITETEXT('('X')');
  PRINT(K,6,0);
  SPACE(10);
  'IF' ROWL[J]=J 'THEN' PRINT(T[J,N+1],5,2)
  'ELSE' WRITETEXT('('NOT%FOUND')'); 'END';

'IF' NAV = 0 'THEN' WRITETEXT('('%%NONE')');


NEWLINE(2);
WRITETEXT('('BOUNDED%VARIABLES')');
NEWLINE(1);

'FOR' K:=NAV+1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('X')');
  PRINT(K,3,0);
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'IF' COLL[J]=K 'THEN' 'GOTO' NON BASIC;
    'IF' COLL[J]=10000+K 'THEN' 'GOTO' UPPER LIM; 'END';
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWL[I]=K 'THEN' 'GOTO' BASIC;

  WRITETEXT('('ABSENT%WITHOUT%EXPLANATION')');
  'GOTO' END OF XK REPORT;

  NON BASIC:
  WRITETEXT('('%%AT%LOWER%LIMIT%OF%')');
  PRINT(T[M+3,K],5,2);
  'GOTO' END OF XK REPORT;

  UPPER LIM:
  WRITETEXT('('%%AT%UPPER%LIMIT%OF%')');
  PRINT(T[M+2,J]+T[M+3,K],5,2);
  'GOTO' END OF XK REPORT;

  BASIC:
  SPACE(20); PRINT((T[I,N+1]+T[M+3,K]),5,2);

  END OF XK REPORT: 'END';



NEWLINE(3);
WRITETEXT('('SLACK%VARIABLES')');
NEWLINE(1); NS := 0;
'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWL[I]=1000+R 'THEN' 'BEGIN'

  NEWLINE(1); NS:=NS+1;
  WRITETEXT('('S')'); PRINT(R,3,0);
  SPACE(10); PRINT(T[I,N+1],5,2); 'END';

'IF' NS = 0 'THEN' WRITETEXT('('%%NONE')');
```

```
NEWLINE(3);
WRITETEXT('('DUAL%SOLUTION')');
NEWLINE(2);
WRITETEXT('('SHADOWPRICES%OF%EXPLICIT%RESTRICTIONS')');
NEWLINE(1); NS:=0;

'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' COLL[J]=1000+R 'THEN' 'BEGIN'
  NEWLINE(1); NS:=NS+1;
  WRITETEXT('('P')'); PRINT(R,3,0);
  SPACE(10); PRINT(T[M+1,J],5,2); 'END';

'IF' NS = 0 'THEN' WRITETEXT('('%%NONE')');

NEWLINE(3); NS:=0;
WRITETEXT('('SHADOW%PRICES%OF%LOWER%LIMITS')');
'FOR' K:=NAV+1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' COLL[J]=K 'THEN' 'BEGIN'
  NEWLINE(1); NS:=NS+1;
  WRITETEXT('('OF%X')');
  PRINT(K,3,0); WRITETEXT('('%%AT%')');
  PRINT(T[M+3,K],5,2);
  SPACE(10); PRINT(T[M+1,J],5,2); 'END';

'IF' NS = 0 'THEN' WRITETEXT('('%%NONE')');

NEWLINE(3); NS := 0;
WRITETEXT('('SHADOW%PRICES%OF%UPPER%LIMITS')');
'FOR' K:=NAV+1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' COLL[J]=10000+K 'THEN' 'BEGIN'
  NEWLINE(1); NS:=NS+1;
  WRITETEXT('('OF%X')');
  PRINT(K,3,0); WRITETEXT('('%%AT%')');
  PRINT(T[M+2,J]+T[M+3,K],5,2);
  SPACE(10); PRINT(T[M+1,J],5,2); 'END';

'IF' NS = 0 'THEN' WRITETEXT('('%%NONE')');

END OF REPORT: 'END';
```

CHAPTER XI

Duality

11.1 Block-pivoting with inequalities

Consider a partitioned system of inequalities:

$$\begin{aligned} A_{11}\,\underline{x}_1 + A_{12}\,\underline{x}_2 &\leq \underline{b}_1 \\ A_{21}\,\underline{x}_1 + A_{22}\,\underline{x}_2 &\leq \underline{b}_2 \end{aligned} \qquad (11.1.1)$$

where A_{11} is the block-pivot.

We know from Section 5.5 that the recursive product of a series of pivots equals, with possibly an adjustment of the sign, the determinant of a matrix.

Therefore, we may assume that a Simplex solution of an L.P. problem can be expressed (after re-ordering if necessary), by the partitioned system (11.1.1), because non-zero pivots have been found, with A_{11} being square and non-singular.

The first block of restrictions in (11.1.1) is exactly binding, $\underline{x}_2$ is zero-valued, and $\underline{x}_1$ is of the same order as $\underline{b}_1$. Also, $\underline{x}_1$ is related to $\underline{b}_1$ by the inverse of A_{11}.

$$\underline{x}_1 = A_{11}^{-1}\,\underline{b}_1 \qquad (11.1.2)$$

We may write (11.1.1) as a system of equations by adding slack-variables, $\underline{s}_1$ and $\underline{s}_2$ are the two sub-vectors of slack-variables. We also include the target-row, and we obtain

$$\left.\begin{aligned} A_{11}\,x_1 + A_{12}\,\underline{x}_2 + \underline{s}_1 &= \underline{b}_1 \\ A_{12}\,\underline{x}_1 + A_{12}\,\underline{x}_2 + \underline{s}_2 &= \underline{b}_1 \\ -\underline{w}_1'\;\underline{x}_1 - \underline{w}_2'\;\underline{x}_2 + \tau &= 0 \end{aligned}\right) \qquad (8.1.1)$$

We now apply the block-pivoting technique from section 3.11 .

Tableau 11.1a

Symbolic set-up tableau in extensive form

$\underline{x}_1$	$\underline{x}_2$	$\underline{s}_1$	$\underline{s}_2$	τ	=	Value
A_{11}	A_{12}	I				$\underline{b}_1$
A_{21}	A_{22}		I			$\underline{b}_2$
$-\underline{w}_1'$	$-\underline{w}_2'$			1		

Pre-multiply the pivotal block-row by the inverse of the block-pivot.

The pivotal block-row now becomes:

$\underline{x}_1$	$\underline{x}_2$	$\underline{s}_1$	$\underline{s}_2$	=	Value
I	$A_{11}^{-1}A_{12}$	A_{11}^{-1}			$A_{11}^{-1}\underline{b}_1$

Add the pivotal block-row, pre-multiplied by minus the rest of the pivotal block-column, to the remaining block-rows, and the transformed tableau is written as:

Tableau 11.1b

Sybolic updated tableau in extensive form

$\underline{x}_1$	$\underline{x}_2$	$\underline{s}_1$	$\underline{s}_2$	τ	=	Value
I	$A_{11}^{-1}A_{12}$	A_{11}^{-1}				$A_{11}^{-1}\underline{b}_1$
	$A_{22}-A_{21}A_{11}^{-1}A_{12}$	$-A_{21}A_{11}^{-1}$	I			$\underline{b}_2-A_{21}A_{11}^{-1}\underline{b}_1$
	$-\underline{w}_2'+\underline{w}_1'A_{11}^{-1}A_{12}$	$\underline{w}_1'A_{11}^{-1}$		1		$\underline{w}_1'A_{11}^{-1}\underline{b}_1$

We may re-order the tableau, according to the "shortened" version of the tableau, and suppress the unit matrices, writing the names of the (block-) rows at the side of the tableau instead.

Therefore, the initial tableau is written as:

Tableau 11.1c

Symbolic set-up tableau in shortened form

Name	$\underline{x}_1$	$\underline{x}_2$	$\leq$	Value
$\underline{s}_1$	A_{11}	A_{12}		$\underline{b}_1$
$\underline{s}_2$	A_{21}	A_{22}		$\underline{b}_2$
τ	$-\underline{w}_1'$	$-\underline{w}_2'$		0

The updated tableau associated with the block-pivot A_{11} then becomes:

Tableau 11.1d
Symbolic updated tableau in shortened form

Name	$\underline{s}_1$	$\underline{x}_2$	$\leq$	Value
$\underline{x}_1$	A_{11}^{-1}	$A_{11}^{-1}A_{12}$		$A_{11}^{-1}\underline{b}_1$
$\underline{s}_2$	$-A_{21}A_{11}^{-1}$	$A_{22}-A_{21}A_{11}^{-1}A_{12}$		$\underline{b}_2-A_{21}A_{11}^{-1}\underline{b}_1$
τ	$\underline{w}_1'A_{11}^{-1}$	$-\underline{w}_1'+\underline{w}_2'A_{11}^{-1}A_{12}$		$\underline{w}_1'A_{11}^{-1}\underline{b}_1$

Apparently, the updatingrules for a shortened tableau, can also be extended to block-pivoting.

Recall the rules for updating a simplex tableau, when the shortened version of the tableau is used:

Replace the pivotal element by its reciprocal and interchange the names of the pivot-column and the pivot-row.

Multiply the rest of the pivot-row by the reciprocal of the pivot.

Add the updated pivot-row, multiplied by minus the corresponding element in the pivot-column, to the other rows.

Multiply the remainder of the pivot-column by minus the reciprocal of the pivot.

The corresponding rules for block-pivoting are:

Replace the block-pivot by its inverse and interchange the names of the pivotal block-column and pivotal block-row.

This part of the (block) pivoting operation, starting again with the original tableau:

Name	$\underline{x}_1$	$\underline{x}_2$	$\leq$	Value
$\underline{s}_1$	A_{11}	A_{12}		$\underline{b}_1$
$\underline{s}_2$	A_{21}	A_{22}		$\underline{b}_2$
τ	$-\underline{w}_1'$	$-\underline{w}_2'$		0

is completed for the block-pivot itself, by writing its inverse and replacing the names

Name	$\underline{s}_1$
$\underline{x}_1$	A_{11}^{-1}

The equivalent of this part of the block-pivoting operation in ordinary elementwise pivoting is the replacement of the pivot by its reciprocal. Pre-multiply the rest of the pivotal row by the pivot-inverse. The updated pivotal block-row becomes:

Name	$\underline{s}_1$	$\underline{x}_2$	Value
$\underline{x}_1$	A_{11}^{-1}	$A_{11}^{-1}A_{12}$	$A_{11}^{-1}\underline{b}_1$

The equivalent part-operation of this part of the block-pivoting operation in ordinary simplex-operations is the division of all the other elements of the pivot-row by the pivot-element.

Add to the non-pivotal blocks of the tableau, the updated pivotal block-row, pre-multiplied by minus the corresponding block in the pivotal block-column, i.e. $-A_{21}$ and $\underline{w}_1'$

Name		$\underline{x}_2$	Value
$\underline{s}_2$		$A_{22}-A_{21}A_{11}^{-1}A_{12}$	$\underline{b}_2-A_{21}A_{11}^{-1}\underline{b}_1$
τ		$-\underline{w}_2'+\underline{w}_1'A_{11}^{-1}A_{12}$	$\underline{w}_1'A_{11}^{-1}\underline{b}_1$

Post-multiply the remainder of the pivotal block-column by the pivot-inverse, i.e. post-multiply A_{21} by $-A_{11}^{-1}$. We now write the updated tableau, which is now fully up-dated:

Name	$\underline{s}_1$	$\underline{x}_2$	Value
$\underline{x}_1$	A_{11}^{-1}	$A_{11}^{-1}A_{12}$	$A_{11}^{-1}\underline{b}_1$
$\underline{s}_2$	$-A_{21}A_{11}^{-1}$	$A_{22}-A_{21}A_{11}^{-1}A_{12}$	$\underline{b}_2-A_{21}A_{11}^{-1}\underline{b}_1$
τ	$\underline{w}_1'A_{11}^{-1}$	$-\underline{w}_2'+\underline{w}_1'A_{11}^{-1}A_{12}$	$\underline{w}_1'A_{11}^{-1}\underline{b}_1$

That these rules do indeed give us the correct updated tableau is verified by recalling section 8.7 where the equivalent explicitly written tableau (with unit matrices) was obtained by means of substitution.

This includes the special rule for updating the pivot-column, which we first formulated in Section 8.8 . This rule is generalized to block-pivoting as well, in the form of post-multiplication of the pivotal block-column, by the inverse of the block-pivot. The reason why this rule is applicable is the same as in the case of an individual pivot-column. The unit-matrix which links the $\underline{s}_1$ block-row to the $\underline{s}_1$ block-column in the explicitly written tableau serves as an operator.

11.2 The Duality Theorem

The analogy between the updating of (block) pivot-rows and (block) pivot-columns in a simplex tableau (shortened version), suggests a close analogy between the structure of a simplex tableau and its transpose. There is indeed a "transpose" of a linear programming problem. That "transpose" problem is not in terms of $\underline{x}$ and $\underline{s}$.

To give significance to the transpose of any current simplex tableau, we introduce a new set of variables. We write $\underline{u}$ for the vector of specified variables, and $\underline{d}$ for the vector of slacks, and μ for the objective function. The transpose of a current simplex tableau may then be written and interpreted as:

Transposed updated tableau

Name	$\underline{d}_1$	$\underline{u}_2$	$\leq$	Value
$\underline{u}_1$	$[A'_{11}]^{-1}$	$-[A'_{11}]^{-1}A'_{21}$		$[A'_{11}]^{-1}\underline{w}_1$
$\underline{d}_2$	$A'_{12}[A'_{11}]^{-1}$	$A'_{22}-A'_{12}[A'_{11}]^{-1}A'_{21}$		$-\underline{w}_2+A'_{12}[A'_{11}]^{-1}\underline{w}_1$
μ	$\underline{b}'_1[A'_{11}]^{-1}$	$\underline{b}'_2-\underline{b}'_1[A'_{11}]^{-1}A'_{21}$		$\underline{b}'_1[A'_{11}]^{-1}\underline{w}_1$

The corresponding "original" tableau is retraced by applying the same block-pivoting rules to $[A'_{11}]^{-1}$ as block-pivot. That corresponding "original tableau is found to be:

Backward traced transposed tableau

Name	$\underline{u}_1$	$\underline{u}_2$	$\leq$	Value
$\underline{d}_1$	A_{11}'	$-A_{21}'$		$\underline{w}_1$
$\underline{d}_2$	$-A_{12}'$	A_{22}'		$-\underline{w}_2$
μ	$-\underline{b}_1'$	$\underline{b}_2'$		0

This is a curious tableau, in particular because the partitioning changes with each step.

More meaningful than the simple transpose of the current solution is a very much similar problem, where the signs have been adapted. This adaptation of the signs ensures that the "original" problem is independent from the partitioning. A further requirement to be satisfied by this sign-adaptation is preservation of the signs of the value column and objective function row. That way, both the original L.P. problem and the related "transposed" problem are optimal and feasible for the same vertex. These desiderata are met by the dual problem, which is written in a simplex tableau, as follows:

Dual set-up tableau (shortened form)

Name	$\underline{u}_1$	$\underline{u}_2$	$\leq$	Value
$\underline{d}_1$	$-A_{11}'$	$-A_{21}'$		$-\underline{w}_1$
$\underline{d}_2$	$-A_{12}'$	$-A_{22}'$		$-\underline{w}_2$
μ	$\underline{b}_1'$	$\underline{b}_1'$		0

The corresponding updated tableau for this dual problem is:

Updated dual tableau in symbolic form

Name	$\underline{d}_1$	$\underline{u}_2$	$\leq$	Value
$\underline{u}_1$	$-[A_{11}']^{-1}$	$[A_{11}']^{-1}A_{21}'$		$[A_{11}']^{-1}\underline{w}_1$
$\underline{d}_2$	$-A_{12}'[A_{11}']^{-1}$	$-A_{22}'+A_{12}'[A_{11}']^{-1}A_{21}$		$-\underline{w}_2+A_{12}'[A_{11}']^{-1}\underline{w}_1$
μ	$\underline{b}_1'[A_{11}']^{-1}$	$\underline{b}_2'b_1'[A_{11}']^{-1}A_{21}'$		$-\underline{b}_1'[A_{11}']^{-1}\underline{w}_1$

This is the transpose of the current updated tableau with all the signs in the tableau itself changed round, while the two vectors have the same sign. The sign of the solution value is again inverted. Accordingly the linear programming problems

Maximise	and	Maximise
$\tau = \underline{w}'\underline{x}$		$\mu = -\underline{b}'\underline{u}$
Subject to		
$A\,\underline{x} \leq \underline{b} \quad (\underline{x} \geq 0)$		$-A'\underline{u} \leq -\underline{w} \quad (\underline{u} \geq 0)$

are named the primal (or original) and the dual problem, and are optimal and feasible for the same vertex.

This formulation of the Duality Theorem is somewhat different from what is conventional in the literature. See, for example Garvin [11], p.248, or Baumol [2], p.104. The more conventional form of the Duality Theorem states the equivalence of the L.P. problems:

Maximise	and	Minimise
$\tau = \underline{w}'\underline{x}$		$\lambda = \underline{b}'\underline{u}$
Subject to		Subject to
$A\,\underline{x} \leq \underline{b} \quad (\underline{x} \geq 0)$		$A'\,\underline{u} \geq \underline{w} \quad (\underline{u} \geq 0)$

This difference in formulation arises, because minimization problems and restrictions of the $\geq$ type are not directly dealt with in this book. The two formulations are equivalent, except for the value of the objective function. In the conventional formulation of the Duality Theorem the original "primal" problem and the dual problem have the same solution value. In the form in which the Duality Theorem is presented here, primal and dual problems (both being maximization problems), have solution values of the same absolute value, but of opposite sign.

Example

Consider the following "primal" problem

$$\begin{aligned} \text{Maximise} \quad & \tau = 3x_1 - x_2 \\ \text{Subject to} \quad & x_1 + 2x_2 \leq 5 \\ & x_1 \leq 15 \\ & -x_1 + x_2 \leq -1 \\ & (x_1 \geq 0,\ x_2 \geq 0) \end{aligned}$$

The Duality Theorem tells us that we may also find the optimal values of x_1 and x_2 as the shadow prices of the binding restrictions in the following (dual) problem:

$$\begin{aligned} \text{Minimise} \quad & 5u_1 + 15u_2 - u_3 \\ \text{Subject to} \quad & u_1 + u_2 - u_3 \geq 3 \\ & 2u_1 + u_3 \geq -1 \\ & (u_1 \geq 0,\ u_2 \geq 0,\ u_3 \geq 0) \end{aligned}$$

or, equivalently

$$\text{Maximise} \qquad \lambda = -5u_1 - 15u_2 + u_3$$

$$\text{Subject to} \qquad -u_1 - u_2 + u_3 \leq -3$$

$$-2u_1 - u_3 \leq 1$$

$$(u_1 \geq 0,\ u_2 \geq 0,\ u_3 \geq 0)$$

Not only can we "dualize" the initial and the final optimal and feasible tableau, we can do that with each simplex tableau.

The Simplex tableaux contained in summary tableau 11.2a, illustrate an example where the Simplex Algorithm, applied to the primal problem, leads to the optimum in one step. The summary gives the optima of both problems, ordered on the assumption that the corresponding step is also made in the dual problem.

TABLEAU 11.2 A

SUMMARY OF A PRIMAL AND A DUAL PROBLEM.

PRIMAL SET-UP TABLEAU

NAME	X 1	X 2	VALUE
S 1	(1)	2	5
S 2	2	-	15
S 3	-1	1	-1
T	-3	1	-

DUAL SET-UP TABLEAU

NAME	U 1	U 2	U 3	VALUE
D 1	(-1)	-2	1	-3
D 2	-2	-	-1	1
L	5	15	-1	-

PRIMAL OPTIMUM

NAME	S 1	X 2	VALUE
X 1	1	2	5
S 2	-2	-4	5
S 3	1	3	4
T	3	7	15

DUAL OPTIMUM

NAME	D 1	U 2	U 3	VALUE
U 1	-1	2	-1	3
D 2	-2	4	-3	7
L	5	5	4	-15

The dual problem was solved in this case, by choosing a pivot indicated by the primal tableau. In this example that was also a "reasonable" pivot in the context of the dual problem when seen as an L.P. problem in its own right.

This is not always so, as may be seen when solving both problems by the steps which arise from applying the rule of the steepest ascent to the dual - That rule, applied to the d_1-row of the dual problem activates the u_2 column. The resulting steps are shown in the summary tableau 11.2b.

SUMMARY 11.2 B

THE SAME PROBLEM, THE OPTIMA NOW REACHED BY APPLYING THE RULE OF THE STEEPEST ASCENT TO THE DUAL, MAKING CORRESPONDING STEPS IN THE PRIMAL PROBLEM.

PRIMAL SET-UP TABLEAU

NAME	X 1	X 2	VALUE
S 1	1	2	5
S 2	(2)	-	15
S 3	-1	1	-1
T	-3	1	-

DUAL SET-UP TABLEAU

NAME	U 1	U 2	U 3	VALUE
D 1	-1	(-2)	1	-3
D 2	-2	-	-1	1
L	5	15	-1	-

PRIMAL EQUIVALENT TABLEAU

NAME	S 2	X 2	VALUE
S 1	(-0.50)	2	-2.50
X 1	0.50	-	7.50
S 3	0.50	1	6.50
T	1.50	1	22.50

DUAL INTERMEDIATE TABLEAU

NAME	U 1	D 1	U 3	VALUE
U 2	(0.50)	-0.50	-0.50	1.50
D 2	-2	-	-1	1
L	-2.50	7.50	6.50	-22.50

PRIMAL OPTIMUM

NAME	S 1	X 2	VALUE
S 2	-2	4	5
X 1	1	2	5
S 3	1	3	4
T	3	7	15

DUAL OPTIMUM

NAME	U 2	D 1	U 3	VALUE
U 1	2	-1	-1	3
D 2	4	-2	-3	7
T	5	5	4	-15

The second of these two steps is logical in terms of both the primal and the dual problem, but the first does not make much sense as a step in the primal problem at all. In particular, the rule of the smallest quotient is violated and the s_1 - restriction in the first updated tableau in primal form.

This corresponds to a loss of optimal form of the dual problem, i.e. the u_1 entry in the λ-row was originally positive and became negative.

11.3 Application of the Duality Theorem

Some authors (e.g. Kim [23], p.271) mention the possibility to substitute a dual problem with more variables

than restrictions for a primal problem with more restrictions than variables as a cumputational advantage on account of the size of the tableau. The argument is that the use of an explicit simplex tableau requires a unit matrix, of which the order is given by the number of restrictions. However, if the shortened version of the tableau is used, this problem does not arise. In other words this claim arises, because an inefficient algorithm is used in the first place. The Duality Theorem has played an important role in development* of the theory of linear programming. However, many of the currently useful applications of the concept of duality are valid and indeed more useful in the wider realm of convex programming generally.
This includes the important subject of the interpretation of dual variables as imputed prices.

Computational efficiency is, however, relevant with respect to the number of steps. This is so, in particular when one deals with problems which are effectively minimization problems (See also Dantzig [8] , Section 11.2). The term minimization problem here means that a non-negative combination of (non-negative) specified variables is to be minimized. Obviously that minimum can never be less than zero. That solution is attained already in the trivial solution, but this is not a feasible solution. In economic applications this problem usually arises as one of cost minimization, and costs will have to be incurred because certain requirements (restrictions) have to be satisfied. Solution of the problem by its dual maintains optimality, while application of the normal sequence of Phase I and Phase II to the original problem does not maintain the initial optimality.

"Phase I" methods are also generally less efficient than the elementary optimizing algorithm.

It is not even necessary to actually write dual tableaux. The dual Simplex method performs the dual search operations on tableaux written in their "normal" form.

The following problem illustrates the point

$$\text{Minimise} \quad x_1 + x_2 + x_3$$

$$\text{Subject to} \quad \left.\begin{array}{rl} 2x_1 + x_2 & \leq 3 \\ 2x_2 + x_3 & \leq 8 \end{array}\right) \quad (x_1, x_2, x_3 \geq 0)$$

*See for example Tucker [34]

Formulated as a primal maximization problem, this problem becomes:

$$\text{Maximise} \quad -x_1 - x_2 - x_3$$

$$\text{Subject to} \quad \left.\begin{aligned} -2x_1 - x_2 &\leq -3 \\ -2x_2 - x_3 &\leq -8 \end{aligned}\right\} \quad (x_1, x_2, x_3 \geq 0)$$

The dual problem

$$\text{Maximise} \quad \lambda = 3u_1 + 8u_2$$

$$\text{Subject to} \quad \left.\begin{aligned} 2u_1 &\leq 1 \\ u_1 + 2u_2 &\leq 1 \\ u_2 &\leq 1 \end{aligned}\right\} \quad (u_1, u_2 \geq 0)$$

is optimal in one step (see tableau 11.3a).

TABLEAU 11.3 A

A SMALL PRIMAL FEASIBLE PROBLEM, IT IS OPTIMAL IN ONE STEP.

NAME	u 1	u 2	VALUE
D 1	2	-	1
D 2	1	②	1
D 3	-	1	1
T	-3	-8	-

NAME	u 1	D 2	VALUE
D 1	2	-	1
U 2	0.50	0.50	0.50
D 3	-0.50	-0.50	0.50
T	1	4	4

In this example, $x_1 = 0$, $x_2 = 4$ and $x_3 = 0$ is the optimal and feasible primal solution. But to attain this solution, by applying the rules of the steepest ascent and the smallest quotient to the initially infeasible primal problem requires two steps rather than one. The two steps are summarized in tableau 11.3b.

TABLEAU 11.3 B

ILLUSTRATION OF LOSS
OF OPTIMAL FORM.

NAME	X 1	X 2	X 3	VALUE
S 1	-2	(-1)	-	-3
S 2	-	-2	-1	-8
T	1	1	1	-

NAME	X 1	S 1	X 3	VALUE
X 2	2	-1	-	3
S 2	4	(-2)	-1	-2
T	-1	1	1	-3

NAME	X 1	S 2	X 3	VALUE
X 2	-	-0.50	0.50	4
S 1	-2	-0.50	0.50	1
T	1	0.50	0.50	-4

The two systematic factors in the relatively inefficient performance of the Simplex Algorithm on an initially infeasible but optimal (dual feasible) problem are:

a) Loss of optimal form. The shadowprices of x_1 was initially positive, but became negative during the simplex operations.

b) The use of the rule of the smallest quotient in a context where it is not really appropriate. If the "efficient" rule for the choice of the pivot-row had been taken, we would have "flown" through the s_1 restriction and reached the solution in one step. But, as discussed in section 9.2, the "efficient" rule for the choice of the pivot-row has certain definite drawbacks. The possibility to interchange the concepts of "optimal" and "dual feasible" has its computational advantages after all.

The following suggested modification of the rule for choosing the pivot column avoids unnecessary reductions in the value of

the specified objective function. When there is no feasible solution, we may first of all give preference to columns which promise an increase in a substitute objection function, e.g. the sum of all slacks of violated restrictions, as well as in the value of the specified objective function. When no such columns are available the rule of the steepest ascent may be replaced by the criterion of the dual ratio. We will take the column which promises the greatest increase in the value of the substitute objective function per unit of reduction in the value of the specified objection function. If the substitute objection function is the slack of just one violated restriction, this rule can amount to applying the rule of the smallest quotient to the dual. That quotient then is the reduction in the value of the specified objective function, per unit of increase of the substitute objective function. This full analogy with the rule of the smallest quotient applied to the dual problem arises whenever the restriction which provides the substitute objective function becomes binding in one step. This is the case in the illustration-example contained in tableau 11.3c.

TABLEAU 11.3 C

ILLUSTRATION OF THE DUAL RATIO.

NAME	X 1	X 2	X 3	VALUE
S 1	1	-	-	1
S 2	-5	(-2)	-3	-10
T	8	1	2	-
RATIO	-1.60	-0.50	-0.67	

NAME	U 1	U 2	VALUE	RATIO
D 1	-1	5	8	1.60
D 2	-	(2)	1	0.50
D 3	-	3	2	0.67
T	1	-10	-	

NAME	X 1	S 2	X 3	VALUE
S 1	1	-	-	1
X 2	2.50	-0.50	1.50	5
T	5.50	0.50	0.50	-5

NAME	U 1	D 2	VALUE
D 1	-1	-2.50	5.50
U 2	-	0.50	0.50
D 3	-	-1.50	0.50
T	1	5	5

11.4 The Dual Ratio and Phase I Column Selection in the presence of several violated restrictions

If there are no zeros in the updated form of the objective function row, and no pseudo-zeros either, the dual ratio is also a useful additional criterion for selecting columns during

Phase I of the primal problem. It is here assumed that this is in addition to the "efficient" rule of row-selection, and the rule of the highest step as column-selection criterion. This additional criterion is useful for two reasons. Firstly the dual ratio is the ratio between the loss in specified objective function value and the corresponding gain in (a facet of) the substitute objective function i.e. the value of the slack variable in question. Secondly, simply because the denominator is the tentative pivot, a ratio criterion has a bias against small pivots; as is illustrated in tableau 11.4a

TABLEAU 11.4 A

AVOIDANCE OF SMALL PIVOTS
BY THE USE OF THE DUAL RATIO.

NAME	X 1	X 2	VALUE
S 1	-0.01 (-1)		-1
S 2	-	0.90	-1
T	1	1	-

We may think of this tableau as being an extract or, alternatively, it represents an empty problem, which should not be mishandled either. The rule of the highest step, applied to the sum of the infeasibilities as substitute objective function, indicates the s_1/x_1 cell as pivot, with a figure of -0.01.

Having tentatively indicated this as a possible pivot, we now investigate other columns, while imposing the following modifications on both the row and column-selection rules

a. A violated restriction which is already tentatively indicated as pivotal row, is "badname" and flying through it is allowed only with an incoming variable which also increases the specified objective function. (If a "preferred" column is actually selected in this way there is no longer a "badname".) In potential incoming variables, columns which increase the substitute objective function at the cost of a reduction in the value of the specified objective function, eligible pivotal rows are:

   . equations
   . already non-negative-valued variables
   . the "badname" negative slack-variable

other negative slack-variables if the restriction cannot be reached by that incoming variable - the "efficient" rule of row-selection being again applicable -. (In preferred columns which indicate an increase in the specified and the substitute objective function, negative pivots are only eligible in rows representing equations.)

b. Once we establish the fact that a pivot in such a "badname" row is permissable, the dual ratio criterion has priority over the rule of the highest step. Hence the s_1/x_2 entry becomes pivot instead of the s_1/x_1 cell irrespective of the fact that the rule of the highest step would favour the s_1/x_1 cell.

That situation may change if the "badname" status disappears for one of the following reasons:

b1. A "preferred" column is activated.

b2. The "efficient" rule of row-selection, applied to a column in which no negative entry for the "badname" restriction occurs, indicates a different violated restriction as the pivotal row and the rule of the highest step (applied to the substitute objective function) indicates that the new pivot is preferable to the old one.

The following implication of rule a) is stated here separately, mainly in order to discuss its significance.

c. Columns which have been identified as contradicting the dual ratio between the objective function and the badname-row become ineligible as incoming variable columns, notwithstanding the presence of other possible pivots in such columns.

The rationale of this last point may be illustrated by the example given in tableau 11.4b below

TABLEAU 11.4 B

X2 IS NOT A SUITABLE COLUMN.

NAME	X 1	X 2	VALUE
S 1	-1	-0.20	-1
S 2	-2	-0.01	-1
T	1	1	-

We assume that the s_1/x_1 cell has been tentatively selected as pivot, at the stage of scanning the x_1 column.

The entry of -0.20 in the s_1/x_2-cell contradicts the criterion of the dual ratio between the updated form of the objective function row and the s_1-row. However, throwing only that one pivot in the x_2-column out, does not put this problem right; the -0.01 entry in the s_2/x_2 cell is no better pivot. Yet if we use the sum of the two negative slack-variables as substitute objective function and then apply the rule of the highest step, we find that the s_2/x_2 cell comes out with $s_1+s_2 = 99 + 0$ and is the highest. The only reasonable practical way of avoiding the kind of step is to disqualify (as far as this vertex is concerned) x_2 as incoming variable altogether.

Exercise

Complete the problem tackled in section 9.4.

11.5 Dual Degeneracy

The significance of the term dual degeneracy will be obvious: zeros in the objective function row. If there are a whole series of exact zeros in the objective function row, the criterion of the dual ratio ceases to be an effective criterion of choice between one column and another. Yet dual degeneracy occurs quite often, if anything it is more rather than less common than primal degeneracy.

The criterion of the dual ratio may, however happen to hit a zero in the objective function row and indicate a small pivot.

Example

Tableau 11.5a

Dual degeneracy: it may lead to a pseudo-zero pivot

Name	x_1	x_2	Value
s_1	-1	-0.00	-1
s_2	1	2	2
	1	0	-

Assuming that the -0.00 entry is not actually a zero, the rules indicated in the previous section would lead to the selection of the s_1/x_2 entry as pivot. Obviously something should be done

to avoid that.

As far as selection between several columns with zero entries in the objective function row is concerned, perturbation of the objective function row on similar lines as indicated in Section 8.10 for primal degeneracy is likely to be an effective remedy.

The above example indicates that this needs to be supplemented by a direct restriction on the selection of a small pivot.

The way in which we introduced the criterion of the dual ratio only as a supplementary criterion which is considered when a pivot has already been tentatively selected makes this possible. We refuse to change from one negative Phase I pivot to another similar one in the same row, if the latter is an order of magnitude smaller.

The code offered in section 12.3 puts this requirement as the ratio between the two pivots not being less than 0.1.

CHAPTER XII

LINEAR PROGRAMMING ON THE COMPUTER

12.1 Name-codes and namelists

We need some means to relate rows and columns in Simplex tableaux to their significance in terms of equations, variables etc.

There are two main reasons for this requirement. The user needs to know the significance of the end result, and the machine must be programmed to distinguish equations from inequalities, non-negative variables from variables without sign restriction, distances from upper bounds from "ordinary" variables etc.

Reference to alphanumeric names like x_2, s_4, u_3 etc is not very practical in a programmed computer procedure. The use of an integer number as a namecode is much more practical, at least for purposes of internal machine-use, output of results is a different matter.

There clearly are two lists of names in any L.P. problem. They are the list of the basic variables which are associated with the rows of the tableau, and the list of non-negativity restrictions which are associated with the columns of the tableau. In an explicit tableau the columns include unit vectors for basic variables, in a shortened tableau the list of column-names is the list of binding non-negativity restrictions. Tableau 12.1a below gives the usual presentation of the optimum of an example from Chapter 8, tableau 12.1b gives a completely numerical presentation of the same optimal tableau.

TABLEAU 12.1 A
THE USUAL ALPHANUMERICAL PRESENTATION OF A SHORTENED TABLEEAU.

NAME	X 1	S 1	S 2	VALUE
X 2	0.33	-	0.33	33.33
X 3	0.67	1	-0.33	166.67
T	0.33	1	1.33	333.33

TABLEAU 12.1 B
A NUMERICALLY CODED TABLEAU.

	1	1001	1002	
2	0.33	-	0.33	33.33
3	0.67	1	-0.33	166.67
	0.33	1	1.33	333.33

Besides the obvious method of distinguishing individual elements of vectors by their indices, we need some other device to distinguish between classes of variables, i.e. the vectors to which they belong. The two most suitable devices appear to be the plus or minus sign, and the enlargement. In the above example an enlargement of 1000 has been used to indicate row-names i.e. slack-variables. Problems with more than 1000 variables will normally have a partitional structure and a special algorithm will be used in any case. Special algorithms for partitioned problems will obviously have their own system of name-codes where the name-codes refer to the partitioning. An alternative method of coding, might be the one used in tableau 12.1c, using positive and negative numbers.

TABLEAU 12.1 C

USE OF THE -SIGN AS QUALIFIER.

	1	-1	-2	
2	0.33	-	0.33	33.33
3	0.67	1	-0.33	166.67
	0.33	1	1.33	333.33

Above, the namelists have been written as an extra row and column of the tableau. There are, however, some fairly convincing reasons for declaring namelists as separate arrays in their own right. (Or dimensions, the Fortran Programmer would say.) First of all, these separate arrays can be indicated with names which convey their significance to the human reader of the programme text. Thus, an instruction which is preceded by the condition.

```
'IF' ROWLST [I] > NAV 'THEN'
```

clearly refers to elements of the list of rownames.

There are also more strictly computational arguments. Name-codes are integer variables and some machines use only one address for an integer variable and two for a "real" i.e. possibly fractional valued variable. Also, some computer languages e.g. Fortran cannot accommodate rows with a zero or negative row-index or columns with a zero or negative column-index. Several vectors, e.g. value column, upper bounds vector etc. also have to be fitted in with the tableau, and if they all have to be allocated an index in excess of the normal size of the tableau, it becomes somewhat confusing. No useful

purpose is served with reserving space for the namelists as well in the tableau itself.

12.2 Ordering of the tableau

The choice of pivots on the indication of the numerical content of the tableau usually results in a random ordering of the tableau. If the "original" simplex method is used only the rows lose ordering, the columns stay in place. If the shortened version of the tableau is used, the columns also get into a random non-ordering.

Re-ordering can be done on the indication of the numerical values of namecodes, i.e. the inappropriately ordered tableau 12.2a (below) may be re-ordered on the indication of the name-codes, to become tableau 12.1b.

TABLEAU 12.2 A

A 'DIS-ORDERED' TABLEAU

	1002	1	1001		
3	-0.33	0.67	1		166.67
2	0.33	0.33	-		33.33
	1.33	0.33	1		333.33

In order to be able to (re) order in a meaningful way it is necessary to impose certain conventions for ordered tableaux and namelists. I suggest the following rules:

(a) Positive names to be placed before negative names

(b) Names with low absolute value to be placed before names with higher absolute value.

This ensures ordering according to index, and places names with enlargements at the end of the list. In fact, no use of negative name-codes is made in this chapter, and although negative name-codes are used in Chapter 16, with quadratic programming, a different solution for the ordering problem is used there. The following programmed procedure which incorporates these rules is however offered here.

```
'PROCEDURE' ORDR(T,M,N,ER,RH,ROWLST,COLLST);
'ARRAY' T; 'INTEGER' M,N,ER,RH; 'INTEGER' 'ARRAY' ROWLST,COLLST;
'BEGIN' 'INTEGER' FIRST,LAST,I,J,NAME; 'REAL' NUM;
'BOOLEAN' FINISHED;
  'COMMENT'
  ORDERING PROCEDURE FOR A TABLEAU AS USED FOR MOST SIMPLEX
  ALGORITHMS FOR MATHEMATICAL PROGRAMMING.
  THE ROWS ARE ORDERED ACCORDING TO THE NAMES IN THE ROWLIST,
  AND THE COLULMNS ACCORDING TO THE NAMES IN THE COLUMN-LIST.
  POSITIVE NAMES ARE PLACED BEFORE NEGATIVE NAMES.
  WITHIN EACH GROUP OF NAMES WITH HOMOGENEOUS SIGNS,
  ROWS, AS WELL AS COLUMNS WITH NAMES OF LOW ABSOLUTE VALUE
  ARE PLACED BEFORE THOSE WITH NAMES OF HIGHER ABSOLUTE VALUE.

  THE TABLEAU IS ASSUMED TO CONTAIN M PROPER ROWS, I.E. RES-
  TRICTIONS, N PROPER COLUMNS, I.E. VARIABLES, AND EXTRA
  COLUMNS ON THE RIGHT OP LEFT-HANDSIDE, E.G. VALUE COLUMN,
  AND EXTRA ROWS ON THE TOP OR BOTTOM, I.E. TARGET-ROW,
  SUBSTITUTE OBJECTIVE FUNCTION, UPPER BOUNDS ROW, ETC.
  THE NUMBER OF EXTRA COLUMNS IS INDICATED BY THE NUMBER RH.
  IF THE EXTRA COLUMNS ARE ON THE RIGHTHAND SIDE, RH IS
  POSITIVE, A NEGATIVE VALUE OF RH INDICATES EXTRA COLUMNS
  ON THE LEFTHAND-SIDE,
  THEIR NUMEER IS THE ABSOLUTE VALUE OF THE PARAMETER RH.
  THE PARAMETER ER INDICATES THE NUMBER OF EXTRA ROWS,
  IF ER IS POSITIVE THEY ARE AT THE END, IF ER IS NEGATIVE,
  THEY ARE BEFORE THE TABLEAU ITSELF, I.E. AT THE TOP;

  START:   FINISHED := 'TRUE';

  FIRST:=1; LAST:=N;
  'IF' RH > 0 'THEN' LAST:=LAST+RH 'ELSE' FIRST:=FIRST+RH;

  'FOR' I:=2 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    'IF' ROWLST[I] < ROWLST[I-1] 'AND' ROWLST[I] > 0
    'THEN' 'GOTO' REARRANGE ROWS;
    'IF' ROWLST[I] > ROWLST[I-1] 'AND' ROWLST[I-1] < 0
     THEN' 'GOTO' REARRANGE ROWS;
    'GOTO' END OF ROW ORDERING LOOP;
    REARRANGE ROWS:   FINISHED := 'FALSE';
    'FOR' J:=FIRST 'STEP' 1 'UNTIL' LAST 'DO' 'BEGIN'
      NUM:=T[I-1,J]; T[I-1,J]:=T[I,J]; T[I,J]:=NUM; 'END';
   NAME:=ROWLST[I-1]; ROWLST[I-1]:=ROWLST[I]; ROWLST[I]:=NAME;
   END OF ROW ORDERING LOOP:   'END';
   FIRST:=1; LAST:=M;
   'IF' ER > 0 'THEN' LAST:=LAST+ER 'ELSE' FIRST:=FIRST+ER;
```

```
'FOR' J:=2 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  'IF' COLLST[J] < COLLST[J-1] 'AND' COLLST[J] > 0
  'THEN' 'GOTO' REARRANGE COLUMNS;
  'IF' COLLST[J] > COLLST[J-1] 'AND' COLLST[J-1] < 0
  'THEN' 'GOTO' REARRANGE COLUMNS;
  'GOTO' END OF  COLUMN ORDERING LOOP;
  REARRANGE COLUMNS:   FINISHED := 'FALSE';
  'FOR' I:=FIRST 'STEP' 1 'UNTIL' LAST 'DO' 'BEGIN'
    NUM:=T[I,J-1]; T[I,J-1]:=T[I,J]; T[I,J]:=NUM; 'END';
  NAME:=COLLST[J-1]; COLLST[J-1]:=COLLST[J]; COLLST[J]:=NAME;
  END OF COLUMN ORDERING LOOP: 'END';

'IF' 'NOT' FINISHED  'THEN' 'GOTO' START;

END OF ORDR: 'END';
```

If the tableau is at all large, the ordering method which is used by the procedure ORDR is not very efficient.

The procedure interchanges only adjoining names and their associated vectors. Whether the new positions are correctly ordered is then tested in the next ordering loop. The obvious implication of this approach is that large numbers of permutations may be made which do not so far bring any name in its correct eventual position.

Considering the following 4-element list:

1002, 1001, 4, 3.

The eventual ordering of this list is 3, 4, 1001, 1002, in the following stages:

1002,	1001,	4,	3	interchange	1002	and	1001
1001,	1002,	4,	3	interchange	1002	and	4
1001,	4,	1002,	3	interchange	1002*	and	3

1001,	4,	3,	1002	interchange	1001	and	4
4,	1001,	3,	1002	interchange	1001*	and	3
4,	3,	1001,	1002	interchange	4*	and	3*

3,	4,	1001,	1002,	final ordering

Permutations which bring a name in its actually correct position have been marked with an asterisk after the namecode in question.

The ordering of a large list may involve many permutations which don't bring any name in the position of its final ordering.

For large lists this problem tends to be aggravated, the "ineffective" permutations becoming much more numerous, even relative to the "effective" ones.

As far as the logic of the ordering problem is concerned, this is more or less unavoidable, because the final "correct" ordering is not initially known.

What is, however, possible, is to delay re-ordering of the tableau-matrix until the ordering of the namelists is known. It is after all quite a bit cheaper to interchange 2 name-codes than to interchange 2 rows of a large tableau:

ORDR can in fact be used to order only a namelist. One just lists one of the two-order parameters as being zero, i.e.

ORDR (M, 0, 0, 0, ROWLIST, COLLST)

only accesses the first zero columns of the tableau-matrix, and the first zero elements of the list of column-names. This effectively means re-ordering the list of row-names. The tableau matrix itself, and the list of column names is not accessed.

The following procedure makes use of this device to order the tableau in two stages:

```
'PROCEDURE' ORDL(T,M,N,ER,RH,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,ER,RH; 'INTEGER' 'ARRAY' ROWL,COLL;

'BEGIN' 'INTEGER' I,J,R,K; 'REAL' NUM;

  'PROCEDURE' ORDR(T,M,N,ER,RH,ROWL,COLL);
  'ARRAY' T; 'INTEGER' M,N,ER,RH;
  'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

  'INTEGER' 'ARRAY' DUP ROWL[1:M], DUP COLL[1:N];

  'COMMENT'
  ORDERING PROCEDURE FOR A LARGE TABLEAU.
  THE TABLEAU IS ORDERED IN TWO STAGES.
  FIRST THE NAMELISTS ARE COPIED INTO THE DUPLICATE LISTS,
  WHICH ARE THEN ORDERED BY CALLING ORDR, WITH ONE OF THE
  ORDER-PARAMETERS SET AT ZERO.
  THEREAFTER, THE TABLEAU ITSELF IS ORDERED, ON INDICATION
  OF THE DIFFERENCES BETWEEN THE ORIGINAL AND THE REORDERED
  NAMELISTS.
  FINALLY, THE REORDERED NAMELISTS ARE COPIED TO THE ORI-
  GINAL ONES.
  ;
```

```
ORDER ROWLIST:
'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO' DUP ROWL[R]:=ROWL[R];
ORDR(T,M,0,0,0,DUP ROWL,COLL);

ORDER COLUMN LIST:
'FOR' K:=1 'STEP' 1 'UNTIL' N 'DO' DUP COLL[K]:=COLL[K];
ORDR(T,0,N,0,0,ROWL,DUP COLL);

ORDER ROWS:
'FOR' R:=1 'STEP' 1 'UNTIL' M-1 'DO'  'BEGIN'
  'FOR' I:=R+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' DUP ROWL[R]=ROWL[I] 'AND' 'NOT' ROWL[R]=DUP ROWL[R]
  'THEN' 'BEGIN'
    ROWL[I]:=ROWL[R]; ROWL[R]:=DUP ROWL[R];
    'FOR' J:=1 'STEP' 1 'UNTIL' N+RH 'DO' 'BEGIN'
      NUM:=T[R,J]; T[R,J]:=T[I,J]; T[I,J]:=NUM; 'END';
    'GOTO' NEXT ROW; 'END';
  NEXT ROW: 'END';

ORDER COLUMNS:
'FOR' K:=1 'STEP' 1 'UNTIL' N-1 'DO' 'BEGIN'
  'FOR' J:=K+1 'STEP' 1 'UNTIL' N 'DO'
  'IF' DUP COLL[K]=COLL[J] 'AND' 'NOT' COLL[K]=DUP COLL[K]
  'THEN' 'BEGIN'
    COLL[J]:=COLL[K]; COLL[K]:=DUP COLL[K];
    'FOR' I:=1 'STEP' 1 'UNTIL' M+ER 'DO' 'BEGIN'
      NUM:=T[I,K]; T[I,K]:=T[I,J]; T[I,J]:=NUM; 'END';
    'GOTO' NEXT COLUMN; 'END';
  NEXT COLUMN: 'END';

END OF ORDL: 'END';
```

12.3 Commented text of a linear programming procedure

The programmed procedure offered in this section assumes a mixed system as discussed in Chapter X. It contains two calls to the ordering procedure discussed in the previous section. Of these, the call just below the label OUT is strictly for the purpose of orderly presentation of the end result. The first call to the ordering procedure, following the label ORDER, is part of the linear programming algorithm. The reason is that random ordering may already arise during the preliminary inversion phase. As may be seen from its listing below there are two fairly distinct phases in the algorithm. One phase consists in making the variables without sign-restrictions into basic variables irrespective of their sign, the other phase is the linear programming algorithm proper.

The search operation for a pivot row following the label RETURN IN INVERSION is comparable with the similar search operation which occurs in ordinary matrix inversion. The phase of an LP algorithm which deals with the systematic search for a feasible solution is conventionally indicated as "Phase I". In this algorithm, "Phase I" is preceded by a phase which deals with entering "free" variables into the basis. That phase "enter variables without sign restriction" could therefore be called "Phase 0".

The search operations which control this phase are sufficiently different from the ones in the proper LP algorithm to justify keeping them completely separate. The actual updating of the tableau, including the administration of the namelists is however common for all phases. The return to "Phase 0" in order to find a pivot row to match the next "free" variable is controlled by the logical variable INVERTED.

Just as in ordinary matrix inversion (compare Sections 3.10 and 3.14), the pivot may be found elsewhere than on the main diagonal. Or no non-zero pivot may be found at all, if the rank of the left-hand block column is lower than the number of free variables. In that case the "unbounded" loop of Phase 0 will be activated. At the end of Phase 0, the tableau is re-ordered. The implication of this arrangement is obviously that free variables must be placed before bounded variables, i.e. they are recognisable as the ones with the lowest indices.

The re-ordering will therefore put the rows which refer to variables without sign restriction at the top of the tableau. The search operations for a pivot row during the LP algorithm proper then run from the row with index NAV+1 onwards.

The two phases of the normal simplex operations, i.e. Phase I for finding a feasible solution and Phase II for finding the optimum are almost completely integrated.

The distinct "Phase I" part of the programme consists of:

* Finding whether a feasible solution exists, and setting the logical variable FEASIBLE accordingly. Technically, Phase I is re-entered at every step, but this becomes trivial, once a feasible solution exists.

* Setting the objective function coefficient for each column. During Phase I, this is done as follows: In the first instance, the substitute objective function is the sum of all infeasibilities (negative stacks). If the current basis is not feasible, no column qualifies as incoming variable, unless it promises a positive ascent of this substitute objective function.

Columns which pass this test are then loaded as "preferred" or "non-preferred" columns. Preferred columns are those which promise an increase in both the specified and the substitute objective function, and the actually used preference coefficient is the sum of the two objective functions.

Non-preferred columns are those which promise an increase in the substitute objective function, but only at the cost of a loss in specified objection function value. Preference for the preferred columns is made effective by scaling the preference coefficient of non-preferred columns down by a factor 1000.

If the current basis is feasible, the objective function obviously is specified objective function. Control then passes to the search operations of the optimizing algorithm.

In principle all columns - other than those referring to equations - are scanned for potential pivots, going through the tableau from left to right. The following rules apply:

* The rule of the smallest quotient, applied to positive pivots and negative ones in connection with equations. This row-selection rule applies to columns which indicate a positive non-zero ascent in the specified preference direction (= a negative non-zero entry in the $m+1^{th}$ row of the tableau).

* The "efficient" rule of row-selection (see section 9.2). When this rule applies we are dealing with a non-preferred Phase I column. The acceptable minimum value of a negative pivot in an inequality-row is 1/3 of the average of all the negative entries in the column in question which are associated with violated restrictions (including equations). (There is also a minimum absolute value of 0.000001 for all pivots.)

* The rule of the highest step with, for violated restrictions, modifications as discussed in sections 11.4 and 11.5. When the current basis is feasible the rule of the highest step works as a column-selection rule, but in Phase I this is only true for the choice between different preferred columns.

Once it becomes necessary to find the pivot in a non-preferred column, the rule of the highest step effectively becomes a row-selection rule.

Note, however, that the dual ratio condition is enforced only partially, i.e. going from left to right. If a shift to a different violated restriction actually takes place, this may well be one for which the dual ratio is in fact violated in columns further to the left than the one which is currently being scanned.

The other complicating feature which arises concerns the upper limits on specified variables and the fact that negative entries in the tableau "stand in" for positive potential pivots in relation to upper limits. The procedure follows section 10.3 fairly precisely, but it does in practice complicate the search operations quite noticeably.

The procedure-text is now listed, as follows:

```
'PROCEDURE' LINP(T,M,N,NEQ,NAV,ROWLST,COLLST,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST;

'BEGIN' 'INTEGER' I,J,R,TRYR,K,COLN,ROWN,TRYN,N OF NC;
'REAL' ASC,HIG,QUO,TQUO,PIV,NUM,COP,VNEV,DUAL RATIO,
N ASC;
'BOOLEAN' INVERTED,FEASIBLE,UPPERBOUND;

'PROCEDURE' ORDL(T,M,N,ER,RH,ROWLST,COLLST);
'ARRAY' T; 'INTEGER' M,N,ER,RH;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';
```

```
'COMMENT' LINEAR PROGRAMMING PROCEDURE.
M RESTRICTIONS AND N VARIABLES.

THE PROCEDURE ACCOMODATES A MIXED SYSTEM,
WITH UPPER BOUNDS ON ALL NON-NEGATIVE VARIABLES.

THE FIRST NEQ ROWS REFER TO EQUATIONS,
THE REST TO INEQUALITIES.
EQUATIONS SHOULD BE PRESENTED WITH A NON-POSITIVE CONSTANT,
IF NECESSARY BY CHANGING THE SIGN OF ALL ENTRIES IN A ROW.
IF THIS IS NOT DONE, IT WILL BE DONE BY THE PROGRAMME,
WITHOUT FURTHER WARNING.

THE FIRST NAV COLLUMNS REFER TO VARIABLES FOR WHICH
THE NON-NEGATIVITY RESTRICTION DOES NOT APPLY, WHILE IT DOES
APPLY TO THE REMAINING N-NAV COLUMNS.

ONE SHOULD PREVIOUSLY DECLARE THE ARRAYS, T, THE TABLEAU,
AS WELL AS THE INTEGER ARRAYS ROWLST, THE LIST OF ROWNAMES,
AND COLLST, THE LIST OF COLUMN-NAMES.
ONE SHOULD SUPPLY THE TABLEAU, BUT NAMES ARE GENERATED
BY LINP.
THE TABLEAU SHOULD BE SUPPLIED IN LESS THAN OR EQUAL TO FORM,
WITH THE VALUE COLUMN AS THE LAST COLUMN, INDEX N+1,
AND THE TARGET-ROW AS THE LAST ROW, INDEX M+1.

THE TABLEAU-DECLARATION SHOULD RESERVE AN ADDITIONAL N+2 TH
COLUMN FOR THE UPDATED FORM OF UPPER BOUNDS, AND AN M+2 TH
ROW FOR THE NON-UPDATED FORM OF THE UPPER BOUNDS-VECTOR.
THE USER SHOULD SUPPLY THE NUMERICAL CONTENT OF THE UPPER
BOUNDS ROW-VECTOR, THE UPPER BOUNDS TO BE PUT ON THE
SPECIFIED NON-NEGATIVE VARIABLES.
TO ACCOMODATE LARGE UPPER BOUNDS FOR VARIABLES WHERE NO
MEANINGFUL UPPER BOUND IS INTENDED, A ZERO MAY BE SUPPLIED
INSTEAD, AND THE PROGRAMME WILL SUBSTITUTE A MILLION FOR IT.
VARIABLES WITHOUT SIGN RESTRICTION DO     N O T     HAVE
UPPER BOUNDS, BUT DUMMY-NUMBERS HAVE TO BE PUT FOR THEM.

THERE ARE TWO NAMELISTS, WHICH ARE FILLED BY THE PROCEDURE.
THE SPECIFIED ECONOMIC VARIABLES HAVE NAME-CODES EQUAL TO
THEIR INDICES,  I.E. 1 TO N,
AND THE SLACKS OF THE INEQUALITIES HAVE NAME-CODES, EQUAL TO
THEIR INDICES PLUS 1000,  I.E.  FROM 1001 TO 1000 + M.

ON EXIT, THE LIST OF COLUMN NAMES MAY ALSO CONTAIN ENTRIES
WITH ENLARGEMENTS OF 10000.
A NAME-CODE OF 10000 + J INDICATES THE UPPER BOUND ON THE
J TH VARIABLE.;

'IF' REENTRY # 0 'THEN' 'BEGIN'
  'COMMENT'
```

```
  FOR REENTRY=0 THE NORMAL LP ALGORITHM IS FOLLOWED,
  INCLUDING THE FILLING OF THE NAMELISTS,
  OTHERWISE THE PROCEDURE EXPECTS AN ALREADY
  UPDATED TABLEAU, WITH NAMELISTS FILLED, WHICH NEEEDS
  RE-OPTIMIZING AND/OR FINDING A NEW FEASIBLE SOLUTION.
  THE REENTRY PARAMETER ALLOWS RE-ENTRY OF THE ALGORITHM
  AFTER CHANGING THE RIGHTHAND-SIDE OR THE VALUE COLUMN,
  IN WHICH CASE REENTRY=1 SHOULD BE SUPPLIED.

  THE SAME PARAMETER ALSO SERVES AS ABNORMAL EXIT PARAMETER.

  FOR NORMAL, I.E. OPTIMAL AND FEASIBLE EXIT,
  THE REENTRY PARAMETER IS ASSIGNED THE VALUE ZERO, EVEN IF
  ITS VALUE ON ENTRY WAS DIFFERENT.
  AN UNBOUNDED PROBLEM IS INDICATED BY REENTRY = 1,
  AN EMPTY PROBLEM BY REENTRY = -1;

  INVERTED := 'TRUE';
  'GOTO' PHASE I; 'END';

FILL NAMELISTS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' COLLST[J] := J;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' ROWLST[I] := 1000+I;

SET UPPER BOUNDS AND FILL DUMMY ENTRIES:
'FOR' J:=NAV+1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[M+2,J]=0
'THEN' T[M+2,J]:=1000000;
T[M+1,N+1]:=T[M+2,N+1]:=T[M+1,N+2]:=T[M+2,N+2]:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M+2 'DO' T[I,N+2]:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' NAV 'DO' T[M+2,J]:=0;

ATTEND EQUATIONS WITH THE WRONG DIRECTION:
'FOR' I:=1 'STEP' 1 'UNTIL' NEQ 'DO' 'BEGIN'
  'IF' T[I,N+1] > 0 'THEN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
  'IF' 'NOT' T[I,J]=0 'THEN' T[I,J]:=-T[I,J]; 'END';

ATTEND PRIMAL DEGENERACY:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,N+1]=0
'THEN' 'BEGIN'
  T[I,N+1]:=1;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[I,J] < 0 'THEN'
  T[I,N+1]:=T[I,N+1]-T[I,J];
  T[I,N+1]:=0.00000000000001*T[I,N+1];
  'IF' I < NEQ+1 'THEN' T[I,N+1]:=-T[I,N+1]; 'END';

ATTEND DUAL DEGENERACY:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[M+1,J]=0
'THEN' 'BEGIN'
  T[M+1,J]:=1;
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,J] > 0 'THEN'
  T[M+1,J]:=T[M+1,J]+T[I,J];
  T[M+1,J]:=0.000000000001*T[M+1,J];
  'IF' J < NAV+1 'THEN' T[M+1,J]:=-T[M+1,J]; 'END';
```

```
PHASE 0:
ENTER VARIABLES WITHOUT SIGN RESTRICTION:
K:=0; INVERTED:='FALSE';
RETURN IN INVERSION:
K:=K+1;
'IF' K > NAV 'THEN' 'BEGIN'
  INVERTED:='TRUE'; 'GOTO' ORDER; 'END';

COLN:=K; QUO:=10000000000000000; ROWN:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[I] > NAV 'AND' T[I,K] ≠ 0 'THEN' 'BEGIN'
  TQUO:=T[I,N+1]/T[I,K];
  'IF' TQUO < 0 'THEN' TQUO:=-TQUO;
  'IF' TQUO < QUO 'THEN' 'BEGIN'
    QUO:=TQUO; R:=I; ROWN:=ROWLST[I]; 'END'; 'END';
'IF' ROWN=0 'THEN' 'GOTO' UNBOUNDED;
QUO:=VNBV:=T[R,N+1]/T[R,K]; 'GOTO' MAKE THE STEP;

ORDER:
ORDL(T,M,N,2,2,ROWLST,COLLST);

PHASE I:     FEASIBLE := 'TRUE';
FIND WHETHER A FEASIBLE SOLUTION EXISTS:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,N+1]<0 'THEN'
FEASIBLE := 'FALSE';

MAXIMIZE:    HIG:=0;  COLN:=0;  VNBV:=0;  R:=K:=0;
DUAL RATIO := 10000000000000;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' COLLST[J] < 1000 'OR' COLLST[J] > NEQ + 1000
'THEN' 'BEGIN'

  INITIALIZE SUBSTITUTE PREFERENCE DIRECTION:
  N ASC := ASC := 0;  N OF NC := 0;

  'IF' FEASIBLE 'THEN' ASC:=-T[M+1,J] 'ELSE'
  'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' T[I,N+1] < 0 'THEN' 'BEGIN'
    ASC := ASC-T[I,J];
    'IF' T[I,J]<0 'THEN' 'BEGIN'
      N OF NC := N OF NC +1;
      N ASC := N ASC - T[I,J]; 'END'; 'END';

  REFUSE UNDERSIZE PHASE I PIVOTS:
  'IF' 'NOT' FEASIBLE 'AND' ASC < 0.0000001
  'THEN' 'GOTO' FINISHED WITH THIS COLUMN;

  'IF' 'NOT' ASC > 0 'THEN' 'GOTO' FINISHED WITH THIS COLUMN;

  PUT PREFERENCE FOR PREFERRED COLUMNS:
  'IF' 'NOT' FEASIBLE 'AND'
  T[M+1,J] > 0 'AND' ASC > 0 'THEN'
  ASC := 0.001*ASC;
  'IF' 'NOT' FEASIBLE 'AND' T[M+1,J] < 0 'THEN'
  ASC := ASC-T[M+1,J];
```

```
SEARCH FOR SMALLEST QUO WITH JTH COLUMN:
QUO:=10000000000000000;
TRYN := TRYR := 0;

'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ABS(T[I,J]) > 0.00000001 'THEN' 'BEGIN'
  'IF' T[I,J] > 0 'AND' T[I,N+1] > 0
  'THEN' 'GOTO' CHECK QUOTIENT;
  'IF' T[I,J] < 0 'AND' T[M+1,J]<0 'THEN' 'BEGIN'
    'IF' ROWLST[I]<1000 'THEN' 'GOTO' TRY UPPERBOUND;
    'IF' ROWLST[I]>1000+NEQ
    'THEN' 'GOTO' FINISHED WITH THIS PIVOT; 'END';

  'IF' T[I,J] < 0 'AND' T[I,N+1] < 0
  'THEN' 'GOTO' CHECK QUOTIENT;
  'IF' T[I,J] < 0 'AND' ROWLST[I] < 1000 'THEN'
  'GOTO' TRY UPPER BOUND;
  'IF' T[I,J] <0 'OR' T[I,N+1] < 0 'THEN'
  'GOTO' FINISHED WITH THIS PIVOT;

  CHECK QUOTIENT:
  UPPERBOUND:= 'FALSE';
  TQUO:=T[I,N+1]/T[I,J];

  'IF' T[I,J] < 0 'AND' T[I,N+1]<0 'THEN' 'BEGIN'
    'IF' ROWLST[I] > 1000 'AND' ROWLST[I] < NEQ+1001
    'THEN' 'GOTO' CHECK DUAL RATIO;
    'IF' I=R 'AND' 'NOT' K=0 'THEN' 'BEGIN'
      'IF' T[M+1,J] > 0 'AND' T[M+1,K]>0
      'THEN' 'GOTO' CHECK DUAL RATIO; 'END'; 'END';

  'IF' T[I,J] < 0 'AND' -T[I,J]*N OF NC*3 < N ASC
  'THEN' 'GOTO' TRY UPPERBOUND;

  'IF' (T[I,N+1]<0 'AND' T[I,J]<0) 'THEN' 'BEGIN'
    'IF' 'NOT' TRYR=0 'THEN' 'BEGIN'
      'IF' (ROWLST[TRYR]=TRYN 'AND' T[TRYR,J]<0
      'AND' T[TRYR,N+1]<0)
      'THEN' 'BEGIN'
        'IF' TQUO < T[TRYR,N+1]/T[TRYR,J]
        'THEN' 'GOTO' TRY UPPERBOUND; 'END'; 'END'; 'END';

  CHECK DUAL RATIO:
  'IF' (T[I,N+1]<0 'AND' T[I,J]<0 'AND' T[M+1,J]>0)
  'AND' 'NOT' K=0 'THEN' 'BEGIN'
    'IF' (I=R 'AND' -T[M+1,J]/T[I,J] > DUAL RATIO)
    'AND' T[M+1,K]>0
    'THEN' 'GOTO' FINISHED WITH THIS COLUMN; 'END';
```

```
  MEASURE:
  'IF' TQUO > QUO 'THEN' 'BEGIN'
    'IF' T[I,J] > 0 'OR' UPPERBOUND
    'THEN' 'GOTO' FINISHED WITH THIS PIVOT;
    'IF' TRYR > 0 'THEN' 'BEGIN'
      'IF' TRYN < 10000 'AND' T[TRYR,J] > 0
      'THEN' 'GOTO' FINISHED WITH THIS PIVOT;
      'IF' TRYN > 10000 'AND' T[TRYR,J] < 0
      'THEN' 'GOTO' FINISHED WITH THIS PIVOT; 'END'; 'END';-

  QUO:=TQUO;
  'IF' QUO*ASC < HIG 'AND' (T[I,J] > 0 'OR' UPPERBOUND)
  'THEN' 'GOTO' FINISHED WITH THIS COLUMN;
  TRYR:=I; TRYN:=ROWLST[I];
  'IF' UPPERBOUND 'THEN' TRYN:=TRYN+10000;
  'GOTO' FINISHED WITH THIS PIVOT;

  TRY UPPER BOUND:
  'IF' T[I,J] > 0 'THEN' 'GOTO' FINISHED WITH THIS PIVOT;
  'IF' ROWLST[I]>1000 'AND' ROWLST[I]<1001+M
  'THEN' 'GOTO' FINISHED WITH THIS PIVOT;
  UPPERBOUND:='TRUE'; TQUO:=-T[I,N+2]/T[I,J];
  'GOTO' MEASURE;
  FINISHED WITH THIS PIVOT: 'END';

'IF' QUO > 999000000000000 'AND' COLLST[J] > 1000
'AND' COLLST[J] < 10000 'THEN' 'GOTO' UNBOUNDED;
'IF' QUO > T[M+2,J] 'THEN' 'BEGIN'
  'IF' COLLST[J] > 1000 'AND' COLLST[J] < 10000
  'THEN' 'GOTO' END OF DIRECT HIT LOOP;
  QUO:=T[M+2,J];
  'IF' COLLST[J] < 10000 'THEN' TRYN := COLLST[J]+10000
  'ELSE' TRYN := COLLST[J]-10000;
  END OF DIRECT HIT LOOP: 'END';

CHECK HEIGHT OF STEP:
'IF' TRYR ≠ 0 'AND' K≠0 'THEN' 'BEGIN'
  'IF' (TRYR=R 'AND' ROWLST[TRYR]=TRYN
  'AND' T[TRYR,J]<0 'AND' T[TRYR,N+1]<0 'AND' T[M+1,J]>0
  'AND' T[M+1,K]>0 'AND' -T[M+1,J]/T[TRYR,J] < DUAL RATIO)
  'THEN' HIG:=0; 'END';

'IF' QUO*ASC > HIG 'THEN' 'BEGIN'
  HIG:=QUO*ASC; VNBV:=QUO;
  'IF' 'NOT' ABS(TRYN-COLLST[J]) = 10000 'THEN' R:=TRYR
  'ELSE' R:=0;
  'IF' 'NOT' R=0 'THEN' 'BEGIN'
    'IF' ROWLST[R]=TRYN 'AND' T[R,J]<0 'AND' T[R,N+1]<0
    'THEN' DUAL RATIO:=-T[M+1,J]/T[R,J];
    'END';
  ROWN:=TRYN; K:=J; COLN:=COLLST[J]; 'END';
```

```
  FINISHED WITH THIS COLUMN: 'END';

'IF' COLN = 0 'THEN' 'BEGIN'
  'IF' FEASIBLE 'THEN' 'GOTO' OUT 'ELSE' 'GOTO' EMPTY;
  'END';

MAKE THE STEP:
COLLST[K]:=ROWN; QUO:=VNBV;

ADJUST RIGHTHAND SIDE AND UPPER BOUNDS COLUMN:
'FOR' I:=1 'STEP' 1 'UNTIL' R-1, R+1 'STEP' 1 'UNTIL' M+1
'DO' 'IF' T[I,K] # 0 'THEN' 'BEGIN'
  T[I,N+1]:=T[I,N+1]-T[I,K]*QUO; 'IF' T[I,N+1]=0
  'THEN' T[I,N+1]:=0.00000001;
   T[I,N+2]:=T[I,N+2]+T[I,K]*QUO;
  'IF' T[I,N+2]=0 'THEN' T[I,N+2]:=0.00000001;  'END';

CONSIDER ONE COLUMN UPDATE:
'IF' ABS(ROWN-COLN)=10000 'THEN' 'BEGIN'
  'COMMENT'
  DIRECT HIT OF THE UPPER BOUND ON THE COLUMN-VARIABLE.
  NO FULL UPDATING OF THE TABLEAU IS REQUIRED;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
  'IF' T[I,K]#0 'THEN' T[I,K]:=-T[I,K];
  'GOTO' CHECK FOR STATUS;  'END';

ATTEND UPPERBOUNDS OF PIVOTAL PAIR:
COP:=T[M+2,K]; NUM:=T[M+2,K]:=T[R,N+1]+T[R,N+2];
T[R,N+1] := QUO; T[R,N+2] := COP-QUO;
'IF' T[R,N+2]=0 'THEN' T[R,N+2]:=0.00000001;
'IF' QUO=0 'THEN' T[R,N+1]:=0.00000001;

REFORMULATE ROW WITH UPPER BOUND NAME:
'IF' ROWN > 10000 'THEN'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' T[R,J]:=-T[R,J];

UPDATE:
PIV:=T[R,K];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[R,J]#0 'THEN'
T[R,J]:=T[R,J]/PIV;

'FOR' J:=1 'STEP' 1 'UNTIL' K-1,K+1 'STEP' 1 'UNTIL' N 'DO'
'IF' 'NOT' T[R,J] = 0 'THEN'

'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M+1 'DO'
'IF' 'NOT' T[I,K] =0 'THEN' T[I,J]:=T[I,J]-T[R,J]*T[I,K];
'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO' 'BEGIN'
  'IF' I < M+1 'AND' ABS(T[I,K]) < 0.0000001
  'THEN' T[I,K]:=0;
  'IF' 'NOT' T[I,K]=0 'THEN' T[I,K]:=-T[I,K]/PIV; 'END';
T[R,K]:=1/PIV;
```

```
ROWLST[R]:=COLN;

IF NECESSARY REFORMULATE NEW ROW WITH UPPERBOUND NAME:
'IF' COLN > 10000 'THEN' 'BEGIN'  ROWLST[R]:=COLN-10000;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[R,J]≠0 'THEN'
  T[R,J]:=-T[R,J];
  COP:=T[R,N+1]+T[R,N+2]; T[R,N+1]:=COP-QUO; T[R,N+2]:=QUO;
  'END';

CHECK FOR STATUS:
'IF' 'NOT' INVERTED 'THEN' 'GOTO' RETURN IN INVERSION;
'IF' FEASIBLE 'THEN' 'GOTO'  MAXIMIZE 'ELSE' 'GOTO' PHASE I;


EMPTY:
REENTRY := -1;
NEWLINE(1);
WRITETEXT('('EMPTY%PROBLEM')');
NEWLINE(1);
'GOTO' ORDER FOR EXIT;

UNBOUNDED:
REENTRY := 1;
NEWLINE(1);
WRITETEXT('('UNBOUNDED%COLUMN')');
WRITE(30,FORMAT('('S-NDDDDDD')'),COLLST[J]);
NEWLINE(1);
'GOTO' END OF LINP;


OUT:
'IF' FEASIBLE 'THEN' REENTRY :=0 'ELSE' REENTRY := -1;
ORDER FOR EXIT:
ORDL(T,M,N,2,2,ROWLST,COLLST);

END OF LINP:  'END';
```

12.4 Printing a Simplex Tableau

The tableau printing procedure offered in this section is an adaptation of the matrix printing procedure mentioned in Section 2.18. The tableau is printed with its two namelists. The following different features are, however, noticeable.

In connection with the printing of a rownameslist on the left-hand side of each block, the number of columns per block-column is reduced (from 15 to 13). The facility to "skip" leading rows and columns has been integrated with the printing of names. Name-codes are printed only for "proper" rows and columns, not for supplementary vectors, e.g. target-row and value column. If all proper rows are skipped, i.e. only one or more "extra" rows are printed, no list of rownames is required and 15 rather than 13 columns per block-column are printed.

The printing of the indices of the leading element of each block is replaced by the printing of the list of columnnames. This is done every thirty lines, and a blank line is inserted

half-way. The procedure can also be used for printing certain blocks of the tableau itself, in the same way as with the matrix printing procedure. The facility for scaling has not been maintained. The text of the procedure is as follows:

TEXT-LISTING OF THE TABLEAU-PRINTING PROCEDURE.

```
'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWLST,COLLST;

'BEGIN' 'INTEGER' NRDO,NRTD,NCOLDO,NCOLTD,
PAGEC,RPRINP,CPRINP,I,J,NAME,INDEX;
'REAL' NUM;

  'COMMENT'
  TABO STANDS FOR TABLEAU OUT.
  IT IS A TABLEAU PRINTING PROCEDURE, PRINTING THE TABLEAU
  IN BLOCKS.
  THESE BLOCKS ARE BORDERED AT THE TOP AND THE LEFTHANDSIDE,
  BY THE APPROPRIATE PARTS OF THE NAMELISTS.

  THE TABLEAU-MATRIX IS OF ORDER M+ER BY N+RH.
  ER STANDS FOR EXTRA ROWS, E.G. TARGET-ROW, ETC.
  RH STANDS FOR RIGHT-HAND SIDE COLUMNS, E.G. VALUE COLUMN,
  DISTANCES COLUMN, ETC.
  SR STANDS FOR SKIP ROWS.
  SC STANDS FOR SKIP COLUMNS.

  THE AUXILIARY VARIABLES NRDO,NRTD,NCOLDO,NCOLTD,PAGEC,
  RPRINP, AND CPRINP  HAVE THE SAME SIGNIFICANCE AS IN MATO.


  NRDO:=NCOLDO:=0; NRTD:=M+ER; NCOLTD:=N+RH;
  NEWLINE(1);

  START:
  NEWLINE(3);
  RPRINP:=30;
  'IF' SC < N 'THEN' CPRINP:=13 'ELSE' CPRINP:=15;

  ADJUST NUMBER OF COLUMNS:
  'IF' NCOLTD < CPRINP 'THEN' CPRINP:=NCOLTD;
  NCOLTD:=NCOLTD-CPRINP;

  CHECK FOR PAGE:
  'IF' NRTD > 15 'AND' N > 15 'THEN'
  PAGEC:=NRDO/30-ENTIER(NRDO/30) 'ELSE'
  PAGEC:=1;
  'IF' PAGEC=0 'THEN' 'BEGIN'
    'COMMENT'
    BLOCKS ALL TO START AT THE TOP OF A PAGE.
    THIS ALLOWS THE USER TO STICK THEM TOGETHER,
    WHILE KEEPING ROWS AND COLUMNS IN LINE.;
    RUNOUT;    'END';
  NEWLINE(2);
```

```
HEADING:

ALPHANUMERICAL COLUMNNAMES LIST:
WRITETEXT('('NAME%!%%%%%%!!%')');
'FOR' J:=1 'STEP' 1 'UNTIL' CPRINF 'DO' 'BEGIN'

  'IF' SC+NCOLDO+J > N 'THEN' 'GOTO' PRINT FOR VALUE COLUMN;

  SPACE(2);
  NAME := COLLSTC SC+NCOLDO+J];

  X 1STAR IN CL:
  'IF' 'NOT' (NAME>N 'AND' NAME<2*N+1 'AND' N<300)
  'THEN' 'GOTO' X 2STAR IN CL;
  WRITETEXT('('%X*')'); INDEX:=NAME-N;
  'GOTO' WRITE THE INDEX IN CL;

  X 2STAR IN CL:
  'IF' 'NOT' (NAME>2*N 'AND' NAME<3*N+1 'AND' N<300)
  'THEN' 'GOTO' X IN CL;
  WRITETEXT('('X**')'); INDEX:=NAME-2*N;
  'GOTO' WRITE THE INDEX IN CL;

  X IN CL:
  'IF' 'NOT' (NAME>0 'AND' NAME<1000)
  'THEN' 'GOTO' SLACK IN CL;
  WRITETEXT('('%%X')'); INDEX:=NAME;
  'GOTO' WRITE THE INDEX IN CL;

  SLACK IN CL:
  'IF' 'NOT' (NAME>1000 'AND' NAME<2000)
  'THEN' 'GOTO' UPLIM IN CL;
  WRITETEXT('('%%S')'); INDEX := NAME-1000;
  'GOTO' WRITE THE INDEX IN CL;

  UPLIM IN CL:
  'IF' 'NOT' ((NAME>2000 'AND' NAME<3000)
  'OR' NAME >10000) 'THEN' 'GOTO' ARTFV IN CL;
  WRITETEXT('('%%E')');
  'IF' NAME < 10000 'THEN' INDEX:=NAME-2000
  'ELSE' INDEX:=NAME-10000;
  'GOTO' WRITE THE INDEX IN CL;

  ARTFV IN CL:
  'IF' 'NOT' (NAME>3000 'AND' NAME<4000)
  'THEN' 'GOTO' DUALV IN CL;
  WRITETEXT('('%%A')'); INDEX:=NAME-3000;
  'GOTO' WRITE THE INDEX IN CL;

  DUALV IN CL:
  'IF' 'NOT' (NAME<-1000 'AND' NAME>-2000)
  'THEN' 'GOTO' DUALSL IN CL;
  WRITETEXT('('%%P')'); INDEX:=-NAME-1000;
  'GOTO' WRITE THE INDEX IN CL;
```

```
   DUALSL IN CL:
   'IF' 'NOT' (NAME<0 'AND' NAME>-1000)
   'THEN' 'GOTO' DUAL UL IN CL;
   WRITETEXT('('%%D')'); INDEX:=-NAME;
   'GOTO' WRITE THE INDEX IN CL;

   DUAL UL IN CL:
   'IF' 'NOT' (-NAME>3000 'AND' -NAME<4000)
   'THEN' 'GOTO' SHADOWPRICE UPPER LIMIT IN CL;
   WRITETEXT('('%%Z')'); INDEX:=-NAME-3000;
   'GOTO' WRITE THE INDEX IN CL;

   SHADOWPRICE UPPER LIMIT IN CL:
   'IF' 'NOT' (-NAME>2000 'AND' -NAME<3000)
   'THEN' 'GOTO' WRITE THE INDEX IN CL;
   WRITETEXT('('%%U')'); INDEX:=-NAME-2000;

   WRITE THE INDEX IN CL:
   'IF' INDEX<99 'THEN'
   WRITE(30,FORMAT('('NDS')'),INDEX)
   'ELSE'
   WRITE(30,FORMAT('('NDD')'),INDEX); 'END';

'GOTO' PUT UPPER BORDERLINE OF TABLEAU ITSELF;

PRINT FOR VALUE COLUMN:
'IF' SC+NCOLDO+J=N+1 'THEN' WRITETEXT('('%!!%%%VALUE')');

PUT UPPER BORDERLINE INSIDE HEADING:
NEWLINE(1);
'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO'
WRITETEXT('('--------')');
'IF' SR<M 'THEN' WRITETEXT('('--------------')');
'IF' SC+NCOLDO+CPRINP = N+1
'THEN' WRITETEXT('('----')');

CODING COLUMNLIST:
NEWLINE(1);
'IF' SR<M 'THEN' WRITETEXT('('%%%%%!%CODE%!!%')');

'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO' 'BEGIN'
   'IF' SC+J+NCOLDO > N+1 'THEN' 'GOTO' BLANK;
   'IF' SC+J+NCOLDO = N+1
   'THEN' 'GOTO' SEPARATION HEADING;
   'IF' ABS(COLLST[SC+NCOLDO+J]) > 9999 'THEN'
   WRITE(30,FORMAT('('-NDDDDDD')'),COLLST[SC+NCOLDO+J])
   'ELSE' WRITE(30,FORMAT('('-NDDDDS')'),COLLST[SC+NCOLDO+J]);
   'GOTO' BLANK;
   SEPARATION HEADING:
   WRITETEXT('('%!!%%%%%')');
   BLANK:  'END';
NEWLINE(1);
```

```
PUT UPPER BORDERLINE OF TABLEAU ITSELF:
'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO'
WRITETEXT('('--------')');
'IF' SR<M 'THEN' WRITETEXT('('--------------')');
'IF' SC+NCOLDO+CPRINP = N+1
'THEN' WRITETEXT('('----')');

ADJUST NUMBER OF ROWS:
'IF' NRTD < RPRINP 'THEN' RPRINP:=NRTD;
NRTD:=NRTD-RPRINP;

PRINT THE LINES:
'FOR' I:=1 'STEP' 1 'UNTIL' RPRINP 'DO' 'BEGIN'
  NEWLINE(1);

  'IF' SR+NRDO+I=M+1 'THEN' 'BEGIN'
    PUT LOWER BORDERLINE:
    'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO'
    WRITETEXT('('--------')');
    'IF' SR<M 'THEN' WRITETEXT('('--------------')');
    'IF' SC+NCOLDO+CPRINP = N+1
    'THEN' WRITETEXT('('----')');
    NEWLINE(1); 'END';

  'IF' I=16 'THEN' NEWLINE(1);
  'IF' SR > M-1 'THEN' 'GOTO' NO ROWLIST NEEDED;

  'IF' SR+I+NRDO > 0 'AND' SR+I+NRDO < M+1
  'THEN' 'BEGIN'

    ALPHANUMERICAL ROWNAMES LIST:
    NAME := ROWLST[SR+NRDO+I];

    X 1STAR IN RL:
    'IF' 'NOT' (NAME>N 'AND' NAME<2*N+1 'AND' N<300)
    'THEN' 'GOTO' X 2STAR IN RL;
    WRITETEXT('('X*')'); INDEX:=NAME-N;
    'GOTO' WRITE THE INDEX IN RL;

    X 2STAR IN RL:
    'IF' 'NOT' (NAME>2*N 'AND' NAME<3*N+1 'AND' N<300)
    'THEN' 'GOTO' X IN RL;
    WRITETEXT('('X**')'); INDEX:=NAME-2*N;
    'GOTO' WRITE THE INDEX IN RL;

    X IN RL:
    'IF' 'NOT' (NAME>0 'AND' NAME<1000)
    'THEN' 'GOTO' SLACK IN RL;
    WRITETEXT('('X')'); INDEX:=NAME;
    'GOTO' WRITE THE INDEX IN RL;

    SLACK IN RL:
    'IF' 'NOT' (NAME>1000 'AND' NAME<2000)
    'THEN' 'GOTO' UPLIM IN RL;
    WRITETEXT('('S')'); INDEX := NAME-1000;
    'GOTO' WRITE THE INDEX IN RL;
```

```
  UPLIM IN RL:
  'IF' 'NOT' ((NAME>2000 'AND' NAME<3000)
  'OR' NAME >10000) 'THEN' 'GOTO' ARTFV IN RL;
  WRITETEXT('('B')');
  'IF' NAME < 10000 'THEN' INDEX:=NAME-2000
  'ELSE' INDEX:=NAME-10000;
  'GOTO' WRITE THE INDEX IN RL;

  ARTFV IN RL:
  'IF' 'NOT' (NAME>3000 'AND' NAME<4000)
  'THEN' 'GOTO' DUALV IN RL;
  WRITETEXT('('A')'); INDEX:=NAME-3000;
  'GOTO' WRITE THE INDEX IN RL;

  DUALV IN RL:
  'IF' 'NOT' (NAME<-1000 'AND' NAME>-2000)
  'THEN' 'GOTO' DUALSL IN RL;
  WRITETEXT('('P')'); INDEX:=-NAME-1000;
  'GOTO' WRITE THE INDEX IN RL;

  DUALSL IN RL:
  'IF' 'NOT' (NAME<0 'AND' NAME>-1000)
  'THEN' 'GOTO' DUAL UL IN RL;
  WRITETEXT('('D')'); INDEX:=-NAME;
  'GOTO' WRITE THE INDEX IN RL;

  DUAL UL IN RL:
  'IF' 'NOT' (-NAME>3000 'AND' -NAME<4000)
  'THEN' 'GOTO' SHADOWPRICE UPPER LIMIT IN RL;
  WRITETEXT('('Z')'); INDEX:=-NAME-3000;
  'GOTO' WRITE THE INDEX IN RL;

  SHADOWPRICE UPPER LIMIT IN RL:
  'IF' 'NOT' (-NAME>2000 'AND' -NAME<3000)
  'THEN' 'GOTO' WRITE THE INDEX IN RL;
  WRITETEXT('('U')'); INDEX:=-NAME-2000;


  WRITE THE INDEX IN RL:
  'IF' NAME > N 'AND' NAME < 2*N+1 'THEN'
  WRITE(30,FORMAT('('ND')'),INDEX) 'ELSE' 'BEGIN'
    'IF' INDEX < 100
    'THEN' WRITE(30,FORMAT('('NDS')'),INDEX)
    'ELSE' WRITE(30,FORMAT('('NDD')'),INDEX); 'END';
  'IF' ABS(NAME)<10000 'THEN' WRITETEXT('('%!')');


  CODED ROWNAMES LIST:
  'IF' ABS(NAME)>10000
  'THEN' WRITE(30,FORMAT('('S-NDDDDD')'),NAME)
  'ELSE' WRITE(30,FORMAT('('-NDDD')'),NAME);
  'END'

'ELSE' 'BEGIN'
  'IF' SR+I+NRDO = M+1 'THEN' WRITETEXT('('%%T%%!%%%%%')')
  'ELSE' WRITETEXT('('%%%%%!%%%%%')'); 'END';
```

```
    NO ROWLIST NEEDED:

    'FOR' J:=1 'STEP' 1 'UNTIL' CPRINP 'DO' 'BEGIN'
      'IF' SC+NCOLDO+J-1 = 0 'THEN' WRITETEXT('('%!!%')');
      'IF' SC+NCOLDO+J-1 = N 'THEN' WRITETEXT('('%!!')');
      'IF' NCOLDO+J=1 'AND' SC>0 'THEN' WRITETEXT('('%!!%')');
      NUM := MATRIX[SR+NRDO+I,SC+NCOLDO+J];
      'IF' NUM=0 'THEN' WRITETEXT('('%%%-%%%')')
      'ELSE' 'BEGIN'
        'IF' NUM=ENTIER(NUM)
        'THEN' WRITE(30,FORMAT('('S-NDDSSS')'),NUM)
        'ELSE' WRITE(30,FORMAT('('S-NDD.DD')'),NUM); 'END';
      'END'; 'END';

  ADJUST INDICES AND RETURN UNLESS READY:
  NRDO:=NRDO+RPRINP;
  'IF' NRDO=M+ER 'THEN' 'BEGIN'
    NRTO:=M+ER; NRDO:=0; NCOLDO:=NCOLDO+CPRINP;
    'IF' NCOLDO=N+EH 'THEN' 'GOTO' END OF TABO;
    'GOTO' START; 'END' 'ELSE' 'GOTO' CHECK FOR PAGE;

  END OF TABO:  NEWLINE(2);  'END';
```

When reading the text of the tableau-printing procedure listed above, one will have noticed, that it contains references to other name-codes than only the ones discussed in section 12.1.

This is because the same procedure is also used for printing quadratic programming tableaux, in which negative name-codes and enlargements of 2000 occur, and integer programming tableaux, in which enlargements by n and 2n occur. The precise significance of these codes will be discussed later on in this book.

12.5 Text of a complete L.P. programme

In this section we offer a complete "main" programme, despite the fact that this contains unavoidably, relatively many system-specific features. The shortness of this programme is the main justification. The main programme controls the input of the data, i.e. the tableau-matrix, and the printing of the results. The Simplex Algorithm itself is performed by the LINP procedure presented in Section 12.3.

Even the input and output of data and results is largely delegated to subordinate procedures. These are the matrix-reading procedure MATI listed in Section 2.18 and the tableau printing procedure listed in the previous section 12.4 of this Chapter, and the results reporting procedure from section 10.4.

The one element of evaluation by the main programme is whether or not to print the full tableau. With a big system it is obviously undesirable to have a pile of paper full of numbers. But the particular limit of 13 columns and 40 rows and columns together arises from the economical use of line-printer paper. If there are no more than 13 columns or less than 15 rows and only two block-columns to print the printing of the full tableau is the most economical way to present the output. For larger systems, output is restricted to a call to the result-reporting procedure only.

Note that whereas the separate call to the reporting procedure provides re-interpretation of the solution vector, the tableau-printing procedure does not do that, i.e. the figure printed after the symbol "x_4" is the distance between x_4 and its lower limit, which may, or may not be zero. The programme text is now listed, as follows:

```
'BEGIN' 'INTEGER' M,N,NAV,NEQ,REENTRY,I,J;

'PROCEDURE' LINP(T,M,N,NEQ,NAV,ROWLST,COLLST,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'ALGOL';

'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;  'ALGOL';

'PROCEDURE' REPO(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';

'COMMENT'
LINEAR PROGRAMMING BY THE SIMPLEX ALGORITHM.
FOR DETAILS OF THE ALGORITHM, SEE THE TEXT OF THE LINP
PROCEDURE.
```

```
PRESENTATION OF DATA:
FIRST THE NUMBER OF RESTRICTIONS I.E. M,
THEN THE NUMBER OF VARIABLES, I.E. N,
FOLLOWED BY THE NUMBER OF EQUATIONS,  NEQ,
AND AS LAST INTEGER ORDER PARAMETER, NAV,
THE NUMBER OF 'FREE' VARIABLES TO WHICH THE
TACIT (NON-NEGATIVITY) RESTRICTION DOES   N O T  APPLY.

THEREAFTER PUNCH EACH ROW OF THE COMPOSITE MATRIX

   A      B
   W'     0
   U'     0
   L'     0

TO REPRESENT  A.X  < OR = B,
MAXIM  -W'X
AND  X  < OR = U
AND  X  > OR = L       .

THE PROGRAMME READS ALL THE ELEMENTS OF THE UPPER AND LOWER
BOUNDS VECTORS U AND L,   DESPITE THE FACT THAT
THE MAIN LINP PROCEDURE USES THESE NUMBERS ONLY FOR
BOUNDED VARIABLES
;
M:=READ;  N:=READ;  NEQ:=READ;  NAV:=READ;
REENTRY:=0;

'BEGIN'
  'ARRAY' TA[1:M+3,1:N+2];
  'INTEGER' 'ARRAY' ROWL[1:M],COLL[1:N];

  READ MAIN TABLEAU MATRIX:
  MATI(TA,M+3,N+1,0,0);

  REINTERPRET:
  'FOR' J:=NAV+1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
    TA[I,N+1]:=TA[I,N+1]-TA[I,J]*TA[M+3,J];
    'IF' TA[M+2,J]=0 'THEN' TA[M+2,J]:=1000000;
    TA[M+2,J]:=TA[M+2,J]-TA[M+3,J];
    'IF' TA[M+2,J] < 0 'THEN' 'BEGIN'
      NEWLINE(1);
      WRITETEXT('('YOU%HAVE%SPECIFIED%A%LOWER%LIMIT%
      %IN%EXCESS%OF%THE%CORRESPONDING%UPPER%LIMIT%
      %THEREBY%CAUSING%EMPTYNESS.')'); 'END';
    'END';

  LINP(TA,M,N,NEQ,NAV,ROWL,COLL,REENTRY);

  REPO(TA,M,N,NEQ,NAV,ROWL,COLL);

  'IF' N < 14 'OR' M+N < 40 'THEN'
  TABO(TA,M,N,0,0,1,1,ROWL,COLL);

  'END'; 'END'
```

12.6 Revised Simplex Algorithms

The algorithms which we discuss under this heading have the following features in common:

a) Packed storage of the set-up tableau

The tableau matrix which defines the typical large L.P. problem tends to a very predominant degree to consist of nothing but zeros, quite often in excess of 99% zeros. Obviously, substantial economies in storage-space are obtained if this matrix is stored in some other way than storing each element.

For example, if the matrix-reading procedure from section 2.10 were used to read the tableau-matrix, one would simply begin by storing the input-information as such, and at the same time compile a list of addresses of the heads of the successive rows.

(There are in fact other input-conventions and other ways of "packed" storage.)

Even if, as may be desirable in some versions of the revised simplex - the set-up tableau is stored twice, once per column, once per row, that still is a fraction of the storage requirement of the unpacked tableau-matrix.

b) No updating of the tableau

The current tableau is normally more or less full of non-zero elements, even if the set-up tableau consists largely of zeros.

Storage and updating of the current tableau is therefore suppressed. Instead, some other representation of the current solution is used. The value column and the list of name-codes are kept currently updated all the time, and in some versions also the objective function row. Other rows and columns are generated as needed.

c) They are utterly impracticable on paper, but specifically geared towards making an efficient use of computer-resources. We now review the separate algorithms.

Explicit inverse with row-updating

This is the oldest one of this group of algorithms and one will find it referred to in some older references as "the" reversed simplex method.

In this method, one stores besides the packed form of the original tableau, the current value column, the current

objective function row and the inverse of the basis-matrix, the objective function being counted as a basic variable.

The current updated tableau (explicit form) is then:

$$U = [B^{-1} T \mid B^{-1}] \qquad (12.6.1)$$

where T is the original tableau matrix in shortened form.

B^{-1} itself is kept updated at each step in the usual way.

B^{-1} is the inverse of the "extended basis-matrix"

$$B = \begin{bmatrix} A_{11} & & \\ A_{21} & I & \\ -w_1' & & 1 \end{bmatrix} \qquad (12.6.2)$$

If a slack-variable is indicated as incoming variable, the pivotal column does not need calculation as it is already present in the inverse itself.

If an element of $\underline{x}$ indicated as incoming variable, the pivotal column is obtained by post-multiplication of the inverse by the corresponding column of the original tableau.

Example (from Chapter VIII)

$$T = \begin{array}{c|ccc|c} & x_1 & x_2 & x_3 & \text{Value} \\ s_1 & 1 & 1 & 1 & 200 \\ s_2 & 1 & 3 & - & 100 \\ \hline \tau & -2 & -5 & -1 & - \end{array}$$

$$B^{-1} = \begin{array}{c|cc|c} & s_1 & s_2 & \tau \\ s_1 & 1 & -1.33 & - \\ x_2 & - & 0.33 & - \\ \hline \tau & - & 1.67 & 1 \end{array}$$

The x_1 column may now be calculated as follows:

$$\begin{array}{c} \\ s_1 \\ x_2 \\ \\ \tau \end{array} \begin{array}{c} \begin{array}{cccc} & s_1 & s_2 & \tau \end{array} \\ \left[\begin{array}{ccc} 1 & -0.33 & - \\ - & 0.33 & - \\ \hline - & 1.66 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ s_1 \\ s_2 \\ \\ \tau \end{array} \begin{array}{c} \begin{array}{c} x_1 \end{array} \\ \left[\begin{array}{c} 1 \\ 1 \\ \hline -2 \end{array}\right] \end{array} = \begin{array}{c} \\ s_1 \\ x_2 \\ \\ \tau \end{array} \begin{array}{c} \begin{array}{c} x_1 \end{array} \\ \left[\begin{array}{c} 0.67 \\ 0.33 \\ \hline -0.33 \end{array}\right] \end{array}$$

(The actual storage of the unit vector for the objective function is normally dispensed with.) The pivotal row is selected in the normal way and the missing part of the pivotal row is now calculated by means of a matrix-vector multiplication, i.e. the original tableau is premultiplied by the relevant row of the inverse. Reference to the objective function may be suppressed at this stage.

Example

$$\begin{array}{c} \\ s_1 \end{array} \begin{array}{c} \begin{array}{cc} s_1 & s_2 \end{array} \\ \left[\begin{array}{cc} 1 & -0.33 \end{array}\right] \end{array} \begin{array}{c} \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \left[\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 3 & - \end{array}\right] \end{array} = \begin{array}{c} \\ s_1 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \left[\begin{array}{ccc} 0.67 & - & 1 \end{array}\right] \end{array}$$

The inverse, the value column and the objective function row can then be updated in the usual way.

If the initial tableau-matrix T is entirely full of non-zero elements, this method requires the same amount of storage as a full explicit tableau, and more computational effort.

Calculation of the remainder of the pivotal row, plus updating the inverse requires as much elementary computation as updating the full tableau, and calculating the pivotal column comes to it, unless the incoming variable is a slack-variable. Obviously, under those assumptions, the shortened tableau, updated at each step, is more efficient. The method comes into its own, only if T consists largely of zero elements. That cuts the memory-space requirement on account of the "packed" representation of T, and it cuts the calculation-time needed for the matrix-vector multiplications, on account of the suppression of multiplication by zero.

Explicit inverse, without row-updating

In this version of the explicit inverse method, the calculation of the remainder part of the pivotal row is suppressed, and it

becomes therefore impossible to update the objective function row in the usual way. One then first of all gives priority to slack-variables as incoming variables: their shadowprices are always available and no need to calculate a pivotal column arises from selecting a slack-variable as incoming variable. When a pivotal column is found in the inverse itself, the step is made, otherwise the objective function row is calculated by means of a matrix vector multiplication (the same could also be done with a substitute objective function).

We have here a trade-off between the calculation effort per step, and the number of steps needed. One is forced to compromise the rule of the steepest ascent (the rule of the highest step is out anyhow for all versions of the revised simplex).

The method is likely to be more efficient than the version with row-updating, if the number of restrictions is substantially less than the number of variables.

If there are say 50 restrictions and 1000 variables, steps which are performed solely on the 50 x 50 inverse are comparatively very cheap indeed and the fact that some of them will not lead to much gain in the solution value does not terribly matter.

The product form inverse

In this algorithm only vectors are updated. The name refers to the fact that one obtains the explicit inverse, as the product of a series of matrices. Each of these matrices consists of a unit matrix of which one column has been replaced by the column which describes the variable which has just left the basis, i.e. the updated pivotal column, divided by minus the pivot, with the pivot itself replaced by the reciprocal.

Example

(The same, for the actual columns, see tableau 8.8, for the explicit inverse see tableau 8.6.)

$$
\begin{array}{c} \\ x_3 \\ x_2 \\ \tau \end{array}
\begin{array}{c} \begin{array}{ccc} s_1 & s_2 & \tau \end{array} \\ \begin{bmatrix} 1 & - & - \\ - & 1 & - \\ 1 & - & 1 \end{bmatrix} \end{array}
\begin{array}{c} \\ s_1 \\ x_2 \\ \tau \end{array}
\begin{array}{c} \begin{array}{ccc} s_1 & s_2 & \tau \end{array} \\ \begin{bmatrix} 1 & -0.33 & - \\ - & 0.33 & - \\ - & 1.67 & 1 \end{bmatrix} \end{array}
=
\begin{array}{c} \\ x_3 \\ x_2 \\ \tau \end{array}
\begin{array}{c} \begin{array}{ccc} s_1 & s_2 & \tau \end{array} \\ \begin{bmatrix} 1 & -0.33 & - \\ - & 0.33 & - \\ 1 & 1.33 & 1 \end{bmatrix} \end{array}
$$

"s_1-matrix" "s_2-matrix" = B^{-1}

In application one obviously only stores columns (and pivots and their name-codes and the row-indices of successive pivotal rows), not matrices.

The product-form representation of the inverse is effective in updating columns as they are needed. In effect one simply reconstructs the updating of the relevant column of the tableau as it would have taken place in the standard form of the simplex method.

Example

Update the value column

	Value			Value			Value
s_1	200	-0.33	s_1	(166.67)	1	x_3	166.67
s_3	(100)	0.33	→ x_2	33.33	-	→ x_2	33.33
τ	-	1.67	τ	166.67	1	τ	333.33
	(x_2 incoming)			(x_3 incoming)			

The ring here indicates the leaving variable, the actual pivot is the reciprocal of the corresponding entry in the already updated pivotal column.

Unfortunately, if only columns are stored, there is no practicable way of updating rows, no consistent information for their updating is available, short of still belatedly updating the whole tableau.

Two possible alternatives are now open:

a. Update the column for which the log indicates that it is the one which was the longest inactive, without even having been able to consult its shadowprice, and accept it as incoming variable if its shadowprice turns out to have the appropriate sign.

b. Keep a set of pivotal rows as well. At first sight, neither procedure appears to be particularly efficient. Procedure (a) is clearly inefficient, and there are other indications, i.e. the machine requirements of the commercially available LP packages, that (b) is used (two channels of backing store tend to be asked for). At first sight, that appears not very efficient either, the total memory storage

requirement of the two product-form presentations of the (primal and dual) inverses easily exceeds the size of the updated tableau-matrix in shortened form.

There is, however, a machine-technical point which greatly enhances the relative advantage of the product-form inverse. Typically one has available a limited amount of high-speed memory ("core") and a virtually unlimited amount of slower back-up memory (disc or magnetic tape). If the problem is of small or medium size, i.e. if the size of the shortened tableau fits the capacity of the core alone, efficient choice of pivots and updating of the tableau at each step is not easily beaten. But once backing-store is involved we get a quite different story:

The back-up memory has to be accessed linearly and asking for only one element at the end of the tableau takes as much time - most of which is spent in waiting for the mechanical part of the back-up memory to move - as performing a routine calculation on each column, provided they are accessed in the order in which they are stored. The product-form inverse is ideally suited to that situation in that it is always accessed in the order in which it was generated. The one obvious disadvantage of the product-form algorithm is that the size of the product-form inverse increases with the number of steps and that number may be several times the number of variables. For this reason a product-form algorithm usually contains a provision for re-inversion. After a set number of steps the current vertex is re-created by a shorter route. Starting from the trivial basis the elements of $\underline{x}$, as far as they are in the (new) current list of basic variables are brought back in, in exchange for the slacks of binding restrictions, without any regard to sign of either the pivot or of any entry in the value column or the objective function.

This re-inversion-feature makes the gain in pure computational effort compared with the explicit inverse, or for that matter the ordinary simplex method (shortened tableau) largely illusory. The product-form wins because for problems with a large number of variables and a large number of restrictions, other methods are not nearly as well suited to accommodate the specific features of a linearly accessible backing-store device.

With a change in computer technology (e.g. silicon chips), we may well see a comeback of other versions of the simplex algorithm.

CHAPTER XIII

PARAMETRIC VARIAT' OF THE LP PROBLEM

13.1 The parametric variation problem

The term "parametric programming" is usually referred to as being due to Gass and Saati [12], [13]. The term "sensitivity analysis" appears to be originally due to G.B. Dantzig [8], section 12.4, who discusses the use of the optimal tableau for purposes of analysis, within the range of validity of the optimal vertex.

If there are minor changes in the originally specified problem, the optimal tableau tells us how this will affect the solution. Dantzig's treatment of the problem includes the analysis of changes in individual coefficients a_{ij}. This is a fairly complicated mathematical problem and it is not easily possible to tackle more than one coefficient at a time, except by computational means, i.e. solving the whole problem again. (See however, also section 2.18 for approximate solutions of this problem).

In this book we follow Garvin [11] and investigate changes in the right-hand side vector $\underline{b}$ and the preference-vector $\underline{w}'$, and include the analysis of adjoining vertices. That definition of-sensitivity analysis turns it into parametric programming.

As far as the mathematical and computational logic of the problem is concerned, sensitivity analysis is parametric programming. The term "sensitivity analysis" simply refers to an important field of application. It might also be called parametric postoptimality analysis. One has found an optimal and feasible solution to an LP program, and wishes to know how sensitive that solution is to variations in the coefficients.

As far as this relates to changes in the right-hand side or in the (linear component of the) objective function, parametric methods are a particularly efficient way of investigating this problem, compared with independently solving a series of closely related programming problems, each differing from the original one by a variation in a few coefficients.

The other main application is in solving mathematical programming problems for the first time. In that case, one first specifies a related problem for which an optimal and/or feasible solution is known or easy to obtain. One then solves this related problem, and then converts the solution of that problem by means of parametric adjustment, into the problem which one really wants to solve.

In the computational set-up of this book parametric variation is technically a re-entry of the Simplex Algorithm.

In that context it is obviously necessary that one can solve, with reasonable efficiency, the general case of a problem which is initially neither feasible nor optimal. One can then re-enter the Simplex Algorithm for additional parametric exercises, i.e. post-optimality analysis.

Parametric programming as a method of solving problems for the first time, has, however been recommended by Dantzig. (Linear programming and extensions [8] sections 11.3 and 11.4).

13.2 Parametric variation of the right-hand side of an L.P. problem

This parametric variation problem is defined as finding a range of solutions to the L.P. problem.

Maximise $\tau = \underline{w}'\underline{x}$ (7.2.1)

Subject to $A\underline{x} \leq \underline{b} + \lambda\underline{v}$ (13.2.1)

This problem differs from the "ordinary" linear programming problem as discussed in section 7.2, by the presence of the parametric variation vector. The vector $\underline{v}$ gives a particular direction of variation. Thus the constants of restrictions 3, 5, and 6 to be increased in the proportions 2, 1, and 5 would be indicated as $v_3 = 2$, $v_5 = 1$, $v_6 = 5$, other elements of $\underline{v}$ being zero. The parameter λ gives the extent of variation. For $\lambda = 0$ we have the originally specified problem. The actual change relative to this original problem is then $\lambda\underline{v}$. Obviously, λ is the parameter which gives the method its name.

We will now investigate how gradually increasing values of λ, starting with $\lambda = 0$, affect the solution of a previously optimal and feasible solution.

Recall sections (8.1) and (8.7) concerning the partitioned solution of a Simplex vertex, and again the summary of these results at the beginning of section 8.11.

Once a particular partitioning (vertex) has been established, we find that the optimality of a particular vertex is independent of the right-hand side vector $\underline{b}$. Reference to this vector does not occur in the updated form of the objective function, nor in the main body of the tableau, only in the value column.

Therefore, as long as the solution remains feasible, we find the solution-vector by substitution of $\underline{b} + \lambda\underline{v}$, for $\underline{v}$ into the appropriate expressions from section 8.7. To do this, we first express (13.2.1) in a partitioned form, the partitioning being understood to refer to A_{11} as block-pivot.

This partitioned expression is:

$$\left.\begin{array}{l} A_{11}\,\underline{x}_1 + A_{12}\,\underline{x}_2 \leq \underline{b}_1 + \lambda\underline{v}_1 \\ \\ A_{21}\,\underline{x}_1 + A_{22}\,\underline{x}_2 \leq \underline{b}_2 + \lambda\underline{v}_2 \end{array}\right) \qquad (13.2.2)$$

Substitution of $\underline{b}_1 + \lambda\underline{v}_1$ for $\underline{b}_1$

and $\underline{b}_2 + \lambda\underline{v}_2$ for $\underline{b}_2$ in the "value column" expressions in section 8.7 (or 11.1), yields the following results:

$$\underline{x}_1 = A_{11}^{-1}\,(\underline{b}_1 + \lambda\underline{v}_1) \qquad (13.2.3)$$

and

$$\underline{s}_2 = \underline{b}_2 + \lambda\underline{v}_2 - A_{21}A_{11}^{-1}\,(\underline{b}_1 + \lambda\underline{v}_1) \qquad (13.2.4)$$

These expressions are equivalent to

$$\underline{x}_1 = A_{11}^{-1}\,\underline{b}_1 + \lambda A_{11}^{-1}\underline{v}_1 \qquad (13.2.5)$$

and

$$\underline{s}_2 = \underline{b}_2 - A_{21}A_{11}^{-1}\,\underline{b}_1 + \lambda(\underline{v}_2 - A_{21}A_{11}^{-1}\underline{v}_1) \qquad (13.2.6)$$

The last terms on the right-hand sides of (13.2.5) and (13.2.6) are similar to the expressions which we obtain for the updated form of a non-basic variable.

We shall make use of this fact to obtain the parametric column i.e.

$$\begin{bmatrix} A_{11}^{-1}\underline{v}_1 \\ \\ \underline{v}_2 - A_{21}A_{11}^{-1}\underline{v}_1 \end{bmatrix}$$

as an ordinary updated column, representing a non-basic variable.

Example

Consider the following* linear programming problem

$$\text{Maximise} \quad \tau = 3x_1 + 7x_2 + 3x_3 + 7x_4 - 20x_5$$

$$\text{Subject to} \quad x_1 + x_2 - x_5 \leq 1$$

$$x_1 + 2x_2 - 3x_5 \leq 0$$

$$x_3 + x_4 - x_5 \leq 1$$

$$x_3 + 2x_4 - 3x_5 \leq 0$$

$$x_1 + 4x_2 + x_3 + 4x_4 - 12x_5 \leq -2$$

We first give the initial and the optimum tableau, (tableaux 13.2a and 13.2b, respectively).

TABLEAU 13.2 A

THE ORIGINAL PROBLEM OF A PARAMETRIC VARIATION EXAMPLE.

NAME	!!	X 1	X 2	X 3	X 4	X 5	!!	VALUE
S 1	!!	1	1	-	-	-1	!!	1
S 2	!!	1	2	-	-	-3	!!	0
S 3	!!	-	-	1	1	-1	!!	1
S 4	!!	-	-	1	2	-3	!!	0
S 5	!!	1	4	1	4	-12	!!	-2
T	!!	-3	-7	-3	-7	20	!!	-

TABLEAU 13.2 B

THE INITIAL OPTIMUM OF A PARAMETRIC VARIATION EXAMPLE.

NAME	!!	S 1	S 2	S 3	S 4	S 5	!!	VALUE
X 1	!!	1.00	0.50	-1.00	1.50	-0.50	!!	1.00
X 2	!!	1.00	-2	2	-3	1.00	!!	1.00
X 3	!!	-1.00	1.50	1	0.50	-0.50	!!	1.00
X 4	!!	2	-3.00	1.00	-2.00	1	!!	1.00
X 5	!!	1.00	-1.50	1.00	-1.50	0.50	!!	1.00
T	!!	1.00	1.00	1.00	1.00	1.00	!!	0

*This problem was taken from: A.R.G. Heesterman, Special Simplex Algorithm for Multi-Sector Problems [20] .

We now wish to know, what solutions would arise, when the constant of the s_4 - restriction was not the zero which was actually specified, but some non-zero (or more generally, a different) number instead. Clearly, for outward adjustment of the s_1-restriction, the non-updated variation-vector in (13.2.1) is

$$\underline{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

and for inward adjustment we have to change the sign of the unit vector. And we already know the updated form of that vector, i.e. the s_4 column of the tableau.

We write the equation equivalent of (13.2.1) as follows:

$$A\underline{x} + \underline{s} = \underline{b} + \lambda\underline{v} \qquad (13.2.7)$$

For the particular example, with $\underline{v}$ being the unit-vector corresponding to s_4, we write (13.2.7) as

$$\begin{aligned}
x_1 + x_2 - x_5 + s_1 &= 1 \\
x_1 + 2x_2 - 3x_5 + s_2 &= 0 \\
x_3 + x_4 - x_5 + s_3 &= 1 \\
x_3 + 2x_4 - 3x_5 + s_4 &= 0 + \lambda \\
x_1 + 4x_2 + x_3 + 4x_4 - 12x_5 + s_5 &= -2
\end{aligned}$$

The corresponding updated tableau is written explicitly, as follows:

$$\begin{aligned}
x_1 + s_1 + \tfrac{1}{2}s_2 - s_3 + 1\tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &= 1 + 1\tfrac{1}{2}\lambda \\
x_2 + s_1 - 2s_2 + 2s_3 - 3s_4 + s_5 &= 1 - 3\lambda \\
x_3 - s_1 + 1\tfrac{1}{2}s_2 + s_3 + \tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &= 1 + \tfrac{1}{2}\lambda \\
x_4 + 2s_1 - 3s_2 + s_3 - 2s_4 + s_5 &= 1 - 2\lambda \\
x_5 + s_1 - 1\tfrac{1}{2}s_2 + s_3 - 1\tfrac{1}{2}s_4 + \tfrac{1}{2}s_5 &= 1 - 1\tfrac{1}{2}\lambda
\end{aligned} \qquad (13.2.8)$$

with a corresponding objective function equation, i.e.

$$s_1 + s_2 + s_3 + s_4 + s_5 + \tau = 0 + \lambda$$

There appears to be an extension of the rule of the smallest quotient, which allows us to calculate a suitable value of λ.

We find the greatest positive value, for which all the basic variables remain positive, by taking the smallest quotient from the ratios of the positive entries in the value column, divided by minus the corresponding negative element in the variation vector presented in updated form.

We compare 1 : 3 for x_2, 1 : 2 for x_4, and 1 : $1\frac{1}{2}$ for x_5, and find $\lambda = 1/3$ for x_2 to be the smallest quotient.

Apparently, the vertex for which x_1, x_2, x_3, x_4 and x_5 are the basic variables is feasible for $\lambda \leq 1/3$.

At the upper end of this inverval, $\lambda = 1/3$, the vertex ceases to be feasible, but generally there will be a different feasible vertex which may be obtained in one step from the parametrically amended current one.

Thus, for $\lambda = 1/3$, the system becomes

$$
\begin{aligned}
x_1 &+ s_1 + \tfrac{1}{2}s_2 - s_3 + 1\tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &&= 1\tfrac{1}{2}\\
x_2 &+ s_1 - 2s_2 + 2s_3 - 3s_4 + s_5 &&= -0\\
x_3 &- s_1 + 1\tfrac{1}{2}s_2 + s_3 + \tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &&= 1\ 1/6\\
x_4 &+ 2s_1 - 3s_2 + s_3 - 2s_4 + s_5 &&= 1/3\\
x_5 &+ s_1 - 1\tfrac{1}{2}s_2 + s_3 - 1\tfrac{1}{2}s_4 + \tfrac{1}{2}s_5 &&= \tfrac{1}{2}
\end{aligned}
$$

with a corresponding updated objective function equation, i.e.

$$s_1 + s_2 + s_3 + s_4 + s_5 + \tau = 1/3$$

We consider the -0 for x_3 to be a negative number. When implementing parametric variation of the right-hand side in a practical computational context, one will put the corresponding tableau-entry at a small non-zero negative value, e.g. -0.0000001, and the normal "Phase I" part of the L.P. algorithm will be activated.

Typically, we need <u>one</u> normal Simplex step to eliminate such a small negative figure. At least, that is the case if an efficient version of the Simplex Algorithm is used.

Both the programmed procedure offered in section 12.3, LINP and the Simplex procedure offered later in this chapter, apply the dual ratio as criterion for selecting the pivotal column, when applied to a tableau in which there is only one violated restriction, while that one violated restriction can be made binding with help of several alternative pivot column-variables, all of them being "non-preferred", i.e. leading to loss of preference-value.

We now make the steps. We first adjust the value column. This is the actual parametric step, or, by contrast to variation of the objective function, the primal parametric step. We now obtain tableau 13.2c.

TABLEAU 13.2 C

DISPLACED OPTIMUM OF THE SAME PARAMETRIC VARIATION EXAMPLE.

NAME	S 1	S 2	S 3	S 4	S 5	VALUE (L=0.33)
X 1	1.00	0.50	-1.00	1.50	-0.50	1.50
X 2	1.00	-2	2	(-3)	1.00	-0.00
X 3	-1.00	1.50	1	0.50	-0.50	1.17
X 4	2	-3.00	1.00	-2.00	1	0.33
X 5	1.00	-1.50	1.00	-1.50	0.50	0.50
T	1.00	1.00	1.00	1.00	1.00	0.33

We then need an ordinary step, to obtain a new optimal and feasible vertex obtaining tableau 13.2d.

TABLEAU 13.2 D

THE SAME VERTEX AS IN TABLEAU 13.2 C, NOW PRESENTED AGAIN IN OPTIMAL AND FEASIBLE FORM.

NAME	S 1	S 2	S 3	X 2	S 5	VALUE (L=0.33)
X 1	1.50	-0.50	-	0.50	0.00	1.50
S 4	-0.33	0.67	-0.67	-0.33	-0.33	0.00
X 3	-0.83	1.17	1.33	0.17	-0.33	1.17
X 4	1.33	-1.67	-0.33	-0.67	0.33	0.33
X 5	0.50	-0.50	0.00	-0.50	-0.00	0.50
T	1.33	0.33	1.67	0.33	1.33	0.33

If the constant of the s_4-restriction is increased beyond the value $b_4 = 1/3$, this only leads to an increase in the value of the s_4 slack-variable, and nothing else happens. In this case, the possibility to increase the parametric variable λ ad infinitum without reducing any variable to zero, arose because the variation-direction was to increase the constant of a single restriction, and that restriction was no longer binding. It can also arise because change of the right-hand side is matched by a combination of basic variables. We may illustrate this case by using the same unit-vector as variation-vector, but in the opposite direction.

The initial result of a reduction of b_4 below zero is that at $\lambda = -2/3$, x_1 ceases to be positive.

The optimal tableau has then become:

TABLEAU 13.2 E

LEFT-DISPLACED OPTIMUM OF THE SAME PARAMETRIC VARIATION EXAMPLE.

NAME	!!	S 1	S 2	S 3	S 4	S 5	!!	VALUE	(L=-0.67)
X 1	!!	1	0.50	(-1)	1.50	-0.50	!!	-0.00	
X 2	!!	1	-2	2	-3	1	!!	3	
X 3	!!	-1	1.50	1	0.50	-0.50	!!	0.67	
X 4	!!	2	-3	1	-2	1	!!	2.33	
X 5	!!	1	-1.50	1	-1.50	0.50	!!	2	
T	!!	1	1	1	1	1	!!	-0.67	

To eliminate x_1 without losing optimality, we need to bring s_3 into the list of basic variables, and we obtain our next tableau:

TABLEAU 13.2 F

THE SAME VERTEX AS IN TABLEAU 13.2 E, NOW PRESENTED AGAIN IN OPTIMAL AND FEASIBLE FORM.

NAME	!	S 1	S 2	X 1	S 4	S 5	!!	VALUE	(L=-0.67)
S 3	!	-1	-0.50	-1	-1.50	0.50	!!	0	
X 2	!	3	-1	2	-	0	!!	3	
X 3	!	-0	2	1	2	-1	!!	0.67	
X 4	!	3	-2.50	1	-0.50	0.50	!!	2.33	
X 5	!	2	-1	1	0	0	!!	2	
T	!	2	1.50	1	2.50	0.50	!!	-0.67	

We will now wish to investigate the implications of even more negative values of λ. To this end it is useful to re-define the λ-variable. Or rather, we express the original variable as the sum of the already attained value and any further change

$$\lambda = \lambda_o + \lambda_1 \qquad (13.2.9)$$

for $\lambda = -2/3$ (13.2.8) is now rewritten as:

$$\begin{aligned}
x_1 &+ s_1 + \tfrac{1}{2}s_2 - s_3 + 1\tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &&= -0 + 1\tfrac{1}{2}\lambda_1\\
x_2 &+ s_1 - 2s_2 + 2s_3 - 3s_4 + s_5 &&= 3 - 3\lambda_1\\
x_3 &- s_1 + 1\tfrac{1}{2}s_2 + s_3 + \tfrac{1}{2}s_4 - \tfrac{1}{2}s_5 &&= 2/3 + \tfrac{1}{2}\lambda_1\\
x_4 &+ 2s_1 - 3s_2 + s_3 - 2s_4 + s_5 &&= 2\ 1/3 - 2\lambda_1\\
x_5 &+ s_1 - 1\tfrac{1}{2}s_2 + s_3 - 1\tfrac{1}{2}s_4 + \tfrac{1}{2}s_5 &&= 2 - 1\tfrac{1}{2}\lambda_1\\
& s_1 + s_2 + s_3 + s_4 + s_5 + \tau &&= -2/3 + \lambda_1
\end{aligned}$$

The updating rules for Simplex tableaux were derived from the rules for solving systems of equations by elimination of the variable associated with the pivotal column from all other rows. They are therefore equally applicable to the parametric problem.

Thus, we could write our tableaux for the last parametric step as in tableau 13.2g
(To distinguish the parametric steps from ordinary steps, we mark the "parametric-pivots" by a square.)

TAELEAU 13.2 G

TAELEAU-PRESTENTATION OF A PARAMETRIC STEP

NAME	!	S 1	S 2	S 3	S 4	S 5	!!	-L=0 VALUE	-L0
X 1	!	1	0.50	-1	1.50	-0.50	!!	1	1.50
X 2	!	1	-2	2	-3	1	!!	1	-3
X 3	!	-1	1.50	1	0.50	-0.50	!!	1	0.50
X 4	!	2	-3	1	-2	1	!!	1	-2
X 5	!	1	-1.50	1	-1.50	0.50	!!	1	-1.50
T	!	1	1	1	1	1	!!	0	1

This presentation allows application of the usual rule of the smallest quotient, to determine the leaving variable, which is in this case x_1.

Now re-adjust right-hand side and redefine parameter, to obtain tableau 13.2h.

TABLEAU 13.2 H

DISPLACED OPTIMUM, WITH PARAMETRIC COLUMN.

NAME	!	S 1	S 2	S 3	S 4	S 5	!!	-L=0.67 VALUE	-L1
X 1	!	1	0.50 (-1)		1.50	-0.50	!!	-0.00	1.50
X 2	!	1	-2	2	-3	1	!!	3	-3
X 3	!	-1	1.50	1	0.50	-0.50	!!	0.67	0.50
X 4	!	2	-3	1	-2	1	!!	2.33	-2
X 5	!	1	-1.50	1	-1.50	0.50	!!	2	-1.50
T	!	1	1	1	1	1	!!	-0.67	1

Make one ordinary step, introducing s_3 as a basic variable to eliminate the now negative variable x_1. The next tableau ready for a new parametric step is as follows:

TABLEAU 13.2 I

ADJUSTED VERTEX, READY FOR THE NEXT PARAMETRIC STEP.

NAME	!	S 1	S 2	X 1	S 4	S 5	!!	-L=0.67 VALUE	-L1
S 3	!	-1	-0.50	-1	-1.50	0.50	!!	0	-1.50
X 2	!	3	-1	2	-	0	!!	3	-
X 3	!	-	2	1	2	-1	!!	0.67	[2]
X 4	!	3	-2.50	1	-0.50	0.50	!!	2.33	-0.50
X 5	!	2	-1	1	0	0	!!	2	0
T	!	2	1.50	1	2.50	0.50	!!	-0.67	2.50

The lowest ratio at which the next basic variable is eliminated by further change in the parameter is now obtained for x_3 at $-\lambda_1 = 0.67/2 = 0.33$.

We now apply the same logic again. We set the attained value of the parameter at

$$\lambda = \lambda_o + \lambda_1 = -0.67 - 0.33 = -1$$

and indicate any further change in the parameter as

$$\lambda_2 = \lambda - \lambda_o - \lambda_1 = \lambda + 1$$

After this adjustment of the right-hand side, the tableau becomes

TABLEAU 13.2 J

DISPLACED OPTIMUM, RELATIVE TO TABLEAU 13.2 I

NAME	S 1	S 2	X 1	S 4	S 5	-L=1 VALUE	-L2
S 3	-1	-0.50	-1	-1.50	0.50	0.50	-1.50
X 2	3	-1	2	-	0	3	-
X 3	-	2	1	2	(-1)	-0.00	2
X 4	3	-2.50	1	-0.50	0.50	2.50	-0.50
X 5	2	-1	1	0	0	2	0
T	2	1.50	1	2.50	0.50	-1.50	2.50

We now make another "ordinary" step, introducing s_5 as a basic variable against x_3. The tableau now becomes:

TABLEAU 13.2 K

NEXT FOLLOWING VERTEX, READY FOR THE PARAMETRIC STEP.

NAME	S 1	S 2	X 1	S 4	X 3	-L=1 VALUE	-L2
S 3	-1	0.50	-0.50	-0.50	0.50	0.50	-0.50
X 2	3	-1	2	0	0	3	0
S 5	0	-2	-1	-2	-1	0	-2
X 4	3	-1.50	1.50	0.50	0.50	2.50	[0.50]
X 5	2	-1	1	0	0	2	0
T	2	2.50	1.50	3.50	0.50	-1.50	3.50

The length of the parametric step is rather greater this time, i.e. $-\lambda_2 = 2\frac{1}{2}/\frac{1}{2} = 5$.

Thus, we subtract the λ_2 column from the value column with a factor 5. The next tableau with x_4 just negative, is:

TABLEAU 13.2 L

THE LAST OF THE LEFT-ADJUSTED OPTIMA.

NAME	!	S 1	S 2	X 1	S 4	X 3	!!	-L=6 VALUE	-L3
S 3	!	-1	0.50	-0.50	-0.50	0.50	!!	3	-0.50
X 2	!	3	-1	2	0	0	!!	3	0
S 5	!	0	-2	-1	-2	-1	!!	10	-2
X 4	!	3	-1.50	1.50	0.50	0.50	!!	-0.00	0.50
X 5	!	2	-1	1	0	0	!!	2	0
T	!	2	2.50	1.50	3.50	0.50	!!	-19	3.50

Introducing s_2 as a basic variable to eliminate the negative x_4-variable, the next tableau, ready for further parametric adjustment (if possible), is:

TABLEAU 13.2 M

AN UNBOUNDED PARAMETRIC TABLEAU.

NAME	!	S 1	X 4	X 1	S 4	X 3	!!	-L=6 VALUE	-L3
S 3	!	0	0.33	0	-0.33	0.67	!!	3	-0.33
X 2	!	1	-0.67	1	-0.33	-0.33	!!	3	-0.33
S 5	!	-4	-1.33	-3	-2.67	-1.67	!!	10	-2.67
S 2	!	-2	-0.67	-1	-0.33	-0.33	!!	0	-0.33
X 5	!	0	-0.67	0	-0.33	-0.33	!!	2	-0.33
T	!	7	1.67	4	4.33	1.33	!!	-19	4.33

Except for the fancyhigh artificial upper limits (see section 10.3), reference to which has been suppressed, the parametric variation problem is now unbounded. Further increase in the value of $-\lambda$ i.e. λ falling below $\lambda = -6$, leads only to indefinite increases in all basic variables, no basic variable being driven towards zero.

Our researches into the effects of changes in b_4 on the optimal solution may be summarised in a graphical mapping.

We plot the value of the objective function τ as a function of the parametric variable λ. Each vertex is represented by a segment of uniform slope, i.e. a straight line. The slope of this line is the amount of change in the value of the objective function per unit of change of the parametric variable.

It is characteristic of the class of convex problems that these segments form a continuous "curve" with a mountain-shape. The linear combination of any two points of the curve generally lies below the curve itself.

For the LP problem, we may state (and prove) this property also in algebraic terms. To do this we first need to express the notion of a combination algebraically. Consider the range of optimal solution values for values of λ within the interval $\lambda^* \leq \lambda \leq \lambda^{**}$. We express the notion that λ is a convex combination of λ^* and λ^{**} as

$$\lambda = p\lambda^* + (1-p)\lambda^{**} \qquad (0 \leq p \leq 1)$$

For a particular value of p (and hence of λ) the value of the optimal solution is

$$\tau(\lambda) = \underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + \lambda \underline{v}_1) = \underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + p\lambda^* \underline{v}_1 + (1-p)\lambda^{**}\underline{v}_1)$$

$$= p(\underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + \lambda^* \underline{v}_1)) + (1-p)(\underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + \lambda^{**}\underline{v}_1)) \qquad (13.2.10)$$

We assume that the LP problem is non-empty for all values of λ in the range $\lambda^* \leq \lambda \leq \lambda^{**}$.

If the same vertex is applicable for all values of λ within the interval we obviously have

$$\tau(\lambda) = p\tau(\lambda^*) + (1-p)\tau(\lambda^{**}) \qquad (13.2.11)$$

the expressions within brackets $(\underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + \lambda^* \underline{v}_1))$ and $(\underline{w}'_1 A_{11}^{-1}(\underline{b}_1 + \lambda^{**}\underline{v}_1))$ being nothing else but $\tau(\lambda^*)$ and $\tau(\lambda^{**})$. In that case we are on a straight section, i.e. (13.2.11) describes $\tau(\lambda)$ as a linear function of p and hence of λ.

The more general relationship is (the notion of the mountain-shape)

$$\tau(\lambda) \geq p\tau(\lambda^*) + (1-p)\tau(\lambda^{**}) \qquad (13.2.12)$$

To prove this we invoke the Duality Theorem, i.e. we note that, irrespective of the value of λ,

$$\left.\begin{aligned} \underline{u}'_1 &= -\underline{w}'_1 A_{11}^{-1} \\ \underline{u}'_2 &= 0 \end{aligned}\right) \qquad (13.12.13)$$

is a feasible solution of the dual problem minimise $(\underline{b}' + \lambda\underline{v}')\underline{u}$, subject to $A'\underline{u} \geq \underline{w}$, $\underline{u} \geq 0$. It follows that the optimal and feasible solution of both the primal and dual problems obey the restriction

$$\tau(\lambda) \leq \underline{w}_1' A_{11}^{-1}(\underline{b}_1 + \lambda\underline{v}_1) \qquad (13.2.14)$$

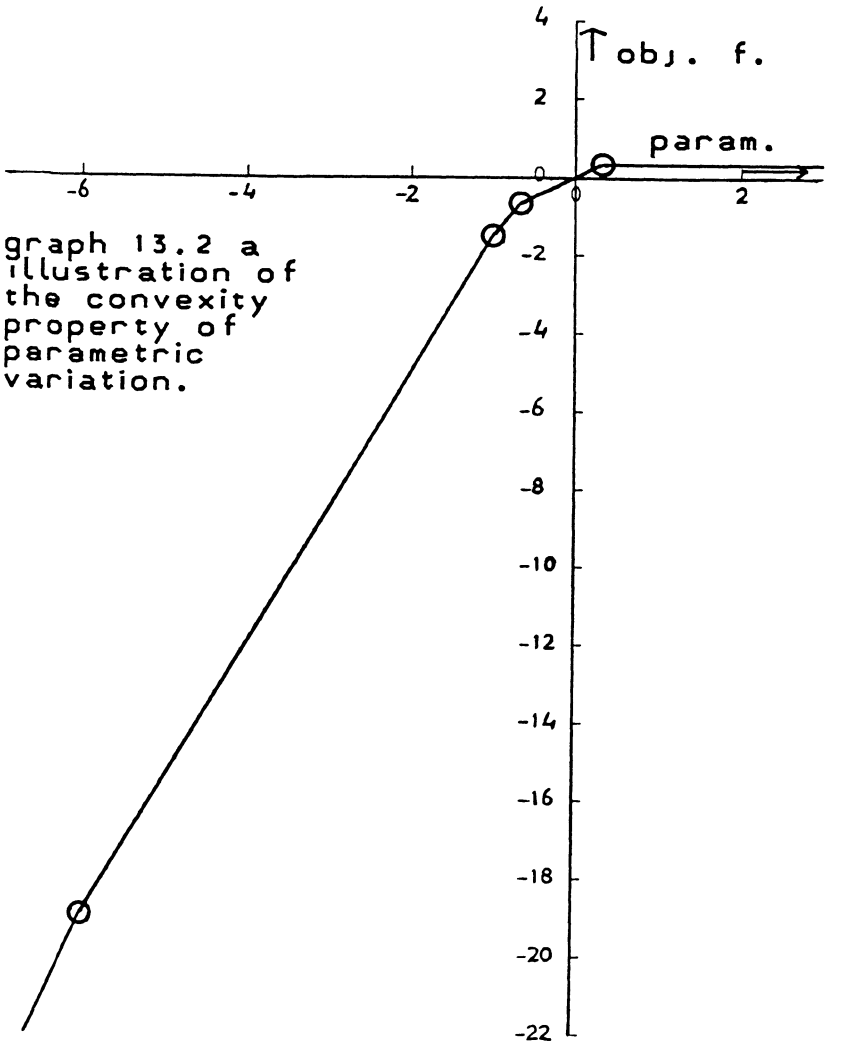
where the = equality sign will hold if A_{11} gives the appropriate partitioning and in general the < inequality sign will hold if, for the value of λ which is currently chosen, A_{11} is not associated with a feasible solution.

Combining (13.2.10) (applied for the initially chosen value of λ (e.g. "in the middle") with (13.2.14) applied for $\lambda = \lambda^*$ and $\lambda = \lambda^{**}$, we find confirmation of (13.3.12).
q.e.d.

We now illustrate the property by way of graphical mapping of the relationship between τ and b_4 in the example used before.

This has been done in graph 13.2a. The points marked in this graph with circles correspond to the vertices where, at certain critical values of λ, a change of basis took place. Note, that the scales of the axes have been adjusted by a factor two, to accommodate the point $\lambda = -6$, $\tau = -19$.

The objective function mapped in relation to changes in b_4.



graph 13.2 a illustration of the convexity property of parametric variation.

In the above example, the parametric variation problem became eventually unbounded in both directions. There are only a finite number of vertices in any LP problem, hence the parametric search operation must sooner or later fail, as it did in this example, or else an abnormal exit of the LP algorithm must be activated on re-entry. For parametric variation of the right-hand side that abnormal exit of the LP algorithm itself arises when an empty problem has been generated.

Suppose, for example, that we had added an additional sixth restriction to the demonstration problem, i.e.

$$x_3 + 2x_4 - 3x_5 \geq -10$$

$$(-x_3 - 2x_4 + 3x_5 \leq 10)$$

Then obviously, for $\lambda = -10$, the now endless bottom left-hand segment of the graph would end.

Thus, the segment with s_2, s_3, s_5, x_2 and x_5 as the basic variables, would run from $\lambda = -10$ to $\lambda = -6$, instead of from $\lambda = -\infty$ to $\lambda = -6$ as is the case, without this additional restriction. For $\lambda < 10$, the restrictions

$$x_3 + 2x_4 - 3x_5 \geq -10$$

and

$$x_3 + 2x_4 - 3x_5 \leq b_4 = \lambda$$

would come to contradict each other.

If an infeasible solution is generated, there is no point in exploring even higher values of λ. Finding a feasible solution after increasing λ still further would in that case contradict the convexity property of the set of vectors $\underline{x}$, λ obeying (13.2.1). It follows that once we find the limit of the feasible space area all problems characterised by even higher values of λ are empty problems.

The case of generating an unbounded ordinary LP problem, after having found that the originally specified problem has a finite optimal and feasible solution - does not arise. It would imply an empty dual problem and finding a finite optimum for the originally specified problem proves the dual (of which only the objective function changes with λ) to be non-empty.

If parametric variation is implemented in conjunction with the linear programming procedure discussed in section 12.3, one would normally end up with an empty problem, rather than the parametric search operation itself failing. The reason is that all variables are assumed to have implicit upper limits and if they are also sign restricted the problem is bounded in all directions.

One would need a special instruction to signal the fact that a fancy-high upperbound had become binding and that the problem was in an intrinsic sense unbounded.

Exercise

Repeat the operations, performed as example in this section for changes in the constant of the fourth restriction, for the first restriction, i.e. replace the originally specified first restriction by $x_1 + x_2 - x_5 \leq 1 + \lambda$.

13.3 Parametric variation of the objective function in an L.P. problem

We will indicate the originality specified objective function as $\tau(\underline{x})$ as before, the parametric component as $\tau^*(\underline{x})$, and the total value of the objective function as $\tau^{**}(\underline{x})$. We are then concerned with finding solutions, for various values of the parameter λ, to the following problem:

Maximise

$$\tau^{**}(\underline{x}) = \tau(\underline{x}) + \tau^*(\underline{x}) = \underline{w}'\underline{x} + \lambda\underline{g}'\underline{x} \qquad (13.3.1)$$

Subject to

$$A\,\underline{x} \leq \underline{b} \qquad (\underline{x} \geq 0) \qquad (7.2.2)$$

In (13.3.1) $\underline{g}'$ is the parametric variation vector for the objective function, e.g. $g_1=1$ and other elements of $\underline{g}$ at zero would mean that we investigate the sensitivity of the solution to changes in w_1. Application of the Duality Theorem converts this problem into the following dual problem

Maximise

$$\Phi = -\,\underline{b}'\,\underline{u} \qquad (13.3.2)$$

Subject to

$$-A\,\underline{u} \leq -\underline{w} - \lambda\underline{g} \qquad (13.3.3)$$

Apparently we can, with the appropriate adjustments of the signs, use the results of the previous section 13.2 to obtain solutions for various alternative objective functions.

If the right-hand side is changed, there generally is a range of variation within which the same vertex stays valid and only the solution vector is adjusted. The end of this range of parametric variation is defined by that vertex ceasing to be feasible.

If the objective function is changed, there is a range of variation, within which the same vertex stays valid, without any change in anything else other than the updated form of the objective function row.

The end of that range of parametric variation is in this case defined by the problem ceasing to be optimal at that vertex.

After a (primal) parametric variation step of the right-hand side one obtains an adjacent vertex, if there is one, by re-entering the Phase I part of the L.P. algorithm.

In the case of parametric variation of the objective function, one obtains the adjacent vertex (if there is one), by re-optimizing.

One could, if one wished, solve the problem of finding the whole family of solutions for a particular dimension of change of the objective function, by actually punching the dual problem and solving the family of problems associated with the corresponding variation of what then becomes the right-hand side. But it is more convenient to perform search operations on a parametric row instead.

Example

We take the same demonstration example as in the previous section. We now carry an additional parametric row rather than an additional parametric column. And we assume that we wish to analyze the sensitivity of the solution for changes in w_1.

Accordingly, our initial equations-system is:

$$
\begin{aligned}
x_1 + x_2 - x_5 + s_1 &= 1\\
x_1 + 2x_2 - 3x_5 + s_2 &= 0\\
x_3 + x_4 - x_5 + s_3 &= 1\\
x_3 + 2x_4 - 3x_5 + s_4 &= 0\\
x_1 + 3x_2 + x_3 + 4x_4 - 12x_5 + s_5 &= -2\\
-3x_1 - 7x_2 - 3x_3 - 7x_4 + 20x_5 + \tau^{**} &= 0\\
-x_1 + \tau^{*} &= 0
\end{aligned}
$$

This system is tabulated in the usual way, see tableau 13.3a.

TABLEAU 13.3 A

SET-UP TABLEAU FOR PARAMETRIC VARIATION OF THE OBJECTIVE-FUNCTION COEFFICIENT FOR X 1.

NAME	X 1	X 2	X 3	X 4	X 5	VALUE
S 1	1	1	-	-	-1	1
S 2	1	2	-	-	-3	0
S 3	-	-	1	1	-1	1
S 4	-	-	1	2	-3	0
S 5	1	4	1	4	-12	-2
T**	-3	-7	-3	-7	20	-
T*	-1	-	-	-	-	0

The initial value of the parameter is again zero and the corresponding initial optimal tableau is tableau 13.3b.

We mark the dual parametric step in a way which is broadly similar to the primal parametric step discussed in the previous section 13.2. We put a square marker ☐ on the critical element of the τ^* row, the element which first causes the corresponding figure in the τ^{**} row to become negative.

TABLEAU 13.3 B

ILLUSTRATION OF THE DUAL PARAMETRIC STEP.

NAME	S 1	S 2	S 3	S 4	S 5	VALUE	
X 1	1	0.50	-1	1.50	-0.50	1	
X 2	1	-2	2	-3	1	1	
X 3	-1	1.50	1	0.50	-0.50	1	
X 4	2	-3	1	-2	1	1	
X 5	1	-1.50	1	-1.50	0.50	1	
T**	1	1	1	1	1	0	(L=0)
T*	1	0.50	☐ -1	1.50	-0.50	1	

This tableau indicated that τ^* row can be added to the complete objective-function row, with a factor of not more than 1. for $\lambda = 1$, the shadow price of s_3 changes sign.

Accordingly we make a parametric step, and obtain the following no longer optimal tableau:

TABLEAU 13.3 C

ILLUSTRATION OF THE ORDINARY STEP WHICH FOLLOWS AFTER A DUAL PARAMETRIC STEP.

NAME	!	S 1	S 2	S 3	S 4	S 5	!	VALUE	
X 1	!	1	0.50	-1	1.50	-0.50	!	1	
X 2	!	1	-2	②	-3	1	!	1	
X 3	!	-1	1.50	1	0.50	-0.50	!	1	
X 4	!	2	-3	1	-2	1	!	1	
X 5	!	1	-1.50	1	-1.50	0.50	!	1	
T**	!	1	1	-0.00	1	1	!	1	(L=1)
T*	!	1	0.50	-1	1.50	-0.50	!	1	

At this point it is useful to mention that the computational implementation of parametric programming offered in this book includes a call to the ordering procedure discussed in section 12.3. Until now, examples have often been un-ordered by file-editing or by suppressing the call to the ordering procedure.

This will not be done anymore now, and the next tableau, re-ordered according to codes, is tableau 13.3d

TABLEAU 13.3 D

THE SECOND DUAL PARAMETRIC STEP.

NAME	!	X 2	S 1	S 2	S 4	S 5	!	VALUE	
X 1	!	0.50	1.50	-0.50	-	0	!	1.50	
X 3	!	-0.50	-1.50	2.50	2	-1	!	0.50	
X 4	!	-0.50	1.50	-2	-0.50	0.50	!	0.50	
X 5	!	-0.50	0.50	-0.50	-	-0	!	0.50	
S 3	!	0.50	0.50	-1	-1.50	0.50	!	0.50	
T**	!	0	2	1.50	2.50	0.50	!	1	(L=1)
T*	!	0.50	1.50	[-0.50]	-	0	!	1.50	

This tableau indicates that the full objective function has attained a maximum value of $\tau^{**} = 1$, while the parametric component is not at its maximum value.

The full objective function is (for $\lambda = 1$), to maximise $\tau^{**} = 4x_1 + 7x_2 + 3x_3 + 7x_4 - 20x_5$, and this function attains a maximum value of 1 for $x_1 = 1\frac{1}{2}$, $x_2 = 0$, $x_3 = x_4 = x_5 = \frac{1}{2}$.

The parametric component, i.e. to maximise x_1, is not at its maximum as may be seen from the negative entry in the τ^*/s_2 cell.

The next parametric step has been marked ☐
For $\lambda = 1 + \lambda_1 = 1 + 3 = 4$, the shadowprice of s_2 will change sign and a new ordinary optimizing step will be needed to restore optimality.

Just as in the case of parametric variation of the right-hand side, a series of parametric steps may end in two ways.

If the parametric component of the objective function is at a maximum at the current vertex, i.e. there are no negative entries in the τ^* row, the search for a column to be marked ☐ as the next incoming variable will fail.

13.4 Parametric adjustment of mixed systems

In this section we survey the complications which arise if parametric change/sensitivity analysis is practised in conjunction with a system which contains equations as well as inequalities, variables with and without non-negativity restrictions.

First of all we must clarify the definition of a parametric variation step.

The parametric variation step finds the smallest (change in the) value of the parameter, which causes the current vertex to be no longer optimal and feasible.

Therefore, a parametric change of the right-hand side vector $\underline{b}$, which causes a variable to which no non-negativity restriction applies, to become negative, it not the end of a parametric step.

Example

Consider again the problem we first introduced in section 13.2, but let us now assume that x_1 is an "absolute" or "free" variable, whereas x_2, x_3, x_4 and x_5 are restricted to non-negative values only.

In that-case x_1 may not be eliminated from the list of basic variables, and the original vertex stays valid until another variable than x_1 is eliminated. We again consider the parametric reduction of the constant of the fourth restriction. The resulting parametric step has been marked in tableau 13.4a.

TABLEAU 13.4 A

X1 IS NOT ELIGEABLE AS (PARAMETRIC) PIVOTAL ROW.

NAME	!	S 1	S 2	S 3	S 4	S 5	!!	(L=0) VALUE	-L0
X 1	!	1	0.50	-1	1.50	-0.50	!!	1	1.50
X 2	!	1	-2	2	-3	1	!!	1	-3
X 3	!	-1	1.50	1	0.50	-0.50	!!	1	[0.50]
X 4	!	2	-3	1	-2	1	!!	1	-2
X 5	!	1	-1.50	1	-1.50	0.50	!!	1	-1.50
T	!	1	1	1	1	1	!!	0	1

Here x_1 is allowed to become negative, and the length of the parametric step is instead determined by the non-negativity of x_3. Implementation of this step leads to tableau 13.4b

TABLEAU 13.4 B

DISPLACED OPTIMUM WITH A NEGATIVE
NON-ZERO VALUE FOR A FREE VARIABLE.

NAME	!	S 1	S 2	S 3	S 4	S 5	!!	(-L=2) VALUE	-L1
X 1	!	1	0.50	-1	1.50	-0.50	!!	-2	1.50
X 2	!	1	-2	2	-3	1	!!	7	-3
X 3	!	(-1)	1.50	1	0.50	-0.50	!!	-0.00	0.50
X 4	!	2	-3	1	-2	1	!!	5	-2
X 5	!	1	-1.50	1	-1.50	0.50	!!	4	-1.50
T	!	1	1	1	1	1	!!	-2	1

As before, an ordinary step will be need to find a new optimal and feasible solution. The x_3-variable is eliminated and x_1 stays at its (negative) value of -2.

TABLEAU 13.4 C

NEW VERTEX (WITH NEGATIVE-VALUED FREE VARIABLE)
THE PARAMETRIC PROBLEM IS NOW UNBOUNDED. (RE-ORDERED)

NAME	!	X 3	S 2	S 3	S 4	S 5	!!	(-L=2) VALUE	-L1
X 1	!	1	2	-	2	-1	!!	-2	2
X 2	!	1	-0.50	3	-2.50	0.50	!!	7	-2.50
X 4	!	2	-	3	-1	-	!!	5	-1
X 5	!	1	-	2	-1	-	!!	4	-1
S 1	!	-1	-1.50	-1	-0.50	0.50	!!	0	-0.50
T	!	1	2.50	2	1.50	0.50	!!	-2	1.50

At that vertex, the parametric variation problem becomes unbounded, unless upper limits on the variables are considered.

This vertex is valid for all values of $\lambda < -2$.

The same logic applies to equations in the case of parametric variation of the objective function.

The shadow-prices of equations are allowed to change sign. Therefore the same vertex stays optimal, until the shadow-price of a non-negativity restriction or an inequality changes sign.

This restrictive definition of the parametric step may imply a loss of information, compared to what the user would like to get out of parametric sensitivity analysis.

One may have defined a variable as not restricted in sign, but would nevertheless wish to see a record of the values of other variables at the point where a particular variable changes sign.

Similarly, one may have defined a restriction as an equation, but wish to record the shadow-prices of other restrictions associated with the value of the parameter, at the point where the shadow-price of an equation changes sign. One could obviously solve this problem by defining the parametric step in terms of a variable (or shadow-price) attaining a zero value. Definitions are to some extent a matter of convention and convenience not of a priori logic.

The computational implementation of parametric sensitivity analysis (to be discussed in the next section), uses the more restrictive definition given above, and it is useful to mention that this problem may be circumvented.

If one wants a record of a variable to become zero, one should declare it as a sign-restricted variable, and declare minus the same variable as another variable, on the lines of the opening paragraph of section 10.1. The same approach may be followed with respect to equations.

Example

Consider again the same problem, where it it now assumed that the third restriction is an equation, but that we wish to keep track of any change in the sign of its shadow-price during parametric sensitivity analysis. The direction of parametric adjustment is assumed to be the same, i.e. increased preference for x_1.

We re-formulate the original problem, as summarized in tableau 13.4d.

TABLEAU 13.4 D

SET-UP TABLEAU FOR PARAMETRIC VARIATION OF THE OBJECTIVE-FUNCTION COEFFICIENT FOR X 1, WITH EQUATION-DUPLICATION OF THE S3 RESTRICTION.

NAME	X 1	X 2	X 3	X 4	X 5	VALUE
S 1	1	1	-	-	-1	1
S 2	1	2	-	-	-3	0
S 3	-	-	1	1	-1	1
S 4	-	-	-1	-1	1	-0.999
S 5	-	-	1	2	-3	0
S 6	1	4	1	4	-12	-2
T	-3	-7	-3	-7	20	-
T*	-1	-	-	-	-	0

Restrictions 3 and 4, i.e.

$$x_3 + x_4 - x_5 \leq 1 \text{ and}$$

$$x_3 + x_4 - x_5 \geq 0.999$$

amount for practical purposes to

$$x_3 + x_4 - x_5 = 1$$

i.e. the old third restriction re-defined as an equation. The new restrictions 5 and 6 are the old restrictions 4 and 5.

It is necessary to put a small, technically non-zero margin between the two restrictions. If rows 3 and 4 are exactly the same except in sign, a small calculation error may lead to an empty problem.

Thus $$x_3 + x_4 - x_5 \leq 1 \text{ and}$$

$$-x_3 - x_4 - x_5 \leq -1.000000000000001$$

are contradicting restrictions. Errors of this order of magnitude would easily arise from rounding and the problem might be found empty. The figure of 0.001 for the margin as chosen in this example is still unduly high. It was taken in order to highlight the fact that it is there at all. In practice, a much smaller margin will be sufficient.

The corresponding central optimal and feasible solution-tableau is given below with the first parametric adjustment step, i.e. the shadow-price of the third restriction indicated in the usual way □

TABLEAU 13.4 E

CENTRAL OPTIMUM, WITH DUAL PARAMETRIC STEP.

NAME	!	S 1	S 2	S 3	S 5	S 6	!!	VALUE
X 1	!	1	0.50	-1	1.50	-0.50	!!	1
X 2	!	1	-2	2	-3	1	!!	1
X 3	!	-1	1.50	1	0.50	-0.50	!!	1
X 4	!	2	-3	1	-2	1	!!	1
X 5	!	1	-1.50	1	-1.50	0.50	!!	1
S 4	!	-	-	1	-	-	!!	0.001
T	!	1	1	1	1	1	!!	0
T*	!	1	0.50	[-1]	1.50	-0.50	!!	1

The parametric adjustment step is now made and we obtain a tableau which requires re-optimization, tableau 13.4f.

TABLEAU 13.4 F

EQUATION-DUPLICATION SIGNALS THE POINT AT WHICH THE SHADOWPRICE CHANGES SIGN, THEN LEADS TO A ZERO LENGTH ORDINARY STEP.

NAME	!	S 1	S 2	S 3	S 5	S 6	!!	VALUE	
X 1	!	1	0.50	-1	1.50	-0.50	!!	1	
X 2	!	1	-2	2	-3	1	!!	1	
X 3	!	-1	1.50	1	0.50	-0.50	!!	1	
X 4	!	2	-3	1	-2	1	!!	1	
X 5	!	1	-1.50	1	-1.50	0.50	!!	1	
S 4	!	-	-	(1)	-	-	!!	0.001	
T*	!	2	1.50	-0.00	2.50	0.50	!!	1	(L=1)
T**	!	1	0.50	-1	1.50	-0.50	!!	1	

Note again that, whereas the figure of 0.001 has been used to emphasis that there is no actual degeneracy, one should think in terms of an ε-difference, hence the heading of tableau 13.4f, which speaks of a step of zero length.

The "ordinary" step which is now needed to regain optimality has not much intrinsic significance, but signals the fact that the shadow-price of the original third restriction has changed sign. Rounded to two decimals, the tableaux 13.4f and 13.4g below, are only distinguishable in the s_3/s_4 columns.

TABLEAU 13.4 G

NEW OPTIMAL AND FEASIBLE SOLUTION, WITH ONLY AN INSIGNIFICANTLY SMALL CHANGE IN THE VALUE COLUMN.

NAME	!	S 1	S 2	S 4	S 5	S 6	!! VALUE	
X 1	!	1	0.50	1	1.50	-0.50	!! 1.001	
X 2	!	1	-2	-2	-3	1	!! 0.998	
X 3	!	-1	1.50	-1	0.50	-0.50	!! 0.999	
X 4	!	2	-3	-1	-2	1	!! 0.999	
X 5	!	1	-1.50	-1	-1.50	0.50	!! 0.999	
S 3	!	-	0	1	0	-0	!! 0.001	
T**	!	2	1.50	0	2.50	0.50	!! 1.001	(L=1)
T*	!	1	0.50	1	1.50	-0.50	!! 1.001	

At this point s_6 is the next variable for which the shadow-price is to change sign. Clearly that would have happened in the first place, if the s_3 restriction had formally been treated as an equation. And this would be the case for $\lambda = 2$ in either case.

13.5 Treating the parameter as a variable

To cope efficiently with upper bounds, we introduce a method of making parametric steps which is somewhat different from the one outlined in the two previous sections.

There is no conceptual or definitional problem concerning upper limits. Upper limits on specified variables are restrictions, and they may become binding. Or parametric variation of the objective function may cause the shadow-prices of upper bounds to become negative, and the upper bounds may cease to be binding, and the corresponding variables may again be represented by normal entries in the list of basic variables. However, the practical computational implementation of parametric sensitivity analysis makes use of an adaptation of the linear programming procedure, and certain special complications arise from the desirability to use basically the same algorithm for parametric as well as for "ordinary" steps.

Recall the double search operation for the pivot row, i.e. for the elimination of a variable or alternatively the slack of its upper bound restriction, which was discussed in section 10.3.

We could duplicate this search operation in connection with parametric variation of the right-hand side. But that is not the end of the complications. The re-entry of the linear programming procedure can cope with a minus zero (e.g. -0.000001), in the value column, indicating a violated restriction, but not with negative entries in the updated upper bounds column. The point is simply that the "Phase I" part of this procedure addresses itself only to negative elements in the main value column. If someone initially supplied a negative upper limit this would in general lead to malfunctioning of the procedure as well.

An adaptation of the LINP procedure (developed in the context of integer programming), in which Phase I also deals with negative upper limit distances, is in fact offered in this book as well. However, the solution opted for here, is to introduce the parameter into the basis as a variable, and to re-define its value afterwards, as being -0.0000001, causing its elimination by the normal "Phase I" block of the basic LP algorithm.

To actually make the step in which the parameter is brought into the basis as a variable, is necessary, only if the leaving variable is an upper limit distance. The method is, however, generally valid, and is illustrated here, without reference to upper limits.

To illustrate this version of the algorithm, we tabulate (13.2.7), with $\underline{v}' = (0,0,0,-1,0)$, - for the same example as illustrated in tableau 13.2g and following-,
Note the change in the sign of the unit vector due to m ng the term $\lambda\underline{v}$ to the left-hand side.

TABLEAU 13.5 A

SET-UP TABLEAU WITH A PARAMETRIC ACTIVITY

NAME	X1	X2	X3	X4	X5	L0	(L=0) VALUE	DIST
S1	1	1	-	-	-1	-	1	X
S2	1	2	-	-	-3	-	0	X
S3	-	-	1	1	-1	-	1	X
S4	-	-	1	2	-3	1	0	X
S5	1	4	1	4	-12	-	-2	X
T	-3	-7	-3	-7	20	0	-	X
BOUND	100	100	100	100	100	X	X	X

As mentioned in section 12.3 (in the procedure-text itself), the number 100 represents a "fancyhigh" number where no intrinsic upper limit is intended, and the computational implementation normally uses 1 000 000, a million.

Compared with section 13.2, where λ was on the right-hand side, this presentation implies an implicit change of the sign of the parametric column.

In short, this example concerns the s_4 restriction being pushed parametrically inwards.

During the "normal" optimizing phase, the parametric column is exempt from search operations, but not from updating operations. The resulting initial optimum is given below in tableau 13.5b.

TABLEAU 13.5 B

INITIAL OPTIMUM, WITH UPDATED, BUT UNACTIVATED PARAMETRIC ACTIVITY.

NAME	S 1	S 2	S 3	S 4	S 5	L0	(L=0) VALUE	LAST
X 1	1	0.50	-1	1.50	-0.50	[1.50]	1	99
X 2	1	-2	2	-3	1	-3	1	99
X 3	-1	1.50	1	0.50	-0.50	0.50	1	99
X 4	2	-3	1	-2	1	-2	1	99
X 5	1	-1.50	1	-1.50	0.50	-1.50	1	99
T	1	1	1	1	1	1	0	X
BOUND	X	X	X	X	X	X	X	X

Note, that there is no general rule that the shadow-price of the parametric variable is positive.

The parametric variable becomes incoming variable, irrespective of its shadow price. The parametric step is now made as a normal step, leading to tableau 13.5c.

TABLEAU 13.5 C

THE PARAMETRIC ACTIVITY HAS ENTERED THE BASIS.

NAME	S 1	S 2	S 3	S 4	S 5	X 1	(L=0.67) VALUE	DIST
L0	0.67	0.33	-0.67	1	-0.33	0.67	0.67	X
X 2	3	-1	-	-	0	2	3	97
X 3	-1.33	1.33	1.33	-	-0.33	-0.33	0.67	99.33
X 4	3.33	-2.33	-0.33	-	0.33	1.33	2.33	97.67
X 5	2	-1	0	-	0	1	2	98
T	0.33	0.67	1.67	0	1.33	-0.67	-0.67	X
BOUND	X	X	X	X	X	100	X	X

The actual variation of the problem arises when λ is redefined to be minus zero, and we re-enter Phase I to make an ordinary step.

TABLEAU 13.5 D

TABLEAU WITH A RE-DEFINED PARAMETRIC VARIABLE.

NAME	S 1	S 2	S 3	S 4	S 5	X 1	(L=0.67) VALUE	DIST
L1	0.67	0.33	-0.67	1	-0.33	0.67	-0.00	X
X 2	3	-1	-	-	0	2	3	97
X 3	-1.33	1.33	1.33	-	-0.33	-0.33	0.67	99.33
X 4	3.33	-2.33	-0.33	-	0.33	1.33	2.33	97.67
X 5	2	-1	0	-	0	1	2	98
T	0.33	0.67	1.67	0	1.33	-0.67	-0.67	X
BOUND	X	X	X	X	X	100	X	X

The question arises, whether the procedure outlined in this section, is fully equivalent with the one outlined in section 13.2. This is indeed the case. Both versions of the parametric variation algorithm consist of pairs of steps, parametric variation steps and ordinary steps, and despite the different presentation of the parametric variation step, we develop after each pair of steps, substantially the same tableau by either method.

When we actually make the step marked in tableau 13.5d, we develop substantially the same tableau as in section 13.2, i.e. tableau 13.2 i, the ordering and the classification of λ as a variable being the only difference. The selection of the same incoming variable in the ordinary step (s_3 in the example), by both versions of the algorithm, is systematic, as may be shown by expressing the dual ratio of a tableau as developed according to the rules of this section in the coefficients as they would arise in the version of the algorithm discussed in section 13.2.

$$-\frac{t^*_{m+1,j}}{t^*_{i,j}} = -\left(\frac{t_{m+1,j} - t_{m+1,n} \cdot t_{i,j}/t_{i,n}}{t_{i,j}/t_{i,n}}\right)$$

$$= -\left(t_{m+1,j} \cdot t_{i,n} - t_{m+1,n}\right) \qquad (13.5.1)$$

$(j = 1,\ldots n-1)$

In (13.5.1), $t_{m+1,j}$ is the objective function row/incoming variable column cell figuring in the search operation for making the ordinary step, made according to the rules of section 13.2 ($j=3$, $m=5$, $t_{6,3}=1$ for the τ/s_3 cell in tableaux 13.2g and 13.2h), $t_{i,j}$ is the corresponding pivotal row element (-1 for x_1/s_3 cell, $i=1$), the corresponding starred expressions are the updated cells, following the parametric step made according to the rules of this section (i.e. $t^*_{6,3} = 1.67$ and $t^*_{1,3} = -0.67$ in tableau 13.5d).

The index n refers to the parametric variable, i.e. assumes the ordering used in this section.

The second expression in (13.5.1) gives the division of the two updating expressions, the third expression is a simplification of the second one.

Since $t_{i,n}$ is the pivot in the parametric step we have $t_{i,n} > 0$. It follows that the expressions

$$-\frac{t_{m+1,j}}{t_{i,j}}, \quad -\frac{t_{m+1,j} * t_{i,n}}{t_{i,j}} \text{ and } -\frac{t^*_{m+1,j}}{t^*_{i,j}}$$

all give the same ranking, at least as far as they are positive.

Eligible pivots are in fact found, only in those columns for which all these expressions are positive in both tableau-presentations. There is no change of sign between $t_{i,j}$ and $t^*_{i,j}$, but the transition from $t_{m+1,j}$ to $t^*_{m+1,j}$ needs further discussion.

It is true that the parametric step, performed according to the rules of this section may lead to the emergence of negative entries in the objective function row, thereby causing the expression given by (13.5.1) to change sign. However, such a change in the sign of an element of the objective function row is of necessity associated with a positive element in the pivotal row, i.e. related to a variable which is not eligible as incoming variable (in a tableau which represents a non-feasible solution), in either version of the algorithm.

The dual ratio is therefore the effective column-selection rule in both versions of the algorithm, and (13.5.1) indicates that both dual ratio search operations will come up with the same incoming variable. The steps are effectively the same as in section 13.2, and it follows that any negative entries in the objective function row which may be developed by the parametric step, when the rules of this section are followed, will again disappear when the ordinary step is made. The stated equivalence property includes the case j=n, to which (13.5.1) is not applicable e.g. x_1 in the example. The $t^*_{i,n}$ element is the reciprocal of the parametric pivot hence positive and not eligible as pivot in a Phase I tableau. Since the same vertex is developed by both methods (e.g. tableau 13.2i) and the use of the dual ratio in section 13.2 guarantees finding a new optimum (unless the problem has become empty), this will also be so when the rules of this section are applied.

Note that no amendment of the basic Phase I search operations as laid down in sections 11.4 and 12.3 is needed. These rules already ensure that the dual ratio is the effective column-selection criterion in the situation discussed in section 12.3 and, for the reasons given above also if the parametric variable is re-defined after entering the basis.

While the above observations merely prove the equivalence of the two methods, the rationale of the method outlined in this section is that parametric steps can be made as "ordinary" steps even if the leaving variable is an upper limit distance and that essentially the same programme code can be used to perform the search operation.

In the case of variation of the objective function, we have a parametric restriction, which is adjusted to require that the parametric component of the objective function is slightly more

than it is in the current vertex. Phase I of the linear programming algorithm is then re-entered, with recognition of a small -0 entry as a violated restriction. This -0 entry is offered to the search-loop of Phase I, irrespective of the true value of the parametric component of the objective function, which is restored just before a step is actually made.

Example

Consider the same numerical example as discussed in section 12.3. We write the initial tableau as in tableau 13.5e below

TABLEAU 13.5 E

SET-UP TABLEAU WITH THE PARAMETRIC ROW SLOTTED AS THE LAST RESTRICTION, INDEX M.

NAME	!	X 1	X 2	X 3	X 4	X 5	!!	VALUE
S 1	!	1	1	-	-	-1	!!	1
S 2	!	1	2	-	-	-3	!!	0
S 3	!	-	-	1	1	-1	!!	1
S 4	!	-	-	-1	-1	1	!!	-1
S 5	!	-	-	1	2	-3	!!	0
S 6	!	1	4	1	4	-12	!!	-2
T*	!	-1	-	-	-	-	!!	0
T	!	-3	-7	-3	-7	20	!!	-

The corresponding initial optimum tableau is now presented as in tableau 13.5f below

TABLEAU 13.5 F

INITIAL OPTIMUM TABLEAU WITH THE PARAMETRIC ROW SLOTTED AS THE LAST RESTRICTION, INDEX M.

NAME	!	S 1	S 2	S 3	S 5	S 6	!!	VALUE
X 1	!	1	0.50	-1	1.50	-0.50	!!	1
X 2	!	1	-2	2	-3	1	!!	1
X 3	!	-1	1.50	1	0.50	-0.50	!!	1
X 4	!	2	-3	1	-2	1	!!	1
X 5	!	1	-1.50	1	-1.50	0.50	!!	1
S 4	!	-	-	1	-	-	!!	0.00
T*	!	1	0.50	-1	1.50	-0.50	!!	1
T	!	1	1	1	1	1	!!	0

The parametric adjustment of the objective function is now made effective by requiring the τ^* restriction to be binding. We artificially activate Phase I, i.e, we declare the τ^* restriction to be violated for the purpose of search operations, but restore the correct entry before the resulting step is actually made. The resulting parametric step is marked in tableau 13.5g below.

TAELEAU 13.5 G

THE DUAL PARAMETRIC SEARCH, ACTIVATED BY
A JUST-VIOLATED PARAMETRIC RESTRICTION.

NAME	!	S 1	S 2	S 3	S 5	S 6	!!	VALUE
X 1	!	1	0.50	-1	1.50	-0.50	!!	1
X 2	!	1	-2	2	-3	1	!!	1
X 3	!	-1	1.50	1	0.50	-0.50	!!	1
X 4	!	2	-3	1	-2	1	!!	1
X 5	!	1	-1.50	1	-1.50	0.50	!!	1
S 4	!	-	-	1	-	-	!!	0.00
T*	!	1	0.50	[-1]	1.50	-0.50	!!	-0.00
T	!	1	1	1	1	1	!!	0

The requirement that the parametric component of the objective function increases by an ε-amount (printed as 0.00 in the tableau) is either attainable in one step, and selection of the incoming variable according to the dual ratio may be left to a "normal" Phase I search operation or alternatively if the parametric component of the objective function is already at its maximum, an empty problem is indicated.

At this point it may be observed that an upper limit is not an acceptable leaving variable in a dual parametric step. Either the parametric restriction's "slack variable" is signalled as the leaving variable, or no parametric step can be made at all, the parametric component of the objective function being already at its maximum.

This consideration makes it possible to proceed, from this point onwards, very much in the same way as in section 13.3, i.e. update the τ^{**} row only, place a $-\varepsilon$ entry in the τ^{**} row/ incoming variable column cell, and make the corresponding ordinary step.

The only difference is that the τ^*-row is slotted in position m as an "ordinary" row of the tableau, permitting the use of the "normal" Phase I search operations in order to identify the incoming variable and the appropriate dual ratio.

The parametric step is, however, not made in the usual "normal" LP sense, we only update the τ**-row, to obtain tableau 13.5h.

TABLEAU 13.5 H

ORDINARY STEP, FOLLOWING THE DUAL PARAMETRIC STEP. (EQUIVALENT TO TABLEAU 13.3 C)

NAME	!	S 1	S 2	S 3	S 5	S 6	!!	VALUE
X 1	!	1	0.50	-1	1.50	-0.50	!!	1
X 2	!	1	-2	2	-3	1	!!	1
X 3	!	-1	1.50	1	0.50	-0.50	!!	1
X 4	!	2	-3	1	-2	1	!!	1
X 5	!	1	-1.50	1	-1.50	0.50	!!	1
S 4	!	-	0	(1)	0	-0	!!	0
T*	!	1	0.50	-1	1.50	-0.50	!!	1
T**	!	2	1.50	-0.00	2.50	0.50	!!	1

The method of carrying on from this point onwards will be obvious.

13.6 Computational implementation of parametric LP

Before addressing ourselves to the details of the computational implementation, it is useful to recapitulate the salient features of the parametric linear programming algorithm.

Parametric steps come in pairs, i.e. a parametric adjustment step and, after re-definition of the problem, an ordinary step, to regain an optimal and feasible solution.

For variation of the right-hand side the parametric variation step may, or may not consist of entering a parametric activity as a basic variable irrespective of its shadow-price, in which case the adjustment redefines the value of that variable as minus zero, and the ordinary step eliminates the parametric variable.

For variation of the objective function, the parametric variation step consists of finding an incoming variable which increases the parametric component of the objective function, at the lowest relative loss in terms of the value of the specified objective function.

The search operations for variation of the right-hand side and the objective function are sufficiently similar to each other and to "normal" LP search operations, to justify their integration in an adaptation of the LINP procedure.

The parametric variation step is initiated by temporarily putting a small negative entry in the tableau, this minus zero (actually -0.0000001) is replaced, just before the step is actually made, by the "true" figure.

To this end, the linear programming code from section 12.5 and the LINP procedure from section 12.3 were suitably amended.

These amendments cover the following points: firstly, two rather than one point of call to the linear programming procedure are required, one for calculating the initial optimal and feasible solution, one for the variation exercise. Also, the linear programming procedure was actually re-named i.e., LINP still is the version from section 12.3, and we now need SIMP for simplex procedure.

The simplex procedure differs from the original linear programming procedure on the following points: there are two additional procedure-parameters of type integer, called PR and PC.

These variables communicate the presence of a parametric λ-variable or a parametric restriction from the calling main programme to the simplex procedure. Inside the simplex procedure, reference to these variables occurs in particular in the loops which control search operations.

PR stands for parametric row, and this variable will be one when there is a parametric τ^* row and otherwise zero; PC stands for parametric column and this variable will be one when there is a parametric column and otherwise zero.

On normal entry of the procedure (REENTRY = 0), the parametric row/column are not included in search operations.

Thus for REENTRY = 0, PR = PC = 0, the action of the simplex procedure is substantially the same as for the LINP procedure.

For REENTRY = 0, PR = 0, PC = 1, the last m^{th} "normal" column of the tableau is updated in the usual way, but it is not considered as pivot-column. In other words, the λ-column is updated, but does not enter the list of basic variables. This is meant for the initial calculation of an optimal and feasible solution, while parametric variation of the right-hand side is to be analyzed with help of the re-entry call.

For REENTRY = 0, PR = 1, PC = 0, the last n^{th} "normal" row is the parametric restriction and is not included in the search operations.

Besides these amendments to the search operations we also need some additional control loops to initiate parametric steps in the first place, to regulate the succession of a parametric variation step by a "normal" step, and to signal the various exits of the parametric adjustment algorithm, and to decide whether the parametric step is made by full tableau-updating, or by vector-adjustment only.

Parametric steps are initiated by the calling main programme. In the case of parametric variation of the right-hand side, the indication of the parametric λ-variable as a pivot-column is set by temporarily substituting a negative number for its shadow-price, and the exclusion of λ as pivot-column variable is removed by entering the procedure with REENTRY = 2.

In the case of variation of the objective function the similar adjustment is substituting a very small negative number for the slack of the parametric restriction. This ensures that the pivotal column is selected by the dual ratio criterion.

The normal search-loops (which are largely the same as in the LINP procedure), now establish which is the correct parametric variation step. A further special control loop then restores the numerical content of the problem as it was before the parametric re-entry. This is done at the start of making the step, just below the label MAKE THE STEP.

To make (or attempt to make) both a parametric adjustment step and the ordinary step, on the basis of a single re-entry call, there is a parametric loop in the "check for status", part of the procedure. In the case of a dual parametric step the similar control loop occurs earlier in the procedure, inhibiting tableau-updating in the parametric step itself.

Both loops control the actual adjustment of the problem, and a "GOTO" instruction to Phase I, as well as, where appropriate an instruction to exit.

Only two steps are allowed on parametric reentry.

The λ-variable, once it has again been eliminated should not re-enter the list of basic variables before the parametric step has been reported to the main programme. Yet its shadow-price may be negative, i.e. it could be that the parametric variation of the right-hand side was in a direction which increases the objective function. During the normal entry call to the simplex procedure (REENTRY = 0), this variable was slotted in the last column, and could not become a basic variable because the end-index of the column-search loop is adjusted. This method of protecting the λ-variable against undesired activation does not

work during the re-entry-call itself. To ensure exit to the main programme, it is therefore protected by allowing only two steps on parametric reentry.

Some fairly obvious minor changes in the code relate to the upper limit on the λ-variable: we do not recognize an upper limit on λ.

Unbounded parametric problems may then appear in two forms.

One or more constants of amply fulfilled restrictions may be adjusted outwards, in which case no pivotal row may be found in the λ-column.

This will be signalled by the "unbounded" exit of the Simplex procedure itself.

We may however, also have what might be called "lack of meaningful boundedness" i.e. a "fancyhigh" non-meaningful upper limit or one of the elements of x becomes binding. Formally this is not unboundedness at all, and it is possible to generate further reentry calls, until an empty problem is generated.

We now first of all list the main programme, which is an adaptation of the main LP programme listed in section 12.5.

TEXT-LISTING OF THE PARAMETRIC LP PROGRAMME.

```
'BEGIN' 'INTEGER' M,N,NAV,NEQ,FR,PC,I,J,REENTRY;

'PROCEDURE' SIMP(T,M,N,NEQ,NAV,ROWLST,COLLST,FR,PC,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,PR,FC,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';

'PROCEDURE' ORDR(T,M,N,ER,RH,ROWLST,COLLST);
'ARRAY' T; 'INTEGER' M,N,ER,RH; 'INTEGER' 'ARRAY' ROWLST,COLLST;
'ALGOL';

'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;  'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';
```

```
'COMMENT'
LINEAR PROGRAMMING BY THE SIMPLEX ALGORITHM,
WITH POST-OPTIMAL PARAMETRIC VARIATION.
FOR DETAILS OF THE ALGORITHM SEE THE TEXT OF THE SIMP PROCEDURE.

PRESENTATION OF DATA:
FIRST THE NUMBER OF RESTRICTIONS AND DUMMY-RESTRICTIONS, I.E. M,
INCLUDING THE PARAMETRIC ROW, IF ONE IS THERE.
THEN THE NUMBER OF VARIABLES AND DUMMY-VARIABLES, I.E. N,
INCLUDING THE PARAMETRIC COLUMN, IF ONE IS THERE.
FOLLOWED BY THE NUMBER OF EQUATIONS, NEQ,
FOLLOWED BY, NAV, THE NUMBER OF VARIABLES
TO WHICH THE TACIT (NON-NEGATIVITY) RESTRICTION DOES   N O T
APPLY.
THEN PUNCH PR,
WHICH IS ZERO IF NO PARAMETRIC VARIATION OF THE OBJECTIVE
FUNCTION IS ASKED FOR, AND ONE IF IT IS ASKED FOR.
FOLLOWED BY PC,
WHICH IS ZERO IF NO PARAMETRIC VARIATION OF THE RIGHT-HAND
SIDE IS ASKED FOR, AND ONE IF IT IS ASKED FOR.

THEREAFTER PUNCH EACH ROW OF THE COMPOSITE MATRIX

   A      B
   W      0
   U      0

TO REPRESENT  A.X  < OR = B,
MAXIM  -W'.X
AND  X  < OR = U

THE NUMERICAL CONTENT OF THE PARAMETRIC ROW IN NON-UPDATED FORM,
IF PRESENT,
SHOULD BE SUPPLIED AS THE LAST ROW OF A, THE ASSOCIATED
LAST ELEMENT OF B BEING ZERO.
THE NUMERICAL CONTENT OF THE PARAMETRIC COLUMN, IF SUPPLIED,
SHOULD BE PUT IN THE LAST COLUMN OF A, WITH INVERTED SIGN,
THE CORRESPONDING ENTRIES IN THE TARGET-ROW AND UPPERBOUNDS
VECTOR BEING ZERO.
;

M:=READ;  N:=READ;  NEQ:=READ;  NAV:=READ;
PR:=READ;  PC:=READ;
REENTRY:=0;
```

```
'BEGIN' 'ARRAY' TA[1:M+2,1:N+2];
  'INTEGER' 'ARRAY' ROWL[1:M],COLL[1:N];
  MATI(TA,M+2,N+1,0,0);
  SIMP(TA,M,N,NEQ,NAV,ROWL,COLL,PR,PC,REENTRY);

  PRINT RESULTS:
  'IF' N < 14 'OR' M+N < 40 'THEN' TABO(TA,M,N,0,0,2,2,ROWL,COLL)
  'ELSE' 'BEGIN'
  TABO(TA,M,0,0,N,0,1,ROWL,COLL);
   TABO(TA,0,N,M,0,1,0,ROWL,COLL); 'END';

  CHECK FOR INTRINSIC BOUNDEDNESS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' COLL[J] > 10000 'AND' TA[M+2,J] > 999999 'THEN' 'BEGIN'
    SIGNAL INTRINSIC UNBOUNDEDNESS:
    NEWLINE(1);
    WRITETEXT('('FANCYHIGH%UPPER%LIMIT%BINDING')');
    NEWLINE(1);
    'GOTO' END OF PARALP; 'END';

  'IF' PR=0 'AND' PC=0 'THEN' 'GOTO' END OF PARALP;
  'IF' REENTRY # 0 'THEN' 'GOTO' END OF PARALP;

  SET REENTRY:
  REENTRY := 2;

  'IF' PC=1 'THEN' 'BEGIN'
    'COMMENT'
    NOW ACTIVATE THE RIGHTHAND SIDE VARIATION ACTIVITY;
    TA[M+2,N] := TA[M+1,N];  TA[M+1,N] := -0.0000001; 'END';

  'IF' PR=1 'THEN' 'BEGIN'
    'COMMENT'
    NOW ACTIVATE TARGET ROW VARIATION VECTOR;
    TA[M,N+2]:=TA[M,N+1]; TA[M,N+1]:=-0.000000001; 'END';

  SIMP(TA,M,N,NEQ,NAV,ROWL,COLL,PR,PC,REENTRY);

  REORDER PARAMETRIC VECTORS TOWARDS END OF TABLEAU:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' COLL[J] = N 'AND' PC=1 'THEN' COLL[J] := 20000;

  'IF' PC=1 'AND' 'NOT' COLL[N]=N 'THEN'
  ORDR(TA,M,N,2,2,ROWL,COLL);
  'IF' COLL[N]=20000 'AND' PC=1 'THEN' COLL[N] := N;
  'IF' PR=1 'AND' 'NOT' ROWL[M]=1000+M 'THEN'
  ORDR(TA,M,N,2,2,ROWL,COLL);
  'GOTO' PRINT RESULTS;

  END OF PARALP:

'END'; 'END'
```

The above programme-text differs from the one listed in section 12.5, on some additional points besides the ones already mentioned.

There is the obvious need to declare, and to read the values of the PR and PC variables. We also need some loops to signal and effectuate the end of the algorithm.

The basic structure is to call the simplex procedure again in the re-entry mode, i.e. to ask for the next parametric step, unless an empty or an unbounded problem is met.

That refers to the two abnormal exits of the linear programming (simplex) procedure, and also to intrinsically unbounded problems which are technically bounded.

Whenever a "fancyhigh" upper limit on one of the specified variables is found binding, this is equivalent to an unbounded problem. We will refer to this situation of a technically optimal solution which has hit the ceiling as "substantially unbounded".

In terms of their significance, the following possibilities arise:

a) the problem is found empty, unbounded or substantially unbounded on return from the first normal call to the simplex procedure. The significance of that result is obvious, i.e. the originally specified problem is empty or unbounded.

b) the problem is found empty, on return from a re-entry call for parametric adjustment of the right-hand side. In that case the problem has become empty, for some result of parametric variation. (This would arise after making the parametric step itself).

c) An empty problem arises, when calling the simplex procedure in the re-entry mode, for parametric variation of the objective function. The empty problem is in that case not the main linear programming problem itself, but the redefined problem with the additional restriction, on τ^* the parametric component of the objective function. When we put a negative (-0.0000001) entry in the value column entry of the τ^* row, this means that an additional increase (of at least 0.0000001) in the value of τ^* is required. When τ^* is already at a maximum value, that requirement cannot be met.

d) An unbounded or substantially unbounded problem arises when calling the simplex procedure in the re-entry mode for parametric variation of the right-hand side. In that case the parametric variable λ is unbounded. If there are variables without non-negativity restrictions in the problem, this may also be the apparent cause of unboundedness, i.e. no parametric step has been made but the unbounded loop of the procedure has been activated. This case is limited to the specified variables either not to change, or not to be restricted in the sign, in which case there is no upper limit either. (The λ-variable itself has its upper limit removed in the procedure itself.)

   Normally unboundedness will be revealed as substantial unboundedness, by the calling main programme, i.e. some specified variable hits the ceiling. This condition is signalled, only after a parametric variation step and a further normal step have technically been completed.

d) The problem may be found unbounded, after calling the simplex procedure in the re-entry mode, for parametric adjustment of the objective function. In that case the LP problem has become unbounded, i.e. for some finite value of λ, the full objective function τ^{**} becomes unbounded.

   Again, unboundedness will normally be revealed by the main programme as substantial unboundedness, rather than the alarm exit of the simplex procedure being activated.

A full listing of the simplex procedure is not given here, this would involve too much duplication of copied parts of the linear programming procedure.

Instead, we list the file of editing instructions which converts LINP into SIMP.

The editing instructions follow the text in the same order as reading, and have the following significance:

```
TC/OLD/
```

Copy text until, somewhere in the text, the characterstring "OLD" as its first non-blank text is found.

```
TS/OLD/
```

Copy entire lines, until a card-line starting with "OLD" as its first non-blank text is found.

R/OLD/NEW/

In the next card-line, replace the character-string "OLD", by the characterstring "NEW",

I? NEW?

Insert text, as quoted.

PC/OLD/

Skip and delete text, until (somewhere in the text), the characterstring "OLD" is found.

T1 - Copy one line

P1 - Skip and delete one line

T.? OLD?

In the next line, copy characters until the characterstring "OLD" is found.

P.? OLD?

In the next line, skip and delete characters, until the characterstring "OLD" is found.

```
LISTING OF THE FILE OF EDITING INSTRUCTIONS WHICH CONVERTS
LINP INTO SIMP:

TC/'PROCEDURE' LINP(/,R/LINP/SIMP/
T.?REENTRY)?,I?FR,FC,?
TC/'INTEGER' M,N,NEQ,/,T.?REENTRY?,I?FR,FC,?
TS/'COMMENT' LINEAR/,R/./,/,T1
I?  WITH POST-OPTIMAL PARAMETRIC VARIATION.

  M RESTRICTIONS AND N VARIABLES,
  INCLUDING ONE EXTRA VECTOR, THE PARAMETRIC ROW/COLUMN
  (IF SUPPLIED).

  FOR THE SIGNIFICANCE OF THE PARAMETERS,T,M,N,NEQ,NAV,
  ROWLST AND COLLST,  SEE THE TEXT OF THE LINP PROCEDURE.
  THE PROVISIONS FOR UPPER LIMITS ON ORDINARY VARIABLES
  ARE ALSO COMMON BETWEEN SIMP AND LINP, AND ARE DESCRIBED
```

```
IN THE COMMENT AT THE HEAD OF LINP.

THE ABBREVATIONS PR AND PC INDICATE
'PARAMETRIC ROW' AND 'PARAMETRIC COLUMN'.
IF THE PROCEDURE IS ENTERED WITH REENTRY=0,
I.E. NORMAL OPTIMIZING,
WITH PC=1 AND PR=0 FOR ADJUSTMENT OF THE RIGHTHAND SIDE,
OR WIHT PR=1 AND PC=0,
FOR VARIATION OF THE OBJECTIVE FUNCTION,
THE LAST 'NORMAL' COLUMN OR ROW OF THE TABLEAU
(NOT COUNTING THE VALUE COLUMN, TARGET ROW ETC.),
IS UPDATED IN THE USUAL WAY,
BUT IS NOT INCLUDED IN SEARCH OPERATIONS.
GENERALLY, FOR REENTRY < -PR+1 AND ALSO < -PC+1,
PR ROWS AND PC COLUMNS ARE NOT INCLUDED IN SEARCH
OPERATIONS.

FOR REENTRY=2, THE CONTROL SWITCHES FOR PARAMETRIC
ADJUSTMENT ARE ACTIVATED, AS FOLLOWS:
FOR PARAMETRIC VARIATION OF THE RIGHT-HAND SIDE,
THE PROCEDURE SHOULD BE ENTERED WITH THE TRUE SHADOWPRICE
OF THE PARAMETRIC COLUMN SAVED IN THE M+2 ND ENTRY
IN THE (N TH) PARAMETRIC COLUMN,
THE SHADOWPRICE ITSELF IS THEN SET AT A NEGATIVE VALUE.
THE PARAMETRIC SEARCH WILL THEN BE MADE, BUT JUST BEFORE THE
STEP IS ACTUALLY MADE, THE TRUE SHADOWPRICE IS RESTORED.
AFTER EACH PRIMAL PARAMETRIC VARIATION STEP, THE STATUS-
LOOP DIRECTS THE PROGRAMME TO ELEMINATE THE PARAMETRIC
VARIABLE.
FOR PARAMETRIC VARIATION OF THE OBJECTIVE FUNCTION,
THE PROCEDURE SHOULD BE RE-ENTERED, WITH REENTRY=2,
AND THE PARAMETRIC RESTRICTION ADJUSTED AS JUST VIOLATED.
THE TRUE VALUE OF THE PARAMETRIC OBJECTIVE FUNCTION
SHOULD TEMPORARILY BE STORED IN COLUMN N+2 OF THE PARA-
METRIC ROW.
IN THAT CASE, THE PARAMETRIC ADJUSTMENT STEP IS MADE
AS A SEPERATE ROW-VECTOR OPERATION, ONLY THE SECOND
(ORDINARY) STEP IS MADE BY THE MAIN STEP-MAKING
LOOP OF THE PROCEDURE.

THE INTEGER VARIABLES PR AND PC (PARAMETRIC ROW AND
PARAMETRIC COLUMN) ARE NORMALLY SUPPLIED AS ZERO,
EXCEPT WHEN THE PROCEDURE IS CALLED IN THE CONTEXT OF
PARAMETRIC PROGRAMMING OR SENSITIVITY ANALYSIS.
PR SHOULD BE SUPPLIED AS ONE IN THE CASE OF
PARAMETRIC ADJUSTMENT OF THE OBJECTIVE FUNCTION, WHILE
PC SHOULD BE SUPPLIED AS ONE IN THE CASE OF PARAMETRIC
ADJUSTMENT OF THE RIGHT-HAND SIDE. ;
?
```

```
PC/;/,P1
TS/'COMMENT'/,T1
I?    FOR REENTRY=0 THE NORMAL LP ALGORITHM IS FOLLOWED
    FROM THE START,
    INCLUDING THE FILLING OF THE NAMELISTS,
    OTHERWISE THE PROCEDURE EXPECTS AN ALREADY UPDATED TABLEAU.

    THE FOLLOWING VALUES OF THE REENTRY-PARAMETER ARE
    ACCOMODATED IN THE RE-ENTRY MODE:

    REENTRY = 1.
    IF THIS VALUE IS SUPPLIED, THE ALGORITHM IS ENTERED
    AGAIN, WITHOUT SPECIAL FEATURES.

    REENTRY = 2, WITH PC = 1.
    PARAMETRIC VARIATION OF THE RIGHT-HAND SIDE.

    REENTRY = 2, WITH PR = 1.
    PARAMETRIC VARIATION OF THE OBJECTIVE FUNCTION.

    THE SIGNIFICANCE OF THE EXIT-VALUES OF THE REENTRY-
    PARAMETER IS THE SAME AS FOR LINP.
    ;
?
PC/;/,P1
TS/RETURN IN INVERSION:/,TS/'FOR' I:=1/
R/'UNTIL' M/'UNTIL' M-PR/
TS/ORDER:/,T1,TS/OR/,
R/,M,/,M-PR,/,R/N,/N-PC,/,R/2/2+PR/,R/2/2+PC/
TS/PHASE I:/,TS/'FOR' I:=/,R/'UNTIL' M/'UNTIL' M-PR,/
I?
  M 'STEP' 1 'UNTIL'  M-3+PR+REENTRY?
T.?'THEN'?,P.E,T1,T.?FEASIBLE?,I?'THEN' ?
TS/MAXIMIZE:/,TS/'FOR'/
R/'UNTIL' N/'UNTIL' N-PC,/
I?
  N 'STEP' 1 'UNTIL' N-3+PC+REENTRY?
TS/INITIALIZE SUBSTIT/,TS/'FOR' I:=/
R/'UNTIL' M/'UNTIL' M-PR,/,P.E,T1
I?     M 'STEP' 1 'UNTIL' M-3+PR+REENTRY 'DO'
?
TS/SEARCH FOR SMALLEST QUO WITH JTH COLUMN:/
TS/'FOR' I:=NAV+1/
R/'UNTIL' M/'UNTIL' M-PR,/,I?
    M 'STEP' 1 'UNTIL' M-3+PR+REENTRY ?
TS/'IF' ROWLST[I]/,R/1000/N/,T2
TC/ROWLST[I] < 1000/,R/1000/N/
TS/TRY UPPER BOUND:/,TS/'IF' QUO > 999/
I?    'IF' QUO > 1000000000 'THEN' 'BEGIN'
      'IF' J=N 'AND' REENTRY=2 'AND' PC=1
      'THEN' 'BEGIN'
        NEWLINE(1);
        WRITETEXT('('PARAMETRIC%REENTRY%PROBLEM%UNBOUNDED%%')');
        'GOTO' UNBOUNDED; 'END'; 'END';

    'IF' J=N 'AND' PC=1 'THEN' 'BEGIN'
      R:=TRYR; K:=COLN:=N; ROWN:=TRYN; VNBV:=QUO;
      'GOTO' MAKE THE STEP; 'END';

?
```

```
TS/MAKE THE STEP:/,T1
I?

  CONSULT PARAMETRIC REENTRY SWITCHES:

  'IF' REENTRY=2 'AND' COLN=N 'AND' PC=1 'THEN' 'BEGIN'
    T[M+1,K]:=T[M+2,K];
    'IF' 'NOT' R=0 'THEN' 'BEGIN'
      'IF' ROWLST[R]=ROWN 'THEN' 'BEGIN'
        QUO:=VNBV;
        'GOTO' ADJUST RIGHTHAND SIDE AND UPPER BOUNDS COLUMN;
        'END'; 'END'; 'END';

  'IF' REENTRY=2 'AND' ROWN=1000+M 'AND' PR=1
  'THEN' 'BEGIN'
    T[M,N+1]:=T[M,N+2];
    'FOR' J:=1 'STEP' 1 'UNTIL' K-1, K+1 'STEP' 1 'UNTIL' N+1
    'DO' T[M+1,J]:=T[M+1,J]-T[M,J]*T[M+1,K]/T[M,K];
    REENTRY:=1;
    T[M+1,K] := -0.0000001; 'GOTO' PHASE I; 'END';

?
TS/CONSIDER ONE COLUMN UPDATE:/
I?  CONSIDER RH ONLY UPDATE:
  'IF' REENTRY=2 'AND' PC=1 'AND' 'NOT' R=0 'THEN' 'BEGIN'
    'IF' ROWN=ROWLST[R] 'THEN' 'BEGIN'
      T[R,N+2]:=T[R,N+2]-T[R,N+1];
      T[R,N+1]:=-0.0000001;
      REENTRY:=1;
      'GOTO' PHASE I; 'END'; 'END';

?
TS/CHECK FOR STATUS:/,T1
I?  'IF' COLN=N 'AND' PC=1 'THEN' 'BEGIN'
    FEASIBLE := 'FALSE';
    T[R,N+1]:=-0.000001*0.000001; 'END';
  'IF' PC=1 'AND' ROWN=N 'THEN' 'BEGIN'
    REENTRY:=0; 'GOTO' END OF SIMP; 'END';
?
TC/END OF LINP;/,R/LINP/SIMP/
TS/ORDER FOR EXIT:/,TS/ORDL(/
I?  'IF' PR=1 'OR' 'NOT' COLLST[N]=N 'THEN'
?
R/;//,T1
I?  'ELSE' ORDL(T,M,N-PC,2,2+PC,ROWLST,COLLST);
?
TS/END OF LINP/,R/LINP/SIMP/
TE,E
```

Part III

SOME GENERAL MATHEMATICAL PROGRAMMING NOTIONS AND RELATED MATRIX ALGEBRA

CHAPTER XIV

TOPOLOGY OF FEASIBLE SPACE AREAS AND ITS RELATION TO DEFINITENESS

14.1 The mathematical programming problem

The theory of mathematical programming is traditionally stated in terms of inequalities. In fact, mixed nonlinear systems with some of the restrictions being linear equations do not give rise to more (extra) complications than those that were discussed in Chapter X with respect to linear programming. However, to avoid unnecessary complications at the level of theory, I shall follow the tradition, and introduce mixed systems of equations and inequalities only at a later stage in an applied context.

We state a general (possibly non-linear) mathematical programming problem as follows:

Maximise

$$\tau = \tau (\underline{x}) \qquad (14.1.1)$$

Subject to

$$f_i (\underline{x}) \geq 0 \qquad (14.1.2)$$

$$(i = 1,2 \ldots\ldots m)$$

Although formulated in a particular way, with the inequality sign in the $\geq$ direction and a zero on the right-hand side, (14.1.2) is in fact quite generally an inequality. Any inequality can be put in that form.

For example

$$x_1 + x_1 \cdot x_2 + x_2^2 \leq 5$$

can be re-written as

$$-x_1 - x_1 \cdot x_2 - x_2^2 + 5 \geq 0$$

and in that form the restriction conforms (14.1.2).

The definition of a function (and hence of a mathematical programming problem) does not require continuity, but we will normally assume that both the objective function τ and the restricting functions f_i are continuous and differentiable.

Just as in linear programming, it is conventional to add the non-negativity restriction

$$\underline{x} \geq 0 \qquad (14.1.3)$$

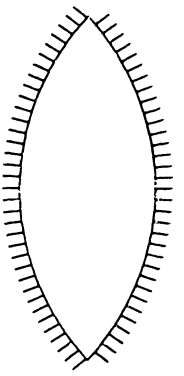
i.e. to assume that all specified variables are restricted to non-negative values.

14.2 Convex and non-convex problems

Some, but not all programming problems are specially complicated because they are non-convex.

That term refers to, for example the shape of a lens used in a pair of spectacles. If the lens is thin at the edge and thick in the middle, it is a convex lens, if it is thick at the edge and thin in the middle it is a concave lens. (See graph 14.2a)

graph 14.2 a
a convex area.



The term convex restriction includes the ordinary linear inequality i.e. a flat surface. We already gave some formal definitions in section 6.5 which are therefore not repeated here. Note that we may speak of an anti-convex restriction (= curved away from the feasible space area), but of a non-convex

area, which may have any shape provided it is not convex.

Non-convexity is a serious complication mainly because it may give rise to a number of local maxima rather than to a single optimum.

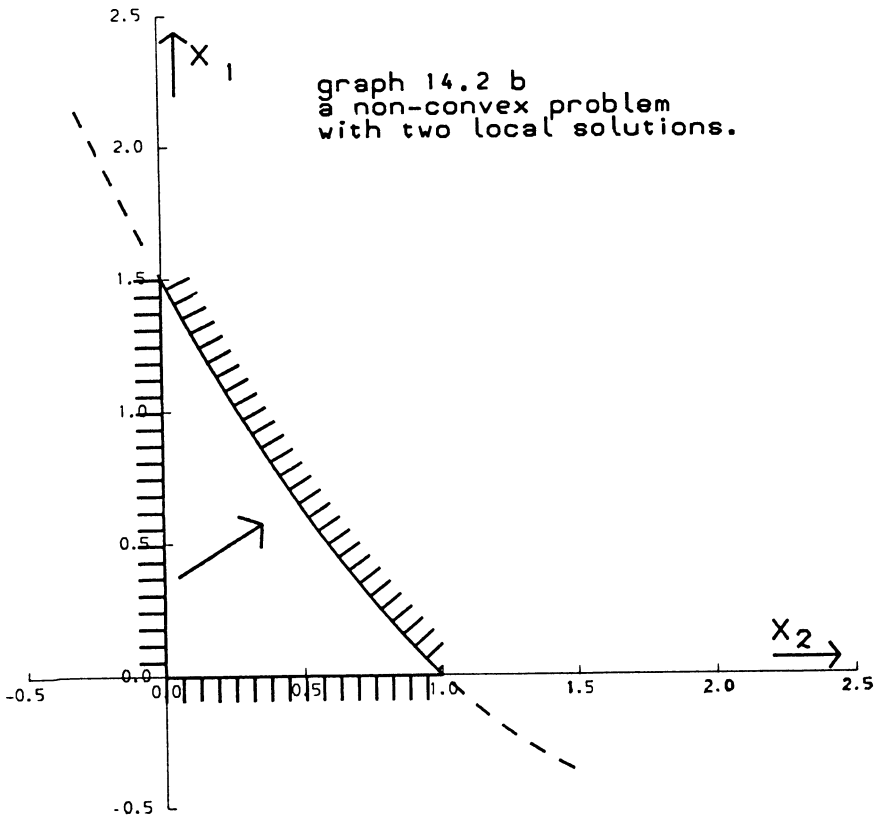
Example

Maximise $\tau = 2x_1 + 3x_2$

Subject to $2x_1 \geq (x_2 - 2)^2 - 1$

$(x_1, x_2 \geq 0)$

(See also graph 14.2b).



In this problem, there are two local maxima, one at $x_1 = 1.5$, $x_2 = 0$, and one at $x_1 = 0$, $x_2 = 1$. They are in this case of equal value, $2x_1 = 2 * 1.5 = 3$ and $3x_2 = 3 * 1 = 3$. But this can only be established by identifying both of them in the first place. And that's precisely the problem. With a large number of restrictions there could be a large number of local solutions and one might be forced to calculate the values of the variables for each one of them.

In a convex problem on the other hand, it is enough to establish the fact that every movement away from a particular point leads to a reduction of the value of the objective function.

Not surprisingly, most (but not all) operationally effective mathematical programming algorithms, refer to convex systems. The same problem i.e. a motley collection of local maxima, may also arise as a result of the specification of the objective function.

Example

Maximise $\quad - x_1 \cdot x_2$

Subject to $\quad x_1 + x_2 - 10 \geq 0$

$\quad (x_1, \quad x_2 \geq 0)$

Here the shape of the feasible space is still just convex (the surface is linear), but there are nevertheless two local maxima $x_1 = 10$, $x_2 = 0$ and $x_1 = 0$, $x_2 = 10$.

We will indicate such an objective function also as non-convex. We will now survey the problem of convexity versus non-convexity, i.e. state some definitions, theorems etc., and generally analyze the implications of various properties of restricting functions and objective functions.

A function $f(\underline{x})$ may be indicated as <u>properly convex</u>, if and only if

$$f(p \cdot \underline{x}^* + (1-p) \cdot \underline{x}^{**}) \geq p \cdot f(\underline{x}^*) + (1-p) \cdot f(\underline{x}^{**}) \quad (14.2.1)$$

is true for all $\underline{x}^*$ and $\underline{x}^{**}$, and (their non-negative linear combinations i.e.) for

$$0 \leq p \leq 1 \quad (14.2.2)$$

If the strict inequality

$$f(p \cdot \underline{x}^* + (1-p)\, \underline{x}^{**}) > p \cdot f(\underline{x}^*) + (1-p)\, f(\underline{x}^{**}) \quad (14.2.3)$$

applies for $0 < p < 1$, we might indicate the function as properly and strictly convex.

The obvious example of a properly convex function which is not strictly convex, is an ordinary linear function. However, a function which is a linear in some segments but curved or kinked at other points can also be convex, and not properly and strictly convex.

If the inequality signs in (14.2.1) and (14.2.3) are in the reverse direction i.e. $\leq$ and $<$, the function is properly anticonvex (for the $\leq$ case), or properly and strictly anticonvex (for the $<$ case).

For restricting functions and objective functions the weaker properties of peripheral and directional convexity* are relevant, and are satisfied by a wider group of functions.

These weaker properties were the reason for naming the stronger property of "proper" convexity as "proper" in the first place. Unless otherwise stated the term "convex" will in the rest of this book be understood as meaning "properly" convex.

We now give some definitions

A restriction is peripherally convex

if and only if

$$\left.\begin{array}{l} f(x^*) \geq 0 \\ f(x^{**}) \geq 0 \end{array}\right) \qquad (14.2.4)$$

(i.e. the points $\underline{x}^*$ and $\underline{x}^{**}$ both satisfying the restriction), implies that a non-negative linear combination also satisfies the same restriction

---

*The terms "peripheral" and "directional" convexity are used here, but they are not conventional terms. It was felt that these terms convey their meaning even without exact definition, and for that reason they will be used in this book. To the extent that there is a tradition, the term quasi-convex (which is the same as directionally convex) is the more conventional one. Note, that not all peripherally convex restrictions are associated with a directionally convex restricting function. Contra-examples with submerged islands cannot be excluded. For reference see Mangasarin [27] Ch.9, as well as Ponstein [30] .

If $\underline{x}^*$ and $\underline{x}^{**}$ satisfy (14.2.4), then

$$f(p\,\underline{x}^* + (1 - p)\,\underline{x}^{**}) \geq 0$$

is also true for (14.2.5)

$$0 \leq p \leq 1$$

(the set of vectors satisfying $f(\underline{x}) \geq 0$ is a convex set).

For a peripherally convex restriction, the line which marks the restriction (the periphery of the set of points which satisfy it), is flat or curved towards the inside of the feasible space area, but the function does not satisfy the definition of a (properly) convex function.

The terms peripheral and directional convexity often occur in association with a specified domain.

The term domain means also satisfying other restrictions as specified. For example, $x_1 \cdot x_2 \geq 6$ is peripherally convex in the $x_1, x_2 \geq 0$ domain. The function $y = x_1 \cdot x_2 - 6$ is not properly convex, and neither is $x_1 \cdot x_2 \geq 6$ convex for all values of x_1 and x_2. That restriction is satisfied for $x_1 = x_2 = -4$. and for $x_1 = x_2 = 4$, but not for $x_1 = x_2 = 0$. That, even within the $x_1, x_2 \geq 0$ domain, the restricting function is not properly convex, is illustrated as follows:

For $x_1 = x_2 = 1$, we find

$$f(x_1, x_2) = f(1, 1) = 1 \times 1 - 6 = -5,$$

and for $x_1 = x_2 = 3$, we find

$$f(x_1, x_2) = f(3, 3) = 3 \times 3 - 6 = 3$$

The function value of the average of the two vectors

$$f(x_1\ x_2) = f(2, 2) = 2 \times 2 - 6 = -2$$

is less than the average of the two function values which is -1. The function is therefore shown to be not properly convex. Yet the shape of the hyperbola $x_1 \cdot x_2 \geq 6$ is neatly convex. And if we perform the same calculation on the half-way linear combination of

$$f(x_1, x_2) = f(1, 6) = 1 \times 6 - 6 = 0$$

and

$$f(x_1, x_2) = f(6, 1) = 6 \times 1 - 6 = 0.$$

i.e. two points on the periphery, the result is a confirmation of the convexity property.

This half-way combination is

$$f(x_1, x_2) = f(3.5,3.5) = 3.5 \times 3.5 - 6 = 6.25$$

which is well above the average of the two function values.

The concept of peripheral convexity is relevant, mainly for functions which define restrictions.

For objective functions the stronger property of directional convexity is more relevant.

A function $f(\underline{x})$ is directionally convex (some domain) if and only if

$$\begin{aligned} f(\underline{x}^*) &\geq \lambda \\ f(\underline{x}^{**}) &\geq \lambda \end{aligned} \qquad (14.2.6)$$

implies (for all λ, $\underline{x}$ being in the specified domain, if one is specified), that for $0 \leq p \leq 1$, we find

$$f(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) \geq \lambda \qquad (14.2.7)$$

(The set of vectors satisfying $f(\underline{x}) > \lambda$ is convex).

To make the difference quite clear, we give an example of a function which is peripherally convex, but not directionally convex.

We define $f(x_1, x_2)$ as follows:

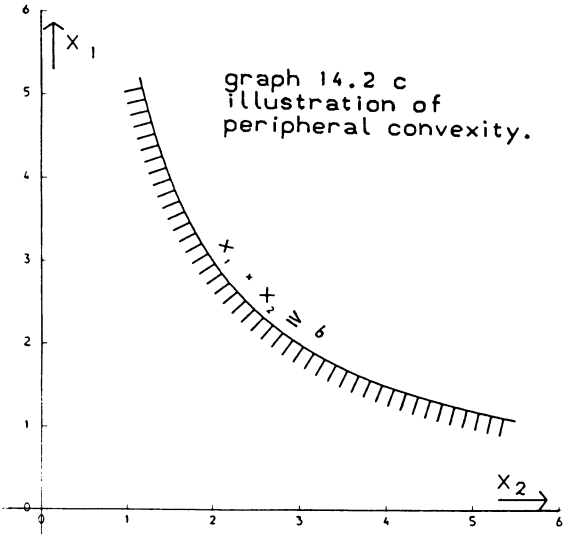
$$f(x_1, x_2) = -1, \quad \text{if } x_1 + x_2 > 5$$

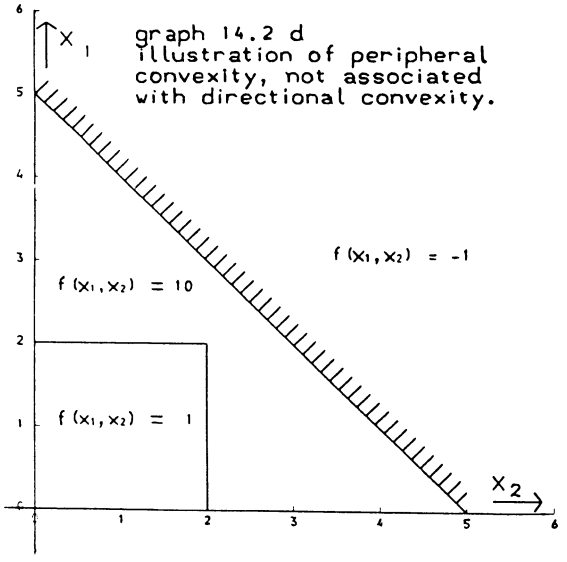
$$f(x_1, x_2) = 1, \quad \text{if } x_1 \leq 2 \text{ and } x_2 \leq 2$$

and

$$f(x_1, x_2) = 10, \text{ in all other cases.}$$

Thus, the restriction $f(x_1, x_2) \geq 0$ is convex, but $f(x_1, x_2) \geq 5$ is not convex. The square in the bottom lefthand side of graph 14.2d makes a dent in the restriction $f(x_1, x_2) \geq 5$. This


graph 14.2 c
illustration of
peripheral convexity.
X 1
X 2
x1 + x2 ≥ 6


graph 14.2 d
illustration of peripheral
convexity, not associated
with directional convexity.
X 1
X 2
f (x1, x2) = -1
f (x1, x2) = 10
f (x1, x2) = 1

non-convex "island" does not invalidate the peripheral convexity of $f(x_1, x_2) \geq 0$ as peripheral convexity is defined in terms of $f(x_1, x_2) \geq 0$. (See also graph 14.2d).

Directional convexity is thus a stronger property than peripheral convexity,

Theorem

Let $f(\underline{x})$ be a continuous and directionally convex function.

Then (even if $f(\underline{x})$ is not properly convex), there exists a properly convex function $g(\underline{x})$, which gives a similar ordering of the valuation of all possible vectors $\underline{x}$, as does $f(\underline{x})$, i.e.

$$g(\underline{x}^{**}) \geq g(\underline{x}^{*})$$

if and only if

$$f(\underline{x}^{**}) \geq f(\underline{x}^{*})$$

and the same for the $>$, $=$, $<$ and $\leq$ signs.

Substitute of a proof

Consider a series of level curves $f(\underline{x}) = \lambda$ for various values of λ, say $\lambda = -100$ to $\lambda = 100$, calibrated at suitable small intervals e.g. $\lambda = -100$, $\lambda = -99{,}99$ etc., with increments of initially 0.01 between two curves.

Take some vectors $\underline{x}^{*}$ and $\underline{x}^{**}$ and a number p in the interval $0 \leq p \leq 1$. We will indicate the higher (not lower) valued of the two vectors as $\underline{x}^{**}$, the other as $\underline{x}^{*}$, i.e. assume

$$f(\underline{x}^{**}) \geq f(\underline{x}^{*}) \qquad (14.2.8)$$

For $p = 0$ and $p = 1$ the proper convexity property (14.2.1) is trivial but satisfied irrespective of the shape of the function. We can therefore limit our investigations to the interval

$$0 < p < 1 \qquad (14.2.9)$$

For $f(\underline{x}^{**}) = f(\underline{x}^{*})$ the property of proper convexity is implied by the property of directional convexity, hence we can limit our investigation to

$$f(\underline{x}^{**}) > f(\underline{x}^{*}) \qquad (14.2.10)$$

A positive linear combination of $\underline{x}^{**}$ and $\underline{x}^{*}$ can logically satisfy one and only one of the following relations:

either

a) $f(p \cdot \underline{x}^{*} + (1 - p) \cdot \underline{x}^{**}) < f(\underline{x}^{*})$

or

b) $f(p \cdot \underline{x}^{*} + (1 - p) \cdot \underline{x}^{**}) = f(\underline{x}^{*})$

or

c) $f(\underline{x}^{*}) < f(p \cdot \underline{x}^{*} + (1 - p) \cdot \underline{x}^{**}) < f(\underline{x}^{**})$

or else

d) $f(\underline{x}^{**}) \leq f(p \cdot \underline{x}^{*} + (1 - p) \cdot \underline{x}^{**})$

Case a) contradicts the assumption of directional convexity, and must be assumed not to arise.

We may illustrate this point by drawing some level curves for a non-convex function, e.g. (see graph 14.2e)

$$f(x_1, x_2) = - x_1 \cdot x_2$$

The point

$\underline{x}^{**} = 5, 0.2$ is on the $- x_1 \cdot x_2 = - 1$ level curve, and

$\underline{x}^{*} = 1, 2$ is on the $- x_1 \cdot x_2 = - 2$ level curve. Both

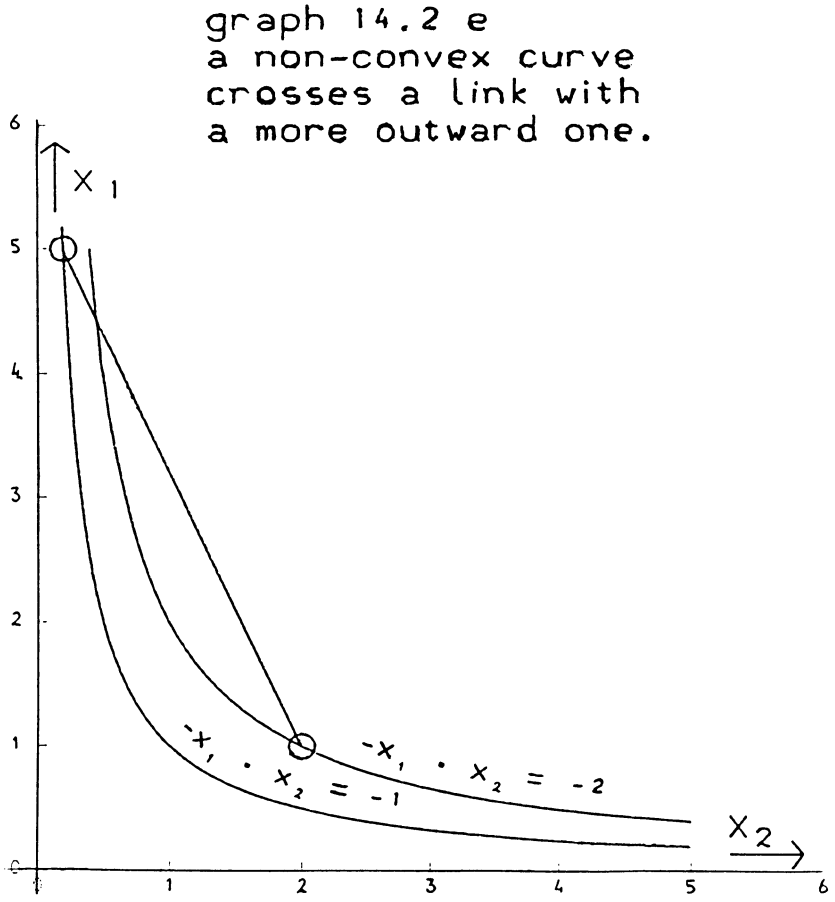
points are marked 0 in the graph.

Linear combinations of these two points are on the "wrong" side of the $- x_1 \cdot x_2 = - 2$ level curve i.e. at points where $- x_1 \cdot x_2$ is less than this proves that the set of points satisfying $- x \cdot x_2 \geq - 2$, is not a convex set.

Therefore, $f(x_1, x_2) = - x_1 \cdot x_2$ is not a directionally convex function.

End of illustration.

Exit case a)


graph 14.2 e
a non-convex curve
crosses a link with
a more outward one.
X 1
X2
-x1 • x2 = -2
-x1 • x2 = -1
0
1
2
3
4
5
6

Case b) does not directly contradict the definition of convexity, although it contradicts the definition of strict convexity. It is however a property of convex sets (strictly convex or not) that a linear combination of some points in the set, including a non-zero contribution of an interior* point, is an interior point.

By assumption the set of points satisfying $f(\underline{x}) \geq f(\underline{x}^*)$ is convex and $\underline{x} = \underline{x}^{**}$ is an interior point of that set. Therefore case b) contradicts the assumed directional convexity.

Exit case b)

Case d) would indicate that, at that particular point, $f(\underline{x})$ itself satisfies the property of convexity as defined in (14.2.1). It does not require further examination.

Exit case d).

We now concentrate on case c) where the straight line between $\underline{x}^*$ and $\underline{x}^{**}$ does not cross either of the two level-curves.

We denote the indicator of the relative valuation of the two vectors as q. i.e. q is the number which fits

$$f(p \cdot \underline{x}^* + (1 - p)\, \underline{x}^{**}) = q \cdot f(\underline{x}^*) + (1 - q) \cdot f(\underline{x}^{**}) \tag{14.2.11}$$

Should

$$q \geq p \tag{14.2.12}$$

be true, there is no problem as proper convexity is not contradicted, at this point for $f(\underline{x})$ itself.

For $q < p$ we re-define the function, i.e. we now define a new function $g(x)$.

$$g(\underline{x}) = (1 - \tfrac{q}{p}) \cdot f(p \cdot \underline{x}^* + (1 - p)\, \underline{x}^{**}) + \tfrac{q}{p} \cdot f(x)$$

$$\text{if } f(\underline{x}) \leq f(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**})$$

and

$$g(\underline{x}) = (1 - \tfrac{1-q}{1-p}) \cdot f(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) + \tfrac{1-q}{1-p} f(x)$$

$$\text{if } f(\underline{x}) > f(p \cdot \underline{x}^* + (1 - p)\, \underline{x}^{**}) \tag{14.2.13}$$

---

*No proof of this point is offered

We may assume that this transformation is in first instance performed with the vectors $\underline{x}^*$ and $\underline{x}^{**}$ and the number p and q, for which the lowest ratio q/p occurs. The transformation leaves the shape and the ordering of all the level curves intact.

It is kinked at the level-curve through $\underline{x} = p \cdot \underline{x}^* + (1 - p)\ \underline{x}^*$. There the function values of $g(\underline{x})$ and $f(\underline{x})$ are the same, but the differences from this value are scaled.

The scale-factor is $\frac{q}{p}$ for values below $f(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**})$ and $\frac{1-q}{1-p}$ for higher values of the function.

The corresponding "backward" transformation is

$$f(\underline{x}) = \left(1 - \frac{p}{q}\right) \cdot g(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) + \frac{p}{q} \cdot g(\underline{x})$$

if

$$g(\underline{x}) \leq g(p \cdot \underline{x}^* + (1 - p)\ \underline{x}^{**})$$

and

$$f(\underline{x}) = \left(1 - \frac{1-p}{1-q}\right) \cdot g(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) + \frac{1-p}{1-q} \cdot g(\underline{x})$$

if

$$g(\underline{x}) > g(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) \qquad (14.2.14)$$

Substitution of the equivalent value in $g(\underline{x})$ for $f(\underline{x})$, $f(\underline{x}^*)$ and $f(\underline{x}^{**})$, according to the backward transformation is now used to express the linear combination in $g(\underline{x})$. Application of (14.2.14) to all three terms of (14.2.11) yields:

$$g(p \cdot \underline{x}^* + (1 - p)\ \underline{x}^{**}) =$$

$$q\left(1 - \frac{p}{q}\right) \cdot g(p \cdot \underline{x}^* + (1 - p) \cdot \underline{x}^{**}) + p \cdot g(\underline{x}^*)$$

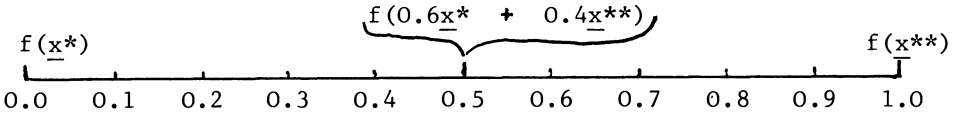
$$+ (1 - q)\left(1 - \frac{1-p}{1-q}\right) \cdot g(p \cdot \underline{x}^* + (1 - p)\ \underline{x}^{**}) + (1 - p)$$

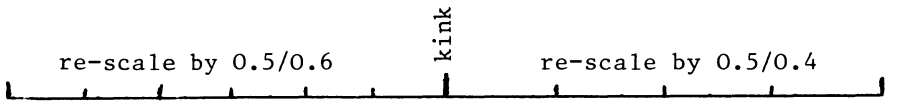
$$\cdot g(\underline{x}^{**}) = p \cdot g(\underline{x}^*) + (1 - p) \cdot g(\underline{x}^{**}) \qquad (14.2.15)$$

The transformation "blows up" differences in function value below the kink, and reduces differences above the kink.

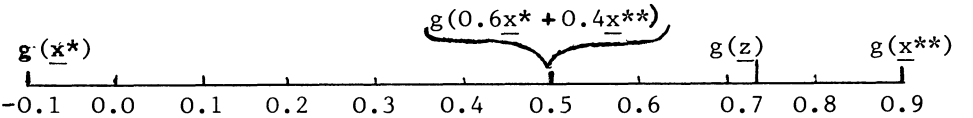
Example

Suppose it is desired to re-name a mid-point to being 60% from the lowest point and 40% from the highest point, we simply adjust the marker points. Starting with $f(\underline{x}^*) = 0.0$, $f(\underline{x}^{**}) = 1.0$, $f(0.6\ \underline{x}^* + 0.4\ x^{**}) = 0.5$,





new scale:



To comply with the mid-point as point of common reference (at the kink), i.e.

$$f(0.6\ \underline{x}^* + 0.4\ \underline{x}^{**}) = g(0.6\ \underline{x}^* + 0.4\ \underline{x}^{**}) = 0.5,$$

it is necessary to change all other marker points.

Thus $f(\underline{x}^*)$ is 0.0, but $g(\underline{x}^*)$ becomes -0.1.

Relative differences for other points are affected in the same systematic way. For example, the evaluation of the point $\underline{x} = \underline{z}$ is assumed to be $f(\underline{x}) = 0.78$. With $p = 0.6$, $q = 0.5$ and $f(0.6\ \underline{x}^* + 0.4\ \underline{x}^{**}) = 0.5$, $g(z)$ becomes (apply (14.2.13), second part)

$$g(\underline{z}) = \left(1 - \frac{0.5}{0.6}\right) \cdot 0.22 + \frac{0.5}{0.6} \cdot 0.78 = 0.73$$

End of Example.

Generally, the transformation does not create new violations of (14.2.1), but on the contrary makes that condition easier to satisfy for other linear combinations than just the one to which it was written. Should any violations be left, the procedure can be repeated.

End of proof-substitute

We now refer back to our illustration of peripheral convexity earlier in the chapter. Clearly we can replace a peripherally convex function by a directionally convex function with the same level curve $f(\underline{x}) = 0$. One simply erases any non-convex islands which may exist at other levels than the critical value of zero.

Therefore our theorem has the following

Corollary

If $f(\underline{x}) \geq 0$ is peripherally convex, then, even if $f(\underline{x})$ is not properly convex (not even directionally convex), there exists an equivalent restriction $g(\underline{x}) \geq 0$, where $g(\underline{x})$ is properly convex. The generalisation of these theorems to proper and strict convexity will be obvious.

Ordinary algebraic restrictions don't normally satisfy the definition of peripheral convexity for all values of $\underline{x}$, but there are restrictions which are peripherally convex in a specified domain, and an often-specified domain is the $\underline{x} \geq 0$ domain. In practice, with continuous and differentiable functions expressed in neat algebraic formulae, it is often possible, (but not so often advantageous) to perform the transformation needed to turn directional or peripheral convexity into proper convexity, by an algebraic reformulation. Thus $f(x,y) = x\,y$ is directionally convex in the $x, y \geq 0$ domain, while $\log f = \log x + \log y$ is properly convex and gives the same ordering. A further consequence of the theorems in this section is the following:

A mathematical programming problem in which the objective function is directionally convex and the restrictions peripherally convex, can be reformulated, and in the reformulated problem, which has the same solution, the objective function as well as the restricting functions will be properly convex. Whether a restricting function is properly convex, or only peripherally convex, is a matter of putting the restriction in a particular form. It makes no difference to the set of vectors satisfied by the restriction. Similarly, a directionally convex objective function gives the same ranking of all solution vectors, as a corresponding properly convex one. If a mathematical programming problem is characterized by a properly convex objective function and properly convex restricting functions, we call such a problem a properly convex problem. If the objective function is directionally convex and/or - one or more of the restricting functions peripherally convex (the rest being properly convex), we speak of a quasi-convex problem.

Exercise 14.2 (convexity)

The following restrictions on x_1 and x_2 are listed:

$$x_1^2 + x_2^2 \leq 16; \quad x_1^2 + x_2^2 \geq 16 \qquad (1;\ 2)$$

$$x_1 + x_2 \leq 10; \quad x_1 + x_2 \geq 10 \qquad (3;\ 4)$$

$$(x_1 + 1)(x_2 + 1) \geq 10; \quad (x_1 + 1)(x_2 + 1) \leq -10 \qquad (5;\ 6)$$

$$(x_1 + 1)(x_2 + 1) \leq 10; \quad (x_1 + 1)(x_2 - 1) \leq -10 \qquad (7;\ 9)$$

Each of these restrictions is considered in isolation from the others, i.e. each defines its own set of pairs x_1, x_2 which satisfy the restriction in question.

Which of these sets is

a) properly convex by its restricting function?

b) quasi (peripherally) convex within the x_1, $x_2 \geq 0$ domain, without being properly convex?

Which of the corresponding restricting function is

c) anti-convex

Write each of the restrictions in the conventional form of $(x_1, x_2) \geq 0$.

Hint: Make use of graphical analysis.

$(x_1, x_2) = p(x_1^*, x_2^*) + (1-p)(x_1^{**}, x_2^{**})$ is the line which joins (x_1^*, x_2^*) and (x_1^{**}, x_2^{**}).

14.3 Tangential subspaces

Consider a functional relationship

$$f(\underline{x}) = 0 \qquad (14.3.1)$$

Assuming that $f(\underline{x})$ is continuous and differentiable, we may determine for any $\underline{x} = \underline{x}^*$ the vector of first-order differentials

$$\underline{v} = \frac{\partial f}{\partial \underline{x}}(\underline{x}^*) \qquad (14.3.2)$$

Answer-help 14.2 a): nrs 1,3 and 4; b): nrs 5 and 8, also 6 in triviality on account of emptyness; c): nrs 2,3 and 4 Graphs for 5-8 at end of chapter.

For the same particular vector $\underline{x} = \underline{x}^*$ we may also determine the inner product

$$c = \underline{v}'\underline{x}^* \qquad (14.3.3)$$

We will indicate the relation

$$\underline{v}'\underline{x} = c = \underline{v}'\underline{x}^* \qquad (14.3.4)$$

as a tangential approximation of $f(\underline{x}) = 0$.

The linear approximation may be different for different points on the graph of $f(\underline{x}) = 0$. We therefore say that (14.3.4) gives the tangential approximation $f(\underline{x}) = 0$ at the point $\underline{x} = \underline{x}^*$.

Example (See also graph 14.3a)

$$f(x_1, x_2) = x_1 + \tfrac{1}{4}x_2^2 - 5 = 0$$

This functional relation is satisfied by

$$x_1 = 4, \; x_2 = 2.$$

At that point, the first-order derivatives are

$$v_1 = \frac{\partial f}{\partial x_1} = 1$$

$$v_2 = \frac{\partial f}{\partial x_2} = \tfrac{1}{2}x_2 = \tfrac{1}{2} \times 2 = 1$$

From (14.3.3) we evaluate the constant as

$$c = 1 \cdot x_1 + 1 \cdot x_2 = 4 + 2 = 6$$

and the linear approximation is found to be

$$x_1 + x_2 = 6$$

There are other tangential approximations of the same function.

For example

$$f(x_1, x_2) = x_1 + \tfrac{1}{4}x_2^2 - 5 = 0$$

is also satisfied by $x_1 = 1. \quad x_2 = 4.$

At that point, the same procedure results in a different tangential approximation, as follows

$$\frac{\partial f}{\partial x_1} = 1$$

$$\frac{\partial f}{\partial x_2} = \tfrac{1}{2}x_2 = \tfrac{1}{2} \times 4 = 2$$

Apply (14.3.4), to find the constant

$$c = 1 \cdot x_1 + 2 \cdot x_2 = 1 + 4 \times 2 = 9$$

and the tangential approximation at $x_1 = 1$, $x_2 = 4$ is found to be $x_1 + 2\,x_2 = 9$.

Before formulating theorems and proofs concerning linear approximations, we introduce a related concept.

A linear inequality restriction

$$\underline{v}'\,\underline{x} \geq c \tag{14.3.5}$$

is said to be the tangential approximation (at the point $\underline{x} = \underline{x}^*$), of

$$f(\underline{x}) \geq 0 \tag{14.3.6}$$

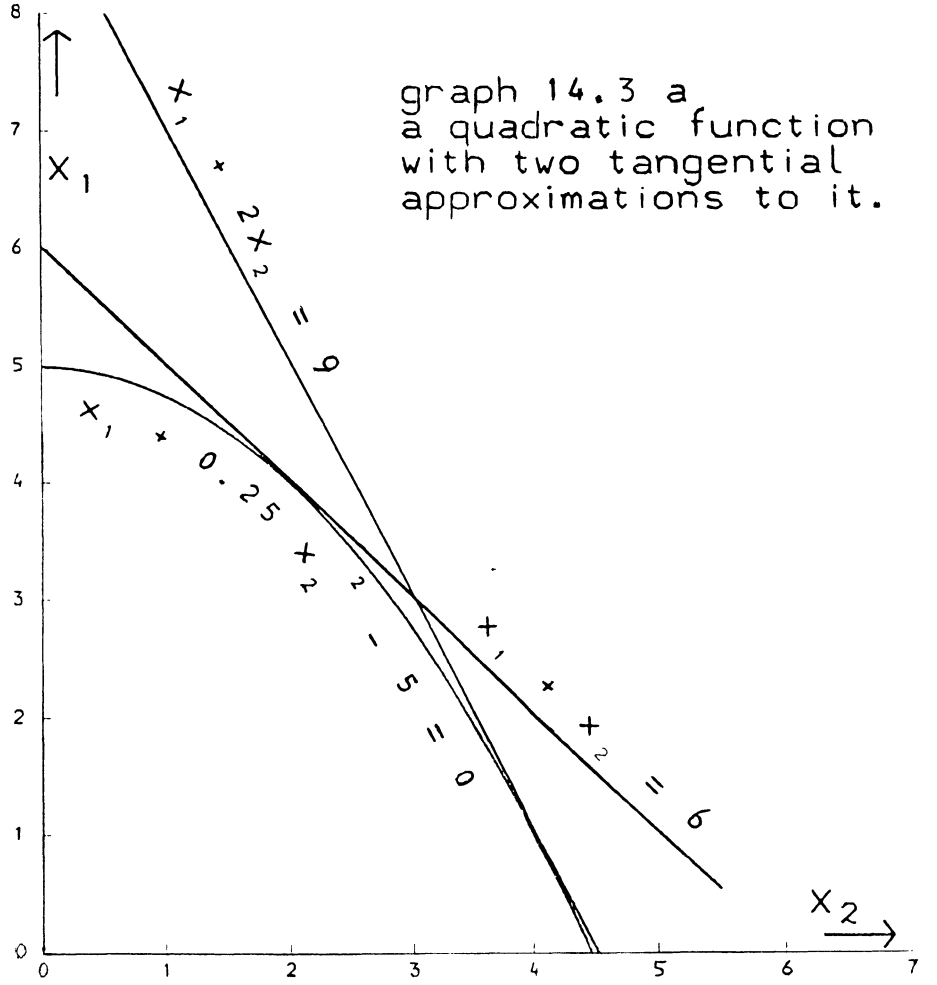
If $\underline{x} = x^*$ satisfies the binding form of (14.3.5), i.e. the tangential approximation

$$\underline{v}'\,\underline{x} = c \tag{14.3.7}$$

as well as (14.3.6) the restriction itself. If the set of points which satisfies the tangential approximation of a restriction includes the set of points satisfied by the restriction itself we also refer to it as a tangential equivalent.

In graph 14.3a, which illustrates the tangential approximation of $x_1 + \frac{1}{4}x_2^2 - 5 \leq 0$, both sets are in the left-hand bottom side of the equation-lines, and the strictly convex set $x_1 + \frac{1}{4}x_2^2 - 5 \leq 0$ just touches $x_1 + x_2 = 6$. Thus $x_1 + x_2 \leq 6$ is the tangential equivalent of $x_1 + \frac{1}{4}x_2^2 - 5 \leq 0$, at the point $x_1 = 4$, $x_2 = 2$. Just as the tangential approximations, the corresponding tangential equivalents may be different for different points of the periphery of the restriction.

At $x_1 = 1$, $x_2 = 4$, the tangential equivalent is $x_1 + 2x_2 \leq 9$.


graph 14.3 a
a quadratic function
with two tangential
approximations to it.
X1
X2
x1 + 2x2 = 9
x1 + 0.25x2^2 - 5 = 0
x1 + x2 = 6
0
1
2
3
4
5
6
7
8

Theorem

Let
the set of vectorpoints satisfying;

$$f(\underline{x}) \geq 0 \tag{14.3.6}$$

be a convex set, $f(\underline{x})$ being continuous and differentiable.

Then
every finite boundary point of that set is also an outward point.

Proof

We may, without loss of generality assume that $f(\underline{x})$ is properly convex. (See the previous section)

Now consider the function

$$z(\underline{x}) = - \underline{v}'\underline{x} + \lambda f(\underline{x}) \tag{14.3.8}$$

where $\underline{v}'$ is a finite vector and λ a finite positive number.

The necessary first-order conditions for a maximum of $z(\underline{x})$ are

$$\frac{\partial z}{\partial \underline{x}} = - \underline{v}' + \lambda \frac{\partial f}{\partial \underline{x}} = 0 \tag{14.3.9}$$

For a convex function the necessary first-order conditions are also sufficient for a maximum. This is true, even if $f(\underline{x})$ itself does not have a finite maximum.

But clearly these conditions are satisfied by any linear approximation, with $\lambda = 1$. (Just compare (14.3.2) with (14.3.9)).

This shows that, for

$$\underline{v}' = \frac{\partial f}{\partial \underline{x}} (\underline{x}^*) \tag{14.3.10}$$

$$z(\underline{x}) = - \underline{v}' \underline{x} + f(\underline{x}) \tag{14.3.11}$$

attains a maximum value at $\underline{x} = \underline{x}^*$.

i.e.

$$- \underline{v}' \underline{x} + f(\underline{x}) \leq - \underline{v}' \underline{x}^* + f(\underline{x}^*) \tag{14.3.12}$$

is true for all $\underline{x}$.

But $f(\underline{x}^*)$ is zero, because that is the definition of a boundary point. Therefore (14.3.12) reduces to

$$- \underline{v}'\underline{x} + f(\underline{x}) \leq - \underline{v}'\underline{x}^* \qquad (14.3.13)$$

From (13.3.13) we immediately infer that, because of the non-negativity of $f(\underline{x})$,

$$- \underline{v}' \underline{x} \leq - \underline{v}' \underline{x}^* \qquad (14.3.14)$$

is true for all $\underline{x}$ which satisfy

$$f(\underline{x}) \geq 0 \qquad (14.3.5)$$

This is the definition of an outward point with $- \underline{v}'$ as the direction to be maximized i.e. $\underline{x} = \underline{x}^*$ is an outward point. q.e.d.

Furthermore, comparing (14.3.14) with (14.3.6), it is clear that we have the following

Corollary (also assuming differentiability)

The tangential approximation of a convex restriction - based on a differentiable restricting function - gives the tangential equivalent, at the point of approximation. This is so at each point where the restriction is binding and a tangential approximation is taken. (A tangential approximation of a convex restriction is satisfied by each vector point which satisfies the original restriction, and is a tangential equivalent.)

Furthermore, it may be observed that if $\tau(\underline{x})$ is a convex function, $-\tau(\underline{x})$ is an anti-convex function: this follows simply from changing the signs in the definitions of (anti) convexity. From this we have the further

corollary

A tangential approximation of an anti-convex restriction is violated by each vector-point that violates the original restriction.

14.4 Extrema and convexity properties of quadratic functions

A quadratic vector function of n variables may be specified as:

$$\phi(\underline{x}) = \phi_o + \underline{a}' \underline{x} + \underline{x}' B \underline{x} \qquad (14.4.1)$$

Here, $\underline{x}$ is an n-dimensional vector, the elements of which are the arguments of the function.

The n-dimensional row-vector $\underline{a}'$ indicates the linear component of the function. For $\underline{x} = o$ the function-value becomes $\phi = \phi_o$, hence the constant ϕ_o indicates the function-value at the point of $\underline{x} = o$.

The matrix B is square (of order n by n), and is assumed to be known. An expression like $\underline{x}'\ B\ \underline{x}$ is named a quadratic form. The quadratic form $\underline{x}'\ B\ \underline{x}$ gives the quadratic component of the function, the two other terms, $\phi_o + \underline{a}'\ \underline{x}$ constitute the linear component. (See also section 2.11)

Example

$$\phi = (x_1 - 3)^2 + (x_2 - 5)^2 + (2\,x_1 - x_2)^2 =$$

$$36 - 6\,x_1 - 10\,x_2 + 5\,x_1^{\,2} + 2x_2^{\,2} - 4\,x_1\,x_2$$

leads to

$$\phi_o = 36 \text{ and } \underline{a}' = \begin{bmatrix} -6 & -10 \end{bmatrix}$$

Unless a further condition is imposed, the matrix B is not fully determined by the function.

We could write it as

$$B = \begin{bmatrix} 5 & -4 \\ 0 & 2 \end{bmatrix} \quad \text{, or as } B = \begin{bmatrix} 5 & 0 \\ -4 & 2 \end{bmatrix}$$

or as any linear combination of these two matrices.

This is because a quadratic form is its own transpose

$$\underline{x}'\ B\ \underline{x} = \underline{x}'\ B'\ \underline{x}$$

Unless the associated matrix contains other information which requires a non-symmetric presentation, a quadratic form is conventionally presented in symmetric form.

We may, in effect take the average of the two presentations offered.

This convention determines the quadratic form in the example as

$$B = \begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}$$

The existence of extrema and the convexity of quadratic functions is to a considerable extent determined by the properties of the matrix B.

In this connection the term definiteness occurs. We distinguish positive definite, positive semidefinite, indefinite, negative semidefinite, and negative definite matrices.

The definitions of these terms are

positive definite:	$\underline{x}'B\underline{x} > 0$, for all $\underline{x} \neq 0$.
positive semi-definite:	$\underline{x}'B\underline{x} \geq 0$, for all $\underline{x}$.
indefinite:	$\underline{x}'B\underline{x} < 0$, for some $\underline{x}$, but also $\underline{x}'B\underline{x} > 0$, for some other $\underline{x}$.
negative semi-definite:	$\underline{x}'B\underline{x} \leq 0$, for all x.
negative definite:	$\underline{x}'B\underline{x} < 0$, for all $\underline{x} \neq 0$.

The definiteness of a square matrix (which by convention is assumed to be a symmetric matrix) is in its turn determined by the determinantal equation (normally referred to as the characteristic equation)

$$|B - I\lambda| = 0. \qquad (14.4.2)$$

In view of the definition of singularity, the requirement (14.4.2) implies, for some non-zero vector $\underline{v}$, the equivalent definition of singularity.

$$B\underline{v} = \lambda\underline{v}. \qquad (14.4.3)$$

Equation (14.4.2) is called the characteristic equation, the vector $\underline{v}$, associated in (14.4.3) with any particular root, is the corresponding characteristic vector, the number λ itself is also known as a latent root or eigenvalue of the matrix.

For a symmetric matrix all the roots of the characteristic equation are equal.

Readers who are not familiar* with the arithmatic of complex numbers may wish to take this statement on trust, and only refer to the example below instead, but a proof is here provided for the benefit of those who are.

*For further reference (in climbing order of algebraic sophistication) see: Theil [33], Chapter 1, Hohn [21], Chapter 10, and Parlett [29].

Proof

Lemma

If $\alpha + \beta i$ is a root, $\alpha - \beta i$ is also a root, and the corresponding characteristic vectors may be required to be $\underline{v} \pm \underline{u}\, i$ where $\underline{v}$ and $\underline{u}$ are real vectors, $i = \sqrt{-1}$ (i.e. to consist also of pairs of complex numbers).

Proof (of the lemma)

Suppose

$$B(\underline{v} + \underline{u}\, i) = (\underline{v} + \underline{u}\, i)(\alpha + \beta i) = \underline{v}\alpha - \underline{u}\beta + (\underline{v}\beta + \underline{u}\alpha)\, i. \tag{14.4.4}$$

We must then assume the real and the irrational parts in (14.4.4) to balance separately, i.e.:

$B\underline{v} = \underline{v}\alpha - \underline{u}\beta$, as well as

$B\underline{u}\, i = (\underline{v}\beta + \underline{u}\alpha)\, i$ (and obviously $B\underline{u} = \underline{v}\beta + \underline{u}\alpha$).

Upon taking the difference rather than the sum of the real and the irrational parts of (14.4.4), we obtain

$$B(\underline{v} - \underline{u}i) = (\underline{v} - \underline{u}i)(\alpha - \beta i) = \underline{v}\alpha - \underline{u}\beta - (\underline{v}\beta + \underline{u}\alpha)\, i \tag{14.4.5}$$

q.e.d. (for the lemma).

(This lemma applies to all square matrices, not just to symmetric matrices).

However, for a symmetric matrix, transposition of (14.4.5) yields

$$(\underline{v}' - \underline{u}'i)\, B = (\alpha - \beta i)(\underline{v}' - \underline{u}'i). \tag{14.4.6}$$

Therefore, pre-multiplication of (14.4.4) by $\underline{v}' - \underline{u}'i$ yields:

$$(\underline{v}' - \underline{u}'i)\, B(\underline{v} + \underline{u}i) = \underline{v}'B\underline{v} + \underline{u}'B\underline{u} = (\underline{v}'\underline{v} + \underline{u}'\underline{u})(\alpha + \beta i). \tag{14.4.7}$$

In (14.4.7) the expression in the middle, $\underline{v}'B\underline{v} + \underline{u}'B\underline{u}$, is real, therefore the righthand side expression $(\underline{v}'\underline{v} + \underline{u}'\underline{u})(\alpha + \beta i)$ must also be real.

We must also exclude $\underline{v}'\underline{v} + \underline{u}'\underline{u} = 0$; $\underline{v} = 0$, $\underline{u} = 0$ would imply that $\alpha + \beta i$ was not required to be a root, (14.4.4) becoming trivial. Therefore, we must assume $\beta = 0$ for the righthand side to be real.
q.e.d. (for the theorem itself).

Note

On casual reading of this proof, it may not be immediately obvious where the symmetry enters the proof.

This is in fact the transposition from (14.4.5) to (14.4.6): the transposition symbol for B itself has been omitted, writing B instead of B'.

Example (of the characteristic equation and the real roots theorem)

$$B = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$

The characteristic equation therefore is:

$$\begin{vmatrix} 5-\lambda & -2 \\ -2 & 1-\lambda \end{vmatrix} = (5-\lambda)(1-\lambda) - 4 = \lambda^2 - 6 + 5 - 4 = \lambda^2 - 6\lambda + 1 = 0.$$

This equation solves as

$$\lambda = \frac{6 \pm \sqrt{36 - 4}}{2} = 3 \pm \sqrt{8},$$

both roots being of necessity real.

Once we solve the roots of the characteristic equation, we have no difficulty in identifying the definiteness of the matrix.

The following rules apply:

Positive definite matrices have only positive roots.
($\underline{x}'B\underline{x} > 0$ for all $\underline{x} \neq 0$ if and only if $B\underline{v} = \underline{v}\lambda$, $v \neq 0$ implies $\lambda > 0$.)

Positive semi-definite matrices have only non-negative roots.
($\underline{x}'B\underline{x} \geq 0$ for all $\underline{x}$ if and only if $B\underline{v} = \underline{v}\lambda$, $\underline{v} \neq 0$ implies $\lambda \geq 0$.)

Indefinite matrices have both positive roots and negative roots. ($\underline{x}'\,B\underline{x} > 0$ for some $\underline{x}$, and $\underline{v}'B\underline{y} < 0$ for some $\underline{y}$, if and only if

$B\underline{v} = \lambda\underline{v}$, for $\lambda > 0$, $\underline{v} \neq 0$, and also
$B\underline{u} = \gamma\underline{u}$ for $\gamma < 0$, $\underline{u} \neq 0$.)

Negative semi-definite matrices have only non-positive roots ($\underline{x}'B\underline{x} \leq 0$ for all $\underline{x}$ if and only if

$B\underline{v} = \lambda\underline{v}$, $\underline{v} \neq 0$ implies $\lambda \leq 0$).

Negative definite matrices have only negative roots ($\underline{x}'B\underline{x} \leq 0$ for all $\underline{x} \neq 0$ if and only if

$B\underline{v} = \lambda\underline{v}$, $\underline{v} \neq 0$ implies $\lambda < 0$).

(Proofs of these rules follow from evaluation of the appropriate expressions $\underline{v}'B\underline{v} = \lambda\underline{v}'\underline{v}$, and a systematic analysis of the confirmations and contradictions of the various definiteness properties which they imply).

Examples

$$\begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$ has the roots $3 + \sqrt{8} > 0$ and $3 - \sqrt{8} > 0$

and is therefore positive definite both roots being positive

(the quadratic form $\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

may also be written as $x_1^2 + (2x_1 - x_2)^2$, confirming its positive definiteness.)

$$\begin{bmatrix} 3 & -2 \\ -2 & 1 \end{bmatrix}$$

characteristic equation

$$\begin{vmatrix} 3-\lambda & -2 \\ -2 & 1-\lambda \end{vmatrix} = (3-\lambda)(1-\lambda) - 4 = \lambda^2 - 4\lambda + 3 - 4 = \lambda^2 - 4\lambda - 1 = 0.$$

$$\lambda = \frac{4 \pm \sqrt{16 + 4}}{2} = 2 \pm \sqrt{5}$$

$2 + \sqrt{5} > 0$, $2 - \sqrt{5} < 0$, the matrix is therefore indefinite, one root being positive, one root being negative.
(N.B.: A different method of recognising definiteness will be discussed in the next section).

Theorem

A quadratic function $\phi(\underline{x}) = \phi_o + \underline{a}'\underline{x} + \underline{x}'B\underline{x}$

is strictly convex if and only if B is negative definite.

Proof

The definition of strict convexity, expressed for a quadratic function by substitution of the right hand side of (14.4.1) into (14.2.3) is:

$$\phi_o + \underline{a}' (p\underline{x}^* + (1-p)\,\underline{x}^{**})$$
$$+ (p\underline{x}^* + (1-p)\,\underline{x}^{**})'B(p\underline{x}^* + (1-p)\underline{x}^{**}) >$$
$$p(\phi_o + \underline{a}'\underline{x}^* + \underline{x}^{*\prime}B\underline{x}^*) + (1-p)(\phi_o + \underline{a}'\underline{x}^{**} + \underline{x}^{**\prime}B\underline{x}^{**}) \qquad (14.4.8)$$

to be true for all $\underline{x}^* \neq \underline{x}^{**}$, p in the interval $0 < p < 1$.

(In (14.4.8) the linear terms cancel immediately, leaving the equivalent condition

$$(p\underline{x}^* + (1-p)\,\underline{x}^{**\prime}B\,(p\underline{x}^* + (1-p)\underline{x}^{**}) >$$
$$p\underline{x}^{*\prime}\,B\underline{x} + (1-p)\,\underline{x}^{**\prime}\,B\underline{x}^{**} \qquad (14.4.9)$$

(for all $\underline{x}^* \neq \underline{x}^{**}$, p in the interval $0 < p < 1$)

working out (14.4.9) we obtain

$$p^2\underline{x}^{*\prime}B\underline{x}^* + 2p(1-p)\,\underline{x}^{*\prime}B\underline{x}^{**} + (1-p)^2\,\underline{x}^{**\prime}B\underline{x}^{**} >$$
$$p\underline{x}^{*\prime}B\underline{x}^* + (1-p)\,\underline{x}^{**\prime}B\underline{x}^{**}.$$

or equivalently

$$p(p-1)\,\underline{x}^{*\prime}B\underline{x}^* - 2p(p-1)\underline{x}^{*\prime}B\underline{x}^{**} + p(p-1)\,\underline{x}^{**\prime}B\underline{x}^{**} > 0$$

or equivalently

$$p(1-p)(\underline{x}^* - \underline{x}^{**})'B\,(\underline{x}^*-\underline{x}^{**}) > 0 \qquad (14.4.10)$$

(for all $\underline{x}^* \neq \underline{x}^{**}$, p in the interval $0 < p < 1$).

In view of the condition on p, the factor p(1-p) is positive, and we end up with the definition of strict convexity being equivalent to the negative definiteness of B.
q.e.d.

Theorem

A quadratic function

$$\phi(\underline{x}) = \phi_o + \underline{a}'\underline{x} + \underline{x}'B\underline{x}$$

attains a unique maximum if and only if B is negative definite.

Proof

Assume the existence of a maximum, therefore, for some vector $\underline{x} = \underline{x}^*$ (which may or may not be the unique maximum) the necessary first order conditions

$$\frac{\partial \phi}{\partial \underline{x}} = \underline{a} + 2B\underline{x} = 0 \qquad (14.4.11)$$

are satisfied.

Substitution of $-2B\underline{x}^*$ for $\underline{a}$ in the function expression (14.4.1) permits us to express the quadratic function as

$$\begin{aligned}\phi(x) &= \phi_o - 2\underline{x}^{*\prime}B\underline{x} + \underline{x}'B\underline{x} \\ &= \phi_o - 2\underline{x}^{*\prime}B\underline{x}^* - 2\underline{x}^{*\prime}B(\underline{x} - \underline{x}^*) + \underline{x}'B\underline{x} \\ &= \phi_o - \underline{x}^{*\prime}B\underline{x}^* + (\underline{x} - \underline{x}^*)'B(\underline{x} - \underline{x}^*) \qquad (14.4.12)\end{aligned}$$

Once a first-order solution to (14.4.11) is shown to exist, (14.4.12) proves that, if the first-order solution corresponds to a unique maximum, B must be assumed to be negative definite (i.e. the quadratic form $(x - \underline{x}^*)'B(x - x^*)$ to be negative for all $\underline{x} - \underline{x}^* \neq 0$.

Conversely if B is negative definite, all its roots are negative, therefore none is zero and a unique solution to the first-order conditions (14.4.12) may be found by inversion of B.

Once the expression (14.4.12) exists, it also proves that negative definiteness of B implies the existence of a unique maximum, with $\underline{x} = \underline{x}^*$ being the point where it is obtained.
q.e.d.

The two theorems together imply that all three of the following statements are equivalent:

1) A unique maximum exists;
2) The function is strictly convex;
   and
3) B is negative definite.

The similar set of statements concerning the existence of a unique minimum of a strictly anti-convex function, i.e. the equivalence between the statements:

1) A unique minimum exists;
2) The function is strictly anti-convex;
   and
3) B is positive definite,

will be obvious.

Note, however, that a more complex situation arises in the case of semi-definiteness.

Of the two main theorems discussed in this section, the one on convexity permits immediate generalization to the semi-definite case.

A quadratic function, $\phi(\underline{x}) = \phi_o + \underline{a}'\underline{x} + \underline{x}'B\underline{x}$ is (properly) convex if and only if B is negative semi-definite.

(The proof is analageous to the strictly convex/negative definite case, and the reader is invited to go through this proof again, and to change, where appropriate, $>$ signs into $\geq$ signs, $<$ signs into $\leq$ signs, the words "strictly convex" to "convex" and "definite" to "semidefinite".)

The generalization of the other theorem on the existence of a maximum is, however, slightly different:

A quadratic function, $\phi(\underline{x}) = \phi_o + \underline{a}'\underline{x} + \underline{x}'B\underline{x}$ attains a maximum only if B is negative semi-definite.

Not only has the word "unique" disappeared and the word "semi" appeared, but the words "if and" have disappeared as well; there is no longer full equivalence between the conditions.

There are functionswhich are characterized by a semi-definite matrix B, which have no finite maximum.

The obvious example of such a function is a linear function: a zero matrix is negative semi-definite. ($\underline{x}'B\underline{x} \leq 0$ for all

$\underline{x}$ is satisfied for a zero matrix as $\underline{x}'B\underline{x} \equiv 0$). Other examples may also be given, e.g.

$$\phi = x_1 + x_2 - (x_1 - x_2)^2 = x_1 + x_2 - x_1^2 + 2x_1 x_2 - x_2^2$$

Here $B = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$ has one negative root,

$$\begin{vmatrix} -1+2 & 1 \\ 1 & -1+2 \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix}, \quad \lambda = -2,$$

and one zero root.

The existence of a finite maximum depends in such cases also on the linear component of the function.

$$\phi = x_1 - x_2 - (x_1 - x_2)^2$$

has a maximum value of $= 0.25$, which is attained whenever x_1 and x_2 satisfy the condition $x_1 - x_2 = 0.5$.

In terms of the formal algebra, we note that if B is singular (14.4.11) defines requirements which may be contradictory, or consistent, in which case they do not determine a unique maximum.

We must begin by assuming the existence of a first order solution, a point which follows in the strictly definite case from the invertability of B.

A strictly convex quadratic function always has a finite maximum; for a convex but not strictly convex function, we can only say that a solution which satisfies the necessary first order conditions is a (non-unique) maximum.

The indefinite and positive (semi) definite cases on the other hand, are clear, at least as far as (the absence of) maxima is concerned.

If B is indefinite, then, even if there is a first order solution, this is a saddle-point.

If B is indefinite, but none of its roots is zero, we can be sure of the existence of one unique first order solution, which may be found by inversion of B. The expression (14.4.12) will then exist but disprove that the solution thus found is a maximum, as $(\underline{x}^* - \underline{x})'B(\underline{x}^* - \underline{x})$ may be positive.

Example

$$\phi = x_1 + x_2 - x_1^2 + 2x_1 x_2.$$

$$a' = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

Here B is indefinite, as may be illustrated by

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -1 < 0, \text{ but also } \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 > 0$$

(The roots are $-\frac{1}{2} + \frac{1}{2}\sqrt{5} > 0$, and $-\frac{1}{2} - \frac{1}{2}\sqrt{5} < 0$.)

The first order conditions are

$$\frac{\partial \phi}{\partial x_1} = 1 - 2x_1 + 2x_2 = 0.$$

$$\frac{\partial \phi}{\partial x_2} = 1 + 2x_2 = 0.$$

These conditions permit a unique first order solution,

$$x_1 = 1, \; x_2 = -0.5.$$

This solution indicates a function value of

$$\phi = 1 - 0.5 - 1 - 1 = -1.5,$$

which is not, however, a maximum, as may be illustrated by $\phi(0,0) = 0$, and also by $\phi(1,1000) = 1 + 1000 - 1 + 2000 = 3000$. Nor is it a minimum, as may be illustrated by $\phi(1000, 0) = 1000 - 1000000 = -999000$.

If B is indefinite, but one of the roots is zero, there is either no first order solution at all, or if there is one, it corresponds to a horizontal ridge, i.e. there is a range of vectors which all satisfy the first order conditions but do not indicate a maximum or a minimum.

If B is positive semi-definite, there may, or may not be, a minimum, but there is no finite maximum. (And no saddle-point either). Any solution which satisfies the first order conditions permits the expression (14.4.12) to exist, and if B is positive semi-definite, this means that the solution in question is a minimum.

If B is also positive definite, such a minimum is also unique.

14.5 Subspaces and the partitial inversion of definite matrices

The association between a definite matrix and an extreme point of a quadratic function allows us to state and prove an algebraic theorem, the proof of which would be much more complicated, if it had to be provided by strict algebraic means

Theorem

Let

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{14.5.1}$$

be a (positive) definite matrix B_{11} and B_{22} being square diagonal blocks

Then

(1) B_{11} and B_{22} are also positive definite

and

(2) $B_{22} - B_{21} B_{11}^{-1} B_{12}$ exists and is also positive definite.

Proof

First, we partition the quadratic form.

By assumption (B being positive definite)

$$\underline{x}' B \underline{x} = \underline{x}_1' B_{11} \underline{x}_1 + \underline{x}_1' B_{12} \underline{x}_2 + \underline{x}_2' B_{21} \underline{x}_1 + \underline{x}_2' B_{22} \underline{x}_2 > 0 \tag{14.5.2}$$

is true for all $\underline{x} \neq 0$.

For $\underline{x}_2 = 0$, $(\underline{x}_1 \neq 0)$ (14.5.2) reduces to

$$\underline{x}_1' B_{11} \underline{x}_1 > 0 \tag{14.5.3}$$

for all $\underline{x}_1 \neq 0$, showing B_{11} to be positive definite. By the same argument, applied for $\underline{x}_1 = 0$, $(\underline{x}_2 \neq 0)$ B_{22} is positive definite, q.e.d. ad (1).

Concerning (2), we denote as $\underline{x} = \underline{x}^*$ the vector at which a quadratic function attains its (unconstrained) minimum.

We consider the minimum value of the quadratic function

$$\phi(\underline{x}) = \phi_o + \underline{a}' \underline{x} + \underline{x}' B \underline{x} \qquad (14.4.1)$$

where $\underline{x}$ is subject to the restriction

$$B_{11} \underline{x}_1 + B_{12} \underline{x}_2 = B_1 \underline{x} = B_1 \underline{x}^* = \underline{y} \qquad (14.5.4)$$

i.e. $B_1 \underline{x}$ is re-named as $\underline{y}$ and required to maintain the value this vector attains for $\underline{x}^*$, the point at which a quadratic function is maximal. The partitioned equivalent of (14.4.1) is

$$\phi(\underline{x}) = \phi_o + \underline{a}_1' \underline{x}_1 + \underline{a}_2' \underline{x}_2 + \underline{x}_1' B_{11} \underline{x}_1 + \underline{x}_1' B_{12} \underline{x}_2 + \underline{x}_2' B_{21} \underline{x}_1 + \underline{x}_2' B_{22} \underline{x}_2 \qquad (14.5.5)$$

Since B_{11} has been shown to be positive definite, this block is invertable. From (14.5.4) we have the following expression for $\underline{x}_1$.

$$\underline{x}_1 = B_{11}^{-1} \underline{y} - B_{11}^{-1} B_{12} \underline{x}_2 \qquad (14.5.6)$$

Substitution of the righthand-side of (14.5.6) for $\underline{x}_1$ into (14.5.5) yields after transposition and re-grouping of some terms:

$$\begin{aligned}\phi(\underline{x}) = \phi_o &+ \underline{a}_1' B_{11}^{-1} \underline{y} + \underline{y}' B_{11}^{-1} \underline{y} \\ &+ \left[\underline{a}_2' - \underline{a}_1' B_{11}^{-1} B_{12}\right]\underline{x}_2 \\ &+ \underline{x}_2' \left[B_{22} - B_{21} B_{11}^{-1} B_{12}\right]\underline{x}_2 \end{aligned} \qquad (14.5.7)$$

This is a quadratic function in $\underline{x}_2$.

Within the $B_1 \underline{x} = \underline{y}$ subspace, $\underline{x}_2$ determines the value of the function completely. Because B as a whole is positive definite, the function has a unique minimum and this minimum is in the $B_1 \underline{x} = \underline{y}$ subspace. Then by the theorem shown in the previous section such a minimum implies that the matrix $[B_{22}-B_{21} B_{11}^{-1} B_{12}]$, which arises in the quadratic forms is positive definite. q.e.d. ad (2).

Since a positive definite matrix B is associated with a negative definite matrix -B; we have the following:

Corollary:

If B is negative definite, then B_{11} and B_{22} are also negative

definite, and $B_{22}-B_{21}B_{11}^{-1}B_{12}$ exists and is negative definite.

14.6 Definite matrices and diagonal pivoting

Theorem

Let D be a square, symmetric and positive-definite matrix.

$$\underline{v}' \, D \, \underline{v} > 0 \tag{14.6.1}$$

true for all $\underline{v} \neq 0$

Let δ be a positive non-zero number

$$\delta > 0 \tag{14.6.2}$$

Then, irrespective of the content of the vector $\underline{z}$

$$M = \left[\begin{array}{c|c} D + \underline{z}\,\delta^{-1}\,\underline{z}' & \underline{z}\,\delta^{-1} \\ \hline \underline{z}'\,\delta^{-1} & \delta^{-1} \end{array}\right] \tag{14.6.3}$$

is a positive-definite matrix.

Proof

Suppose by contra assumption

$$\left[\underline{v}' \;\; \lambda\right] \left[\begin{array}{c|c} D + \underline{z}\,\delta^{-1}\,\underline{z}' & \underline{z}\,\delta^{-1} \\ \hline \underline{z}'\,\delta^{-1} & \delta^{-1} \end{array}\right] \left[\begin{array}{c} \underline{v} \\ \hline \lambda \end{array}\right] \leq 0 \tag{14.6.4}$$

to be true for some $\underline{v}$, λ, not $\underline{v} = 0$, $\lambda = 0$.

Upon working out the expression (14.6.4), we obtain

$$\underline{v}' \, D \, \underline{v} + \underline{v}' \, \underline{z} \, \delta^{-1} \, \underline{z}' \, \underline{v} + 2 \, \underline{v}' \, z \, \delta^{-1} \, \lambda + \lambda^2 \, \delta^{-1} \leq 0 \tag{14.6.5}$$

To present (14.6.5) more compactly, we may denote

$$\underline{v}' \, \underline{z} = \gamma \tag{14.6.6}$$

and (14.6.5) is written as

$$\underline{v}' \, D \, \underline{v} + \delta^{-1}(\gamma^2 + 2\,\gamma\lambda + \lambda^2) \leq 0 \tag{14.6.7}$$

or equivalently

$$\underline{v}' \, D \, \underline{v} + \delta^{-1}(\gamma + \lambda)^2 \leq 0 \qquad (14.6.8)$$

Since δ is by assumption positive non-zero, we infer from (14.6.8)

$$\underline{v} \, D \, \underline{v} \leq 0 \qquad (14.6.9)$$

Unless we assume the $\underline{v}$ is a null vector, (14.6.9) amounts to a direct contradiction of (14.6.1), the positive-definiteness of D.

This would show that the contra assumption (14.6.4) was untrue, except possibly for $\underline{v} = 0$.

However, for $\underline{v} = 0$, (14.6.5) reduces to

$$\lambda^2 \, \delta^{-1} \leq 0 \qquad (14.6.10)$$

which can only be true for $\lambda = 0$ (δ being positive non-zero).

Since $\underline{v} = 0$, $\delta = 0$ is not part of the contra-assumption (14.6.4), that contra-assumption is untrue in all cases and M must be assumed to be positive definite,
q.e.d.

The above theorem is relevant, largely because it shows a way in which a positive definite inverse may be built up.

Let

$$Q = \left[\begin{array}{c|c} D & -\underline{z} \\ \hline \underline{z}' & \delta \end{array}\right] \qquad (14.6.11)$$

be a partially inverted matrix. D being already a positive definite sub-inverse, δ ($\delta > 0$) being the next pivot. Then M is the inverse, and is shown to be positive definite.

Example

$$D = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \delta = 1, \quad \underline{z}' = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 2 & -1 \\ 1 & 1 & 1 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Both D and M positive definite.

(The roots of a non-singular matrix and its inverse are each others reciprocals, as may be seen by inverting $A\underline{u} = \underline{u}\lambda$, to obtain $\underline{u} = A^{-1}\underline{u}\lambda$, therefore $A^{-1}\underline{u} = \underline{u}\lambda^{-1}$.)

The following corollae are stated here without further proof.

For

$$Q = \begin{bmatrix} D_{11} & -Z \\ Z' & D_{22} \end{bmatrix}, \text{ with } D_{11} \text{ and } D_{22} \text{ both positive definite,}$$

$$M = \left[\begin{array}{c|c} D_{11} + ZD_{22}^{-1}Z' & ZD_{22}^{-1} \\ \hline Z'D_{22}^{-1} & D_{22}^{-1} \end{array}\right] \qquad (14.6.12)$$

is positive definite.

If D in (14.6.11) is negative definite and δ negative non-zero, M is also negative-definite.

Example

$$D = \begin{bmatrix} -2 & 1 \\ 1 & -3 \end{bmatrix} \quad \underline{z}' = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \delta = -1$$

$$Q = \begin{bmatrix} -2 & 1 & -1 \\ 1 & -3 & -1 \\ 1 & 1 & -1 \end{bmatrix} \qquad M = \begin{bmatrix} -3 & 1 & -1 \\ 1 & -4 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Similarly, if in (14.6.12) D_{11} and D_{22} are both negative definite, M is also negative definite.

These theorems allows us to test the (positive) definiteness of symmetric matrices with positive non-zero diagonal elements, as well as the positive semi-definiteness of matrices with non-negative diagonal elements. If a matrix can be inverted by positive pivots along the main diagonal, it is a positive definite matrix. As soon as a negative number appears on the main diagonal, it is found to be indefinite. If a zero appears on the main diagonal, the matrix may be positive semi-definite, or indefinite. These two cases may be distinguished as follows:

If a zero appears on the diagonal while other non-zero diagonal pivots further to the bottom-right hand side are available, re-order the quadratic form, taking the next positive non-zero diagonal pivot.

If a two by two block

$$\begin{bmatrix} - & t_{ji} \\ t_{ij} & - \end{bmatrix} \quad \text{appears, with } t_{ij} = t_{ji} \neq 0$$

this proves indefiniteness, since

$$\begin{bmatrix} x_i, & x_j \end{bmatrix} \begin{bmatrix} - & t_{ji} \\ t_{ij} & - \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} =$$

$$= 2\, t_{ij}\, x_i\, x_j = 2\, t_{ji}\, x_i\, x_j < 0$$

is true whenever x_i and x_j are both non-zero and opposite in sign, whereas the same expression is positive non-zero whenever x_i and x_j are non-zero and of the same sign.

Exercise 14.6a

Test the definiteness of:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}, \quad \begin{bmatrix} 3 & 2 & -1 \\ 2 & 4 & 2 \\ -1 & 2 & 3 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Exercise 14.6b

Test the existence of extrema of the following functions:

$$\phi_1(x_1,x_2) = x_1 + 2x_2 + 2x_1^2 + 2x_1x_2 + 4x_2^2$$

$$\phi_2(x_1,x_2) = x_1 - x_2 + 2x_1^2 - 2x_1x_2 + 4x_2^2$$

$$\phi_3(x_1,x_2) = x_1 - 4x_1^2 + 4x_1x_2 - x_2^2$$

$$\phi_4(x_1,x_2) = x_1 - 2x_2 - x_1^2 + 4x_1x_2 - 4x_2^2$$

(Formulate the function as a linear function plus a quadratic form associated with a symmetric matrix, and extract the roots from the matrix. Cross-check by inversion. Verify the first-order conditions only in the case of semi-definiteness.)

(There are answers to these exercises at the end of the chapter)

14.7 The factorization of a semidefinite matrix

Theorem

Let D be a square, symmetric and positive semi-definite matrix of order n.

$$(\underline{v}' \, D \, \underline{v} \geq 0 \text{ for all } \underline{v})$$

Then there exists a real matrix A of the same order as D, satisfying

$$A' \, A = D \tag{14.7.1}$$

where A may be required to be lower-triangular, with each diagonal element a_{ii} being non-negative.

Proof

We first deal with two trivial cases, viz:

a) n = 1

Then $A = [a_{11}] = [\sqrt{d_{11}}]$

q.e.d. for case a

b) D is a zero matrix

Then A is also a zero matrix.
q.e.d. for case b.

Furthermore $d_{ii} = 0$ (i=1,2 .... n), $d_{ij} \neq 0$ (some $i \neq j$) implies indefiniteness, see section 14.6.

In the general non-trivial case, D may now be partitioned as

$$D = \begin{bmatrix} D_{11} & \underline{d}_{12} \\ \underline{d}'_{12} & d_{22} \end{bmatrix} \tag{14.7.2}$$

where $d_{22} > 0$ applies.

(Not finding a positive non-zero diagonal element would imply that one of the special cases discussed above is applicable.)
We now assume the theorem to be true for matrices of order n-1.

We also note that positive semi-definiteness of D implies positive semi-definiteness of $D_{11} - \underline{d}_{12}\, d_{22}^{-1}\, \underline{d}'_{12}$. (See again section 14.6). Therefore, by application of the theorem for a matrix of order n-1, there exists a lower-triangular matrix B, satisfying

$$B'B = \left[D_{11} - \underline{d}_{12}\, d_{22}^{-1}\, \underline{d}'_{12}\right] \tag{14.7.3}$$

with each diagonal element of B being non-negative.

We also denote

$$\sqrt{d_{22}} = c \tag{14.7.4}$$

We now express D as

$$D = \left[\begin{array}{c|c} B' & \underline{d}_{12}\, c^{-1} \\ \hline & c \end{array}\right] \left[\begin{array}{c|c} B & \\ \hline c^{-1}\, \underline{d}'_{12} & c \end{array}\right] =$$

$$\left[\begin{array}{c|c} D_{11} - \underline{d}_{12}\, d_{22}^{-1}\, \underline{d}'_{12} + \underline{d}_{12}\, c^{-1}\, c^{-1}\, \underline{d}_{12} & \underline{d}_{12} \\ \hline \underline{d}'_{12} & c^2 \end{array}\right] =$$

$$\left[\begin{array}{cc} D_{11} & \underline{d}_{12} \\ \underline{d}'_{12} & d_{22} \end{array}\right] \tag{14.7.5}$$

Therefore, for matrices of order n, the assumption that the theorem holds for matrices of order n-1, permits us to express A as

$$A = \left[\begin{array}{c|c} B & \\ \hline c^{-1}\, \underline{d}'_{12} & c \end{array}\right] \tag{14.7.6}$$

and the proof follows by recursive induction.
q.e.d.

Example

$$D = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}, \quad D_{11} - \underline{d}_{12}\, d_{22}^{-1}\, \underline{d}'_{12} = [4] - [2]\ \underset{c=2}{0.25}\ [2] = [3]$$

$$A = \begin{bmatrix} \sqrt{3} & \\ 2/2 & 2 \end{bmatrix} = \begin{bmatrix} \sqrt{3} & \\ 1 & 2 \end{bmatrix}, \quad D = \begin{bmatrix} \sqrt{3} & 1 \\ & 2 \end{bmatrix} \begin{bmatrix} \sqrt{3} & \\ 1 & 2 \end{bmatrix}$$

The following corollae will be obvious:

If D is positive definite, $|D| \neq 0$, all the diagonal elements of A may be required to be non-zero, since they may already be required to be non-negative, we conclude to $a_{ii} > 0$.

If D is negative semidefinite D may be expressed as

$$D = -A'\,A \tag{14.7.7}$$

with the same conditions on A.

14.8 The constrained maximum of an anti-convex function, with linear side-conditions

Recall section 8.11. A linear function constrained by linear side-conditions attains its maximum (within the specified feasible space area), at a vertex. This theorem is also applicable to anti-convex functions, (of which the linear function is the borderline case).

Theorem

Let $\underline{x}$ be an n-dimentional vector.

Let $\tau = \tau(\underline{x})$ be an anti-convex function.

Let a feasible space area be defined by $x_j \geq 0$, j=1,2,..., n, and $\underline{a}_i'\underline{x} \leq b_i$ i=1,2,..., m.

Then

The maximum value of $\tau(\underline{x})$ inside the feasible space area (if finite) is attained at a point $\underline{x} = \underline{x}^*$, which is a vertex i.e. the number of non-zero elements of $\underline{x}^*$ is not greater than the number of exactly met restrictions $\underline{a}_i'\underline{x}^* = b_i$ and the non-zero sub-vector $\underline{x}_1 = \underline{x}_1^*$ may be described by the inverse of a block-pivot as $\underline{x}_1 = \underline{x}_1^* = A_{11}^{-1}\underline{b}_1$, provided the system is ordered accordingly.

Proof

If $\tau(\underline{x})$ is anti-convex, $\tau(\underline{x}) \geq \tau(\underline{x}^*)$ is an anti-convex restriction. Therefore (see section 14.3) the tangential equivalent $\underline{v}'(\underline{x}-\underline{x}^*) = \left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right]'(\underline{x}-\underline{x}^*) \geq \tau(x^*)$ is violated for all points for which $\tau(\underline{x}) < \tau(\underline{x}^*)$ is true. Therefore, if $\tau(\underline{x})$ attains (within the specified feasible space area) its maximum at $\underline{x} = \underline{x}^*$ the linear approximation of the objective function $\tau^*(\underline{x}) = \left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right]'(\underline{x}-\underline{x}^*) + \tau(\underline{x}^*)$ also attains its maximum at $\underline{x} = \underline{x}^*$. (All points at which $\tau^*(\underline{x}) > \tau^*(\underline{x}^*)$ is true also satisfy $\tau(\underline{x}) > \tau(\underline{x}^*)$; since $\tau(\underline{x}^*)$ is the maximum feasible value, such points must be infeasible points.) Therefore, $\underline{x} = \underline{x}^*$ is an optimal solution of the problem of maximizing $\tau^*(\underline{x})$, subject to the same side-conditions. Thus, the familiar L.P. property of the optimum being of a vertex is applicable.
q.e.d.

The following notes recapitulate the proof as given in part II in the linear case, now extended to the general case:

An anti-convex function does not attain an unconstrained maximum, neither in the feasible space area as a whole, nor in any linear sub-space. (See also section 6.7 where the equivalent lemma was stated and proved for the case of a linear objective function and side-restrictions of an unspecified type.) Only a strictly convex function attains an unconstrained maximum. We postulate the possibility to successively identify additional restrictions that are binding on the optimal and feasible solution. Within each linear subspace it is possible to pivot a non-basic variable against the identified restriction. This defines a residual problem, except when the last restriction has been identified and remaining non-basic variables are optimal at their non-negativity restrictions (if present). In each residual problem a new binding restriction may be identified. Therefore the total number of identifiable and invertable binding restrictions equals the number of non-zero variables in the optimal solution q.e.d.

N.B.
A theorem which is very similar to the one discussed in this section, but more general in considering non-linear side-conditions, will be discussed in section 15.5.

Answer 14.6a

Nr 2 is positive semi-definite, the others are indefinite.

Answer 14.6b

For ϕ_1 we find $B = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$; roots $3 + \sqrt{2} > 0$ and $3 - \sqrt{2} > 0$

Therefore, this is a positive definite matrix and the function has a unique minimum.

For ϕ_2, we find $B = \begin{bmatrix} 2 & -1 \\ -1 & 4 \end{bmatrix}$; which has the same roots as for ϕ_1 and the same conclusion follows.

For ϕ_3, we find $B = \begin{bmatrix} -4 & 2 \\ 2 & -1 \end{bmatrix}$; roots $\lambda_1 = -5$, $\lambda_2 = 0$

Therefore, this is a negative semi-definite matrix, and we need to verify the first-order conditions, which are

$$\frac{\partial \phi_3}{\partial x_1} = 1 \quad -8x_1 + 4x_2 = 0$$

$$\frac{\partial \phi_3}{\partial x_2} = \quad 4x_1 - 2x_2 = 0$$

Substitution of $2x_1$ for x_2 by the condition on x_2 reduces the condition on x_1 to $1 = 0$, therefore these conditions cannot be satisfied, and ϕ_3 has no finite extremum value.

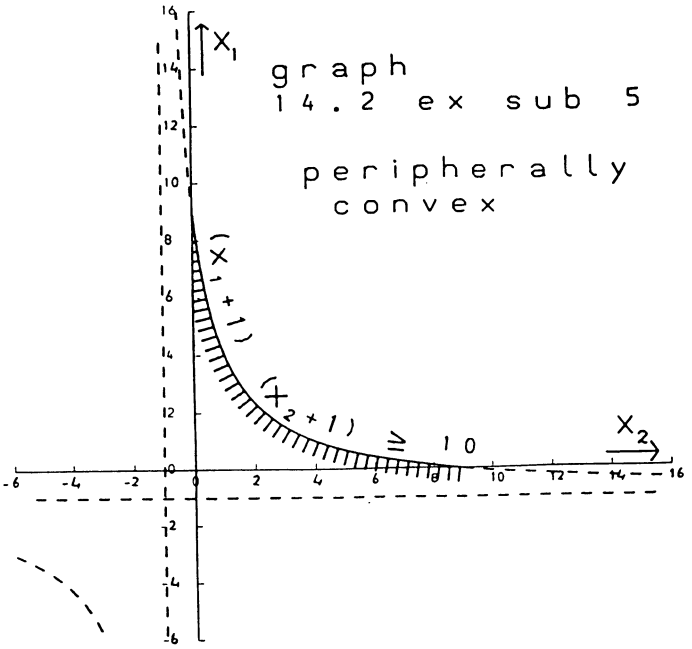
For ϕ_4 we find $B = \begin{bmatrix} -1 & 2 \\ 2 & -4 \end{bmatrix}$

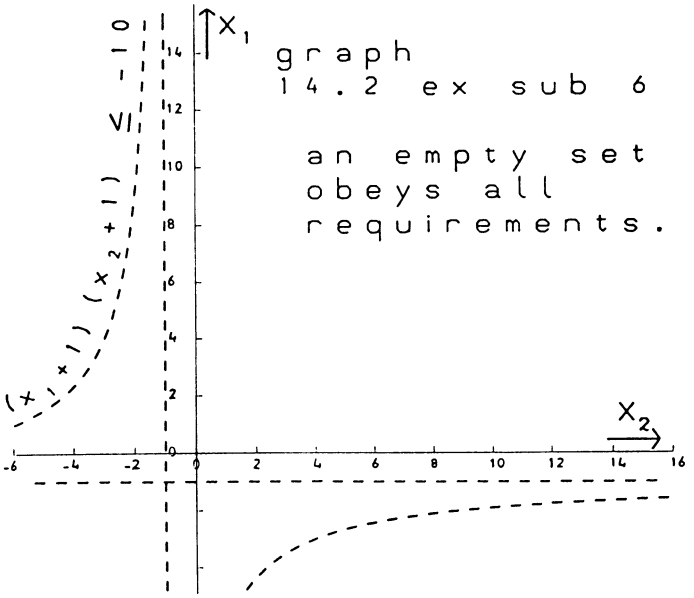
This matrix has the same roots as found above for ϕ_3; we therefore need once more to verify the first-order conditions which are:

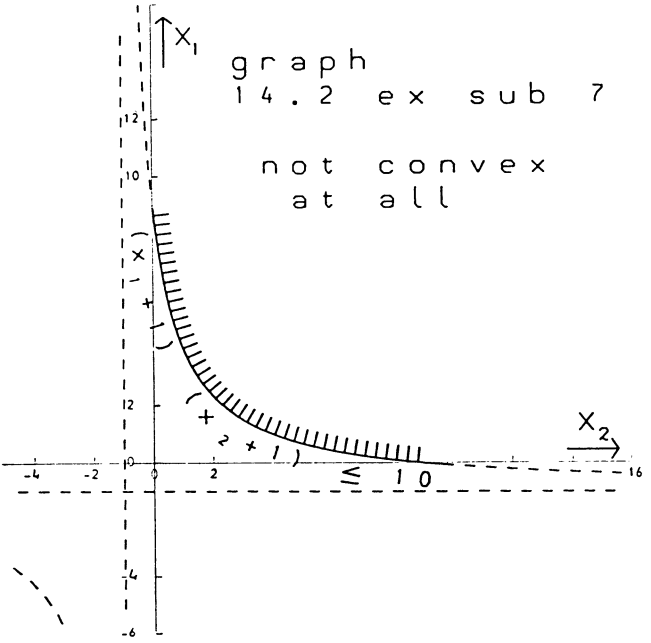
$$\frac{\partial \phi_4}{\partial x_1} = 1 \quad -2x_1 + 4x_2 = 0$$

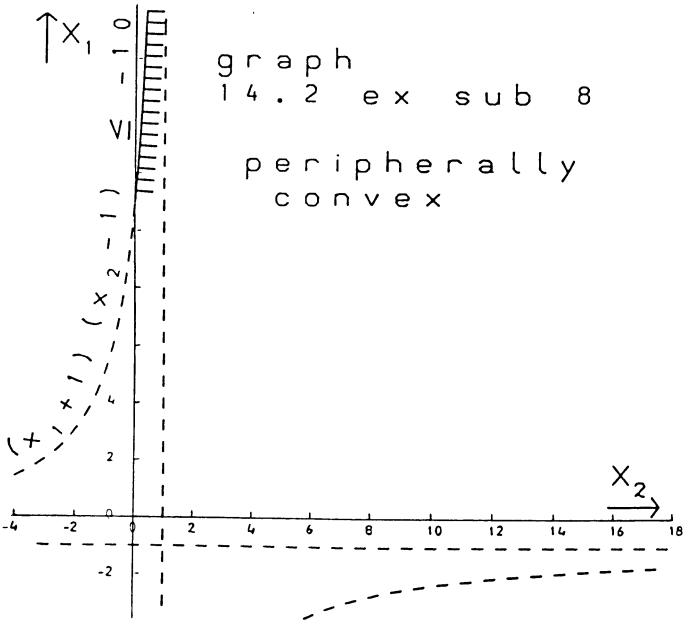
$$\frac{\partial \phi_4}{\partial x_2} = -2 \quad +4x_1 - 8x_2 = 0$$

Substitution of $2x_2 + 0.5$ for x_1 into the condition on x_2 now causes the latter condition to reduce to the trivial requirement $0 = 0$, and we find that ϕ_4 attains a maximum value, whenever x_1 and x_2 satisfy the condition $x_1 = 2x_2 + 0.5$.


graph
14.2 ex sub 5
peripherally
convex
X1
X2
(X1 + 1)
(X2 + 1) ≧ 10


graph
14.2 ex sub 6
an empty set
obeys all
requirements.
X1
X2
VI
- 10
(X1 + 1) (X2 + 1)


X1
graph
14.2 ex sub 7
not convex
at all
X2
≤ 10


X1
graph
14.2 ex sub 8
peripherally
convex
X2

CHAPTER XV

OPTIMALITY CONDITIONS

15.1 The additive properties of restrictions

Consider a set of vector-points, e.g. x_1, x_2, x_3, which are required to satisfy a combination (i.e. all) of a series of restrictions. For example x_1, x_2, x_3 may be required to satisfy

$$x_1 + x_2 + x_3^2 + 5 \geq 0$$

$$x_1 - 2x_2 = 0$$

$$-x_2 + x_3 - 10 \geq 0$$

It follows that x_1, x_2, x_3 must also satisfy

$$p_1(x_1 + x_2 + x_3^2 + 5) + p_2(x_1 - 2x_2) + p_3(-x_2 + x_3 - 10) \geq 0$$

where p_1 and p_3 are non-negative numbers, and p_2, which is associated with the equation $x_1=2x_2$, can be any real number.

For example, we may require $p_1=1$, $p_2=-1$, $p_3=3$, i.e.

$$(x_1 + x_2 + x_3^2 + 5) - (x_1 - 2x_2) + 3(-x_2 + x_3 - 10)$$
$$= x_3^2 + 3x_3 - 25 \geq 0$$

The reason is that each of the separate terms in this expression is required to be non-negative, i.e.

$$x_1 + x_2 + x_3^2 + 5 \geq 0$$

$$-x_1 + 2x_2 = 0$$

and

$$3(-x_2 + x_3 - 10) \geq 0.$$

We therefore have the following properties for various classes of restrictions:

Class a): inequalities of type ≥ 0

If the vector $\underline{x}$ satisfies

$$f_i(\underline{x}) \geq 0 \qquad (15.1.1)$$

$$i=1,2...m$$

then

$\underline{x}$ also satisfies

$$\sum_{i=1}^{m} p_i . f_i(\underline{x}) \geq 0 \qquad (15.1.2)$$

with $p_i \geq 0$ for i=1,2...m

Class b): inequalities of type ≤ 0

If the vector $\underline{x}$ satisfies

$$f_i(\underline{x}) \leq 0 \qquad (15.1.3)$$

$$i=1,2,...m$$

then

$\underline{x}$ also satisfies

$$\sum_{i=1}^{m} p_i \cdot f_i(\underline{x}) \leq 0 \qquad (15.1.4)$$

with $p_i \geq 0$ for i=1,2...m

Class c): equations

If the vector $\underline{x}$ satisfies

$$f_i(\underline{x}) = 0 \qquad (15.1.5)$$

(i=1,2...m)

then

$\underline{x}$ also satisfies

$$\sum_{i=1}^{m} p_i \cdot f_i(\underline{x}) = 0 \qquad (15.1.6)$$

where p_i can be any numbers, positive, zero or negative.

These properties are hardly theorems. They follow simply from the summation of the terms in (15.1.2), (15.1.4) and (15.1.6). If (for certain values of the arguments-vector) each term is non-negative, the sum is also non-negative. However, the

non-negativity of a sum of terms, does not require that each term is non-negative. In the example at the beginning of this section, $x_3 = 5$ satisfies the combined restriction $x_3^2 + 3x_3 - 25 \geq 0$, irrespective of the values of x_1 and x_2. One can obviously state values of x_1 and x_2 which contradict any of the individual restrictions. For $x_1 = -50$, $x_2 = 0$, $x_3 = 5$, none of the individual restrictions is met, but their combination is satisfied.

15.2 Combined restrictions and their tangental equivalent

Consider the following combination of restrictions on two variables x_1 and x_2

$$x_1 + \tfrac{1}{4}x_2^2 - 5 \leq 0$$

$$x_1 - 2x_2 + \tfrac{1}{4}x_2^2 \leq 0$$

Application of the combination-property discussed in the previous section tells us that x_1 and x_2 should also satisfy

$$p_1(x_1 + \tfrac{1}{4}x_2^2 - 5) + p_2(x_1 - 2x_2 + \tfrac{1}{4}x_2^2) \leq 0$$

$$\text{for } p_1 \geq 0,\ p_2 \geq 0.$$

Taking (for example) $p_1=3$, $p_2=5$ this combined restriction is

$$8x_1 - 10x_2 + 2x_2^2 - 15 \leq 0$$

Since both $x_1 + \tfrac{1}{4}x_2^2 - 5 \geq 0$ and $x_1 - 2x_2 + \tfrac{1}{4}x_2 \leq 0$ are convex, a combination of these two restrictions is also convex.

Theorem

Let

$f_i(x)$ be a series of properly convex functions.

$(i=1,2...m)$

Then

$$\left.\begin{aligned} g(x) &= \sum_{i=1}^{m} p_i f_i(\underline{x}) \\ p_i &\geq 0 \quad (i=1,2...m) \end{aligned}\right) \qquad (15.2.1)$$

is also properly convex.

Proof

Recall the definition of a properly convex function. To avoid a possible confusion with multipliers for functions we now use the letter w for the relative weight of a particular point. From Section 14.2 we copy and adapt (the definition of proper convexity). $f_i(\underline{x})$ is properly convex if and only if

$$f_i(w.\underline{x}^* + (1-w).\underline{x}^{**}) \leq w.f_1(\underline{x}^*) + (1-w).f_i(\underline{x}^{**}) \quad (15.2.2)$$

is true for all $\underline{x}^*$ and $\underline{x}^{**}$ and (their non-negative combinations, i.e.)

$$0 \leq w \leq 1 \quad (15.2.3)$$

The corresponding generalization to a combined restriction now follows from application of the additive property discussed in the previous section to all inequalities which state the convexity of the m restrictions by (15.2.2)

$$\sum_{i=1}^{m} p_i.f_i(w.\underline{x}^* + (1-w)\underline{x}^{**}) \leq \sum_{i=1}^{m} (w.f_i(\underline{x}^*) + (1-w).f_i(\underline{x}^{**})) \quad (15.2.4)$$

or equivalently, considering (14.2.1) the definition of the combined function:

$$g(w.\underline{x}^* + (1-w).\underline{x}^{**}) \leq w.g(\underline{x}^*) + (1-w)\ g(\underline{x}^{**}) \quad (15.2.5)$$

which is the definition of proper convexity for $g(\underline{x})$. q.e.d.

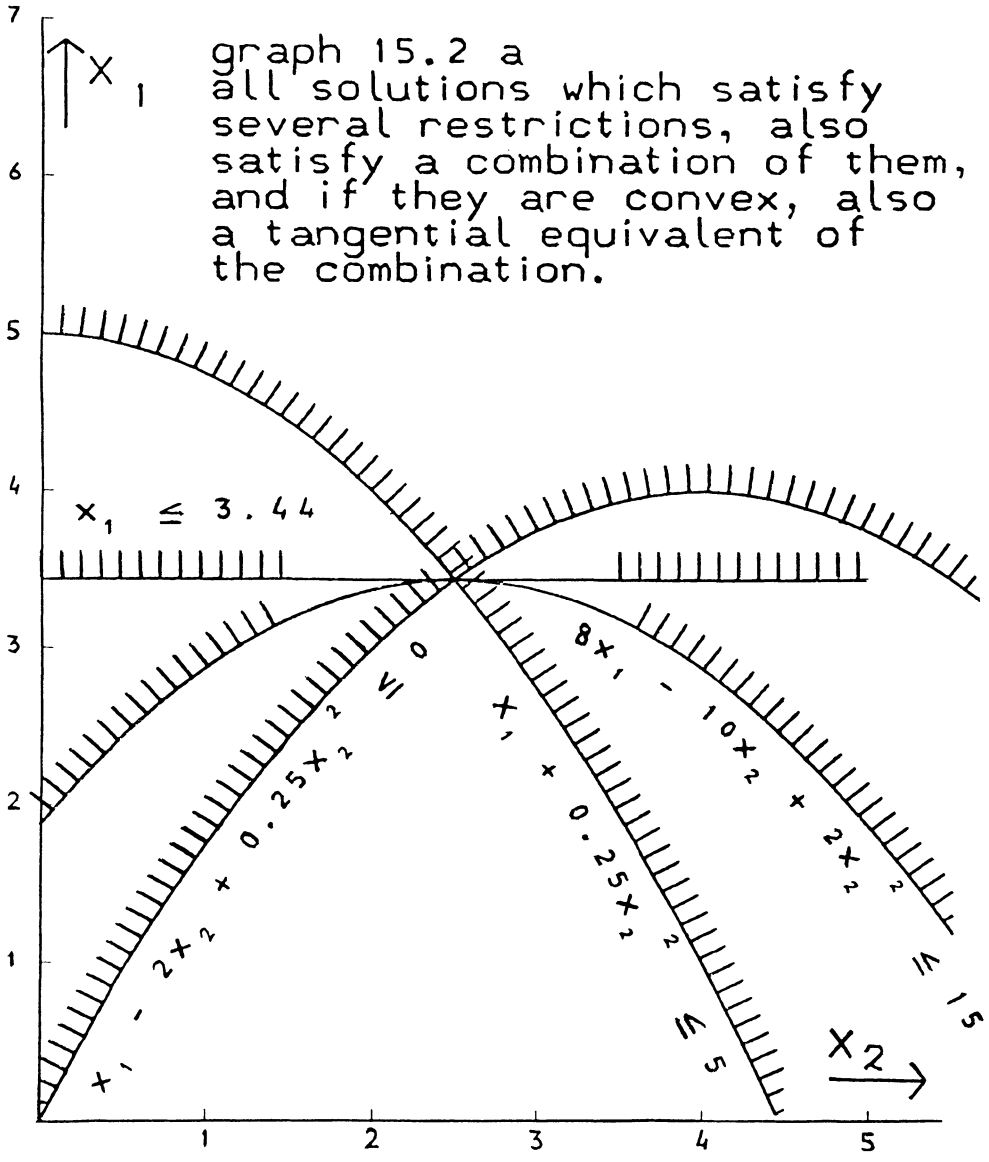
Since all vectors which satisfy a convex restriction also satisfy any tangential equivalent of that restriction, we have the following:

Corollary

If $\underline{x}$ satisfies a series of convex restrictions, that vector satisfies any tangential equivalent of any non-negative combination of them.

(Refer back to the example at the beginning of this section:

Each pair of numbers x_1 and x_2 which satisfies $x_1 + \frac{1}{4}x_2^2 - 5 \leq 0$ as well as $x_1 - 2x_2 + \frac{1}{4}x_2^2 \leq 0$, also satisfies $8x_1 - 10x_2 + 2x_2^2 - 15 \leq 0$, because that is a positive combination of the two. Therefore each pair of numbers which satisfies the two original restrictions, also satisfies $x_1 \leq 3.44$, because that is the


graph 15.2 a
all solutions which satisfy several restrictions, also satisfy a combination of them, and if they are convex, also a tangential equivalent of the combination.
X1
X2
x1 ≤ 3.44
x1 - 2x2 + 0.25x2² ≤ 0
x1 + 0.25x2² ≤ 5
8x1 - 10x2 + 2x2² ≤ 15
1
2
3
4
5
6
7

tangential equivalent of $8x_1 - 10x_2 + 2x_2^2 - 15 \leq 0$, at the point $x_1=3.44$, $x_2=2.5$.)

15.3 The Lagrangean expression and the conditions on its derivatives

A combination of the objective function with one or more of the restricting functions is indicated as a Lagrangean expression after the French mathematician La Grange or de la Grange. [25]

$$L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i \cdot f_i(\underline{x}) \qquad (15.3.1)$$

$$(p_i \geq 0,\ i=1,2...m)$$

The p_i are indicated under various names. In calculus, they are usually indicated as "undetermined multipliers". In the theory of mathematical programming they are indicated as dual variables. This is by analogy to their equivalents in linear programming. If the theory of optimality conditions (i.e. the Kuhn-Tucker theorem, see below) is applied to a linear problem, the p_i turn out to be the variables in the dual problem. Because of their economic interpretation they are also sometimes indicated as "shadowprices". I shall, by and large, follow the general mathematical programming tradition and indicate these multipliers as dual variables. The use of the letter p is, however, a reference to their economic interpretation.

The theorems by John [22] and by Kuhn and Tucker [24] concern the existence, the derivatives and the extremum properties of combinations of the objective function and restricting functions.

John's theorem is the most general. Kuhn and Tucker's refers to convex problems only, and generally assumes stronger conditions leading to stronger results.

We first form the more general (John's) expression

$$J(\underline{x}) = p_o \cdot \tau(\underline{x}) + \sum_{i=1}^{m} p_i\, f_i(\underline{x}) \qquad (15.3.2)$$

$$(p_i \geq 0,\ i=0,1,2...m)$$

where p_o may be zero - as distinct from the Lagrangean expression, where p_o is equal to one.

This is not particularly apparent in John's original publication, but Kuhn and Tucker [24] give an example where John's expression involves a zero multiplier for the objective function. Kuhn and Tucker's contra-example involves non-convexity and non-zero third-order derivatives, but contra-examples with only convex

functions and no third-order derivatives exist as well. The Kuhn-Tucker theorem is therefore associated with a constraint qualification condition.

The constraint qualification condition essentially requires the stronger result $p_0 > 0$ to hold. In practice, the contra-examples are rather special and one normally assumes the Lagrangean expression to exist.

Kuhn and Tucker's main theorem is the following

Let

Maximise $\tau(\underline{x})$

Subject to

$$f_i(\underline{x}) \geq 0$$

$$(i=1,2...m)$$

$$\underline{x} \geq 0$$

be a convex programming problem satisfying the constraint qualification condition.

And let

$$\underline{x} = \underline{x}^*$$

be an optimal and feasible solution to that problem

Then, there exists multipliers $p_i \geq 0$, such that

$$\text{a)} \quad \frac{\partial L}{\partial \underline{x}} \leq 0 \qquad (15.3.3)$$

is true,* together with the complementary slackness conditions. viz:

$$\text{b)} \quad p_i \cdot f_i(\underline{x}) = 0 \quad \text{(all i)} \qquad (15.3.4)$$

p_i is positive non-zero, only if $f_i(\underline{x}) = 0$, the restriction $f_i(\underline{x}) \geq 0$ being binding

---

*Some authors also mention a similar condition with respect to the multipliers, with the opposite sign, i.e. $\frac{\partial L}{\partial p_i} \geq 0$ $(i=1,2...m)$, but this is simply a re-statement of the original restrictions, i.e. $f_i(\underline{x}) \geq 0$.

and

$$c)\ x_j \cdot \frac{\partial L}{\partial x_j} = 0 \quad (\text{all } j) \tag{15.3.5}$$

i.e. the derivative is negative rather than zero, only if the variable is zero. The conditions (15.3.3) are known as the dual requirements or optimality conditions, (15.3.4) and (15.3.5), as the complementary slackness condition. The combination of the binding restrictions, weighed according to the corresponding dual variables,

$$\sum_{i=1}^{m} p_i\, f_i(\underline{x}) \geq 0 \tag{15.3.6}$$

may be indicated as the aggregate restriction.

Example

Maximise

$$\tau = x_1 + x_2$$

Subject to

$$x_1 + x_2^2 \leq 5$$

$$x_1 \leq 2$$

$$x_2 \leq 3$$

$$(x_1 \geq 0,\ x_2 \geq 0)$$

We first put the restrictions in the ≥ 0 form

$$-x_1 - x_2^2 + 5 \geq 0$$

$$-x_1 + 2 \geq 0$$

$$-x_2 + 3 \geq 0$$

The associated Lagrangean expression is

$$L(x_1,x_2) = x_1 + x_2 + p_1(-x_1 - x_2^2 + 5) + p_2(-x_1 + 2) + p_3(-x_2 + 3)$$

The theorem does not instruct us how to find an optimum solution.

Rather it tells us that, when an optimum has been found, certain conditions will be satisfied by the solution. For this example, we find a solution by graphical mapping and substitution.

Graphical mapping of the restrictions (see graph 15.3a), and consideration of x_1, x_2 (top-righthand) as the desirable direction identifies

$$x_1 + x_2^2 \leq 5$$

and

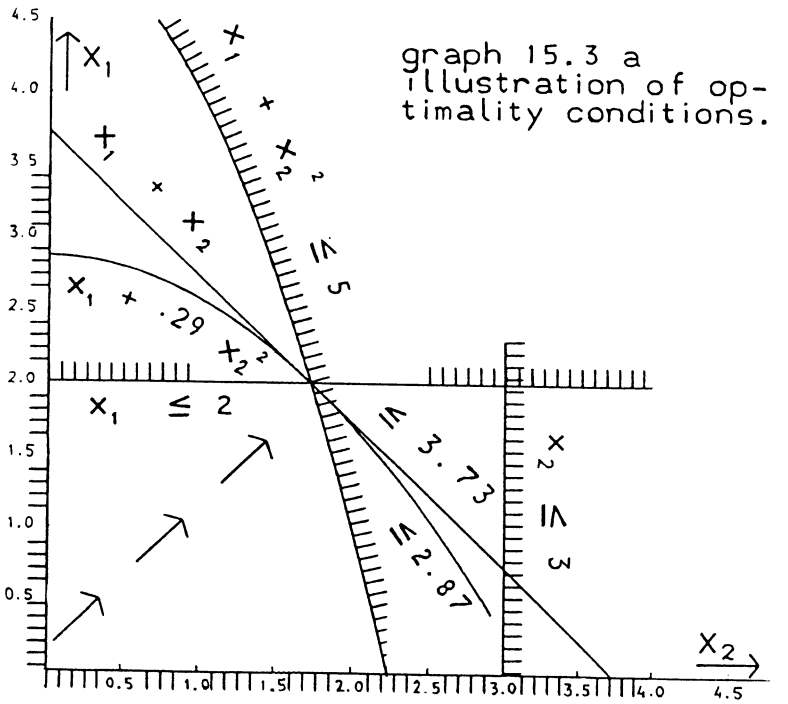
$$x_1 \leq 2$$

as the binding restrictions. Hence x_1=2. Substitution of 2 for x_1 in $x_1 + x_2^2 = 5$ then gives us x_2 as $x_2 = \sqrt{3} = 1.73$.

Note the way in which the aggregate restriction is a kind of "compromise" between the binding restriction.

Accordingly, the Kuhn-Tucker theorem tells us that the Lagrangean expression is effectively reduced to

$$L = x_1 + x_2 + p_1(-x_1 - x_2^2 + 5)$$
$$+ p_2(-x_1 + 2)$$



graph 15.3 a illustration of optimality conditions.

The last term $p_3(-x + 3)$ drops out since the restriction is not binding in the optimum, and (15.3.4) tells us that in that case the associated multiplier is $p_3=0$.

We are now in a position to establish the numerical values for p_1 and p_2. From (15.3.3) we obtain

$$\frac{\partial L}{\partial x_1} = 1 - p_1 - p_2 = 0$$

and

$$\frac{\partial L}{\partial x_2} = 1 - 2x_2 \cdot p_1 = 0$$

For $x_2 = \sqrt{3} = 1.73$ the condition on $\frac{\partial L}{\partial x_2}$ gives us $1 - 2 \times 1.73\, p_1 \doteq 1 - 3.46\, p_1 = 0$.

Hence we are able to solve p_1 as $p_1 = 1/3.46 = 0.29$. The other multiplier then is $p_2 = 1 - p_1 = 0.71$. Therefore, in this example, the Lagrangean is

$$\begin{aligned} L &= x_1 + x_2 + 0.29\,(-x_1 - x_2^2 + 5) \\ &\quad + 0.71\,(-x_1 + 2) \\ &= x_2 - 0.29\,x_2^2 + 2.87 \\ &= -0.29\,(x_2 - 1.73)^2 + 3.73 \end{aligned}$$

The first-order derivative of this expression with respect to x_1 does not exist, i.e. is identically equal to zero. This is because the objective function and the restrictions are all linear in x_1. The derivative with respect to x_2 vanishes for $x_2=1.73$.

The Lagrangean $L = -0.29\,(x_2-1.73)^2 + 3.73$ is a convex function. This is systematic, whenever the Lagrangean is a combination of convex functions. For a problem which is formulated in terms of properly convex functions, the Lagrangean always attains an unconstrained maximum at the optimal point. For numbers x_1 and x_2 which satisfy the stated restrictions we establish the inequality

$$x_1 + x_2 \leq 3.73 - 0.29\,(x_2 - 1.73)^2$$

The restriction $x_1 + x_2 \leq 3.73$ can also be obtained in a somewhat different way, as follows:

Take the combined restriction or aggregate restriction

$$p_1(-x_1 - x_2^2 + 5) + p_2(-x_2 + 2)$$
$$= 0.29\ (-x_1 - x_2^2 + 5) + 0.71\ (-x_1 + 2) \leq 0$$

This combined restriction $x_1 + 0.29\ x_2^2 \leq 2.87$ has been drawn in the graph (without shading). The linear restriction $x_1 + x_2 \leq 3.73$ (again without shading) has been drawn in the same graph as well, and is its tangential equivalent.

Some, but not all, of the above properties are true for the optimal solution of a non-convex problem.

John's theorem is more general than the Kuhn-Tucker theorem, but relative to weaker conditions. Unlike the Lagrangean, John's expression may involve a zero multiplier for the objective function.

$$J(\underline{x}) = p_o\tau(\underline{x}) + \sum_{i=1}^{m} p_i\ f_i(\underline{x}) \qquad (15.3.2)$$

$$(p_o \geq 0,\ p_i \geq 0 \quad i=1,2...m \quad \text{not all } p_i=\text{zero})$$

If the multiplier of the objective function is zero, at least one of the multipliers of the restricting functions may be required to be non-zero. (A Lagrangean expression can be just the objective function, i.e. none of the restrictions is binding.)

Otherwise the content of the theorem is the same, i.e. if there is an optimal and feasible solution the first-order derivatives of $J(\underline{x})$ will vanish for the optimal $\underline{x}=\underline{x}^*$ and the appropriate values of the multipliers. But since John does not require convexity, the first-order condition

$$\frac{\partial J}{\partial \underline{x}} = p_o \frac{\partial \tau}{\partial \underline{x}} + \sum_{i=1}^{m} p_i \frac{\partial f_i}{\partial \underline{x}}$$

does not guarantee a maximum.

It does not follow that a first-order stationary solution of the Lagrangean is a maximum of the Lagrangean. Nor does it follow that (in a non-convex problem) a first-order solution indicates the optimum of the programming problem.

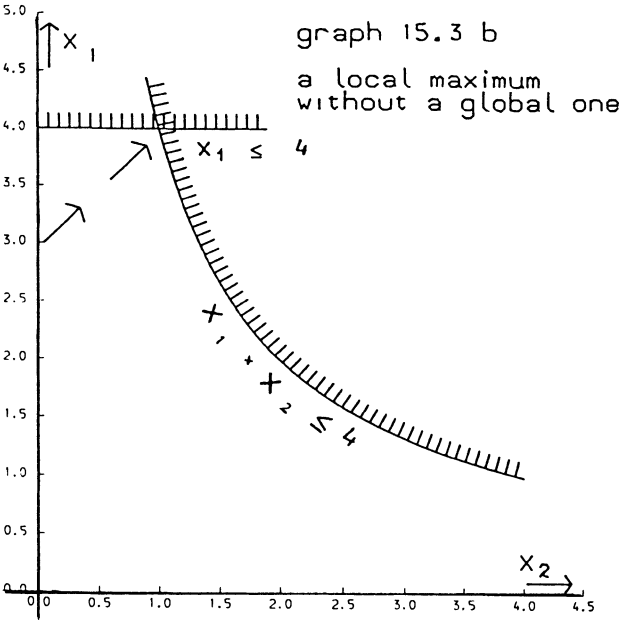
Example

Max. $\tau = 3x_1 + x_2$

Subj. to $x_1 \leq 4$

$x_1 \cdot x_2 \leq 4$

$(x_1, x_2 \geq 0)$



Form the Lagrangean

$$L = 3x_1 + x_2 + p_1(4-x_1) + p_2(4-x_1 \cdot x_2)$$

There is a local maximum at $x_1=4$, $x_2=1$, and we find multipliers in the usual way.

$$\frac{\partial L}{\partial x_1} = 3 - p_1 - p_2 \cdot x_2 = 0 \quad (x_1 > 0)$$

$$\frac{\partial L}{\partial x_2} = 1 - p_2 \cdot x_1 = 0 \quad (x_2 > 0)$$

For $x_1=4$, $x_2=1$ these equations reduce to

$$1 - p_2 \cdot 4 = 0 \rightarrow p_2 = 0.25$$

$$3 - p_1 - p_2 \cdot 1 = 3 - p_1 - 0.25 \rightarrow p_1 = 2.75$$

Therefore the Lagrangean expression is

$$3x_1 + x_2 + 2.75(4-x_1) + 0.25(4-x_1 \cdot x_2)$$

$$= 12 + 0.25\,x_1 + x_2 - 0.25\,x_1 \cdot x_2$$

This is not a convex function, and it has no finite maximum, only a local saddle-point at $x_1=4$, $x_2=1$. And it does not represent a global optimum. The problem is in fact unbounded, because it admits infinite values of x_2. However, if the top of the non-convex "spine" in the x_2-direction is cut off, the same optimality conditions may represent a local maximum which is also the global maximum.

For example, if the restriction $x_2 \leq 4$ is added to the problem, $x_1=4$, $x_2=1$, with $p_1=2.75$, $p_2=0.25$ (as above) is the optimal solution.

The relation between convexity of the Lagrangean and convexity of the programming problem is further complicated by the possibility of directional and/or peripheral convexity (see Section 14.2). A programming problem may in substance "be convex", but if the objective function is formulated as directionally convex but not properly convex, and/or some of the restricting functions as only peripherally convex, the Lagrangean function may be non-convex.

Example

Maximise $\tau = (x_1 + x_2)^2$

Subject to $x_1 \leq 1$

$x_2 \leq 1$

$(x_1 \geq 0, \quad x_2 \geq 0)$

This is substantially a linear programming problem with the maximisation of x_1+x_2 as objective function. The optimal solution obviously is $x_1=1$, $x_2=1$. The associated dual variables are

$$p_1 = p_2 = 4.$$

The Lagrangean expression

$$L = (x_1+x_2)^2 + p_1(1-x_1) + p_2(1-x_2)$$

$$= (x_1+x_2)^2 - 4x_1 - 4x_2 + 8$$

is <u>not</u> a convex function.

It does not even have a local maximum at $x_1=1$, $x_2=1$. Yet $x_1=1$, $x_2=1$ is the true and only optimal solution of the programming problem.

Now suppose that a programming problem is properly convex, both in the objective function and in the restricting functions. In that case the Kuhn-Tucker conditions (15.3.3) to (15.3.5), are sufficient to identify a feasible solution as being the optimum. We saw in section 14.4 that, for a properly convex function, the first-order conditions are sufficient to identify a (possibly non-unique) maximum. The Lagrangean is a linear combination of (by assumption) properly convex functions, i.e. the

objective function and properly convex restricting functions, therefore the Lagrangean is a properly convex function.

It follows that the Kuhn-Tucker conditions identify a maximum of the Lagrangean. We indicate i.e. denote the maximum value of the Lagrangean as $L(\underline{x}^*)$

$$L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i \cdot f_i(\underline{x}) \leq L(\underline{x}^*) \qquad (15.3.7)$$

For vectors $\underline{x}$ which satisfy the specified side-conditions each term $p_i(\underline{x})$ is non-negative. (The restriction is defined as the non-negativity of $f_i(\underline{x})$.) Hence we establish the inequality*

$$\tau(\underline{x}) \leq L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i \cdot f_i(\underline{x}) \leq L(\underline{x}^*) \qquad (15.3.8)$$

The complementary slackness condition implies that in the optimum, the terms $p_i \cdot f_i(\underline{x})$ all vanish. Hence the optimal value of the objective function and the Lagrangean are the same.

Exercise

Solve the contrained maximisation problem:

Maximise $\tau = x_1 + x_2$

Subject to $x_1 \leq 3$, $x_2 \leq 4$, $x_1 + x_2 \leq 4$ $(x_1, x_2 \geq 0)$

Find the optimal values of x_1 and x_2 by graphical mapping. Form the Lagrangean expression, and derive the optimality conditions.

Solve the corresponding values of the dual variables (answer at the head of a further exercise, at the end of the next section.)

---

*Generally this inequality is also valid where, in a non-convex problem, a local maximum is found to be the global maximum. But since the optimality conditions do not themselves tell us whether or not, in a non-convex problem, a local maximum is the global maximum, it is nevertheless necessary to verify by other means whether the global optimum has been attained. Furthermore, in a non-convex problem $L(\underline{x}^*)$ either must be understood to be the maximum value of the Lagrangean attained by any feasible vector $\underline{x}$, or simply as the value of the Lagrangean associated with the optimal solution of the programming problem. In a non-convex problem (but not in a convex problem) there will generally be (non-feasible) vectors $\underline{x}$, for which $L(\underline{x}) \geq L(\underline{x}^*)$ is true.

15.4 Dual variables and right-hand side variations

The dual variables are indicators of the changes in the solution value, which arise as a result of changes in the constants of the restrictions. To show this relationship it is useful to separate exogenous increments in these constants from the initially specified functions.

Hence we denote

$$z_i(\underline{x}) = f_i(\underline{x}) + d_i \tag{15.4.1}$$

Above, $f_i(\underline{x})$ are the originally defined restricting functions, and the d_i are later changes in the constants of these functions, causing the $z_i(\underline{x})$ to become slightly different from the originally specified $f_i(\underline{x})$.

Now consider the optimal solution of an altered programming problem.

Maximise $\tau(\underline{x})$

Subject to $z_i(\underline{x}) \geq 0$ (15.4.2)

$(i=1,2...m,\ \underline{x} \geq 0)$

We formulate the Lagrangean

$$L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i\, z(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i (f_i(\underline{x}) + d_i) \tag{15.4.3}$$

The inequality-relationship (15.3.8) may now be copied and adapted. For the "new" problem this inequality will be

$$\tau(\underline{x}) \leq L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i\, f_i(\underline{x}) + \sum_{i=1}^{m} p_i\, d_i \leq L(\underline{x}^*)$$

$$= \tau(\underline{x}^*) + \sum_{i=1}^{m} p_i\, f_i(\underline{x}^*) + \sum_{i=1}^{m} p_i\, d_i \tag{15.4.4}$$

The inequality (15.4.4) does not by itself depend on the optimality conditions.

It depends on two things, (a) that (for certain multipliers) a maximum of the Lagrangean relative to $\underline{x}$ has been established. That is, for those particular values of the p_i, no other vector $\underline{x}$

yields a higher value of $L(\underline{x})$, than the figure indicated by the right-hand side of (15.4.4). (b) That the solution is feasible, i.e. $p_i\ f_i(\underline{x}) \geq 0$ is true for all i.

Granted these two points, the inequalities

$$\tau(\underline{x}) \leq L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i(f_i(\underline{x}) + d_i) \qquad (15.4.5)$$

i.e. the first part of (15.4.4)

will stay binding as long as the complementary slackness condition is maintained.

Hence we assume that for the changed solution it will still be possible for the same multipliers and the same collection of binding restrictions to maintain

$$\left.\begin{array}{ll} f_i(\underline{x}) = 0 & (p_i \geq 0) \\ \underline{x} \geq 0 & \end{array}\right) \qquad (15.4.6)$$

By assumption all $p_i\ f_i(\underline{x}^*)$ are zero, and the second set of inequalities from (15.4.4) reduces to

$$L(\underline{x}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i(f_i(\underline{x}) + d_i) \leq \tau(\underline{x}^*) + \sum_{i=1}^{m} p_i\ d_i \qquad (15.4.7)$$

The non-active (=amply fulfilled) (non-negativity) restrictions will remain fulfilled even after some changes in the constant non-zero numbers d_i. That, at least, is the case for sufficiently small numbers d_i. If variations in the restrictions i.e. in their constants are considered, the Lagrangean is not only a function of the vector $\underline{x}$, but also of the specified shifts in the constants. Thus, (15.4.7) should be re-written as:

$$L(\underline{x},\underline{d}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i(f_i(\underline{x}) + d_i) \leq \tau(\underline{x}^*) + \sum_{i=1}^{m} p_i\ d_i \qquad (15.4.8)$$

We now evaluate the Lagrangean expression, by first and second-order approximation.

$$L(\underline{x},\underline{d}) = L(\underline{x}^*) + \left[\frac{\partial L}{\partial \underline{x}}\right]' \Delta\underline{x} + \sum_{i=1}^{m} \frac{\partial L}{\partial d_i}\ d_i + \tfrac{1}{2}\Delta\underline{x}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}\right] \Delta\underline{x} \qquad (15.4.9)$$

The first term on the right-hand side of (15.5.9), $L(\underline{x}^*)$ gives the value of $L(\underline{x},\underline{d})$ for $\underline{x}=\underline{x}^*$. The second term $\frac{\partial L}{\partial \underline{x}}\ \Delta\underline{x}$ gives the first-order effect of any changes in the vector $\underline{x}$, $\Delta\underline{x}$ standing for the increment of $\underline{x}$. This term is in fact zero because $\frac{\partial L}{\partial \underline{x}}$ is a zero vector. The third term $\sum_{i=1}^{m} \frac{\partial L}{\partial d_i} \cdot d_i$ gives (the first-order effect of) the changes in the constants of the equations. From (15.5.9) we see that there is no other first order effect as a result of non-zero values of the d_i. This term is in fact $\sum_{i=1}^{m} p_i d_i$.

The last term gives the second order effect and for a convex problem it will be negative (non-positive). Thus, (15.5.9) effectively reduces to

$$L(\underline{x},\underline{d}) = L(\underline{x}^*) + \sum_{i=1}^{m} p_i d_i + \tfrac{1}{2}\Delta\underline{x}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}\right] \Delta\underline{x} \qquad (15.4.10)$$

The Lagrangean is (by first-order approximation) insensitive to any changes in the vector $\underline{x}$, but it is dependent on changes in the constants of the equations. And for small changes in the constant, the (linear) first-order term $\sum_{i=1}^{m} p_i d_i$ will dominate over the second-order effect or any other higher-order effects. If it is possible to find a corresponding adjustment in the variables-vector $\underline{x}$ which keeps the same collection of restrictions binding, thus maintaining the complementary slackness condition, the same conclusion holds for the objective function.

Therefore the dual variables indicate the changes in the objective function which will arise in response to small changes in the constants of the restrictions.

Example

Recall our earlier example (at the beginning of Section 15.3), i.e.

Maximise $\quad x_1 + x_2$

Subject to $\quad x_1 + x_2^2 \leq 5$

$\qquad\qquad x_1 \leq 2$

$\qquad\qquad x_2 \leq 3$

Now suppose that the binding upper bound on x_1 is moved outwards to $x_1 \leq 4$. We found the dual variable associated with this restriction to have the value $p_2 = 0.711$. The original optimum function value was $\tau = x_1 + x_2 = 3.73$. We would therefore expect the new function value to be close to $\tau = 3.73 + 2 \times 0.71 = 5.15$. The value of $\tau = x_1 + x_2 = 5.15$ is the optimal solution of the linear programming problem

$$\begin{aligned} \text{Maximise} \quad & \tau = x_1 + x_2 \\ \text{Subject to} \quad & x_1 + 3.464\, x_2 \leq 8 \\ & x_1 \leq 4 \\ & x_2 \leq 3 \end{aligned}$$

This is the original problem with the following modifications

(a) $x_1 \leq 3$ has been substituted for $x_1 \leq 2$, and

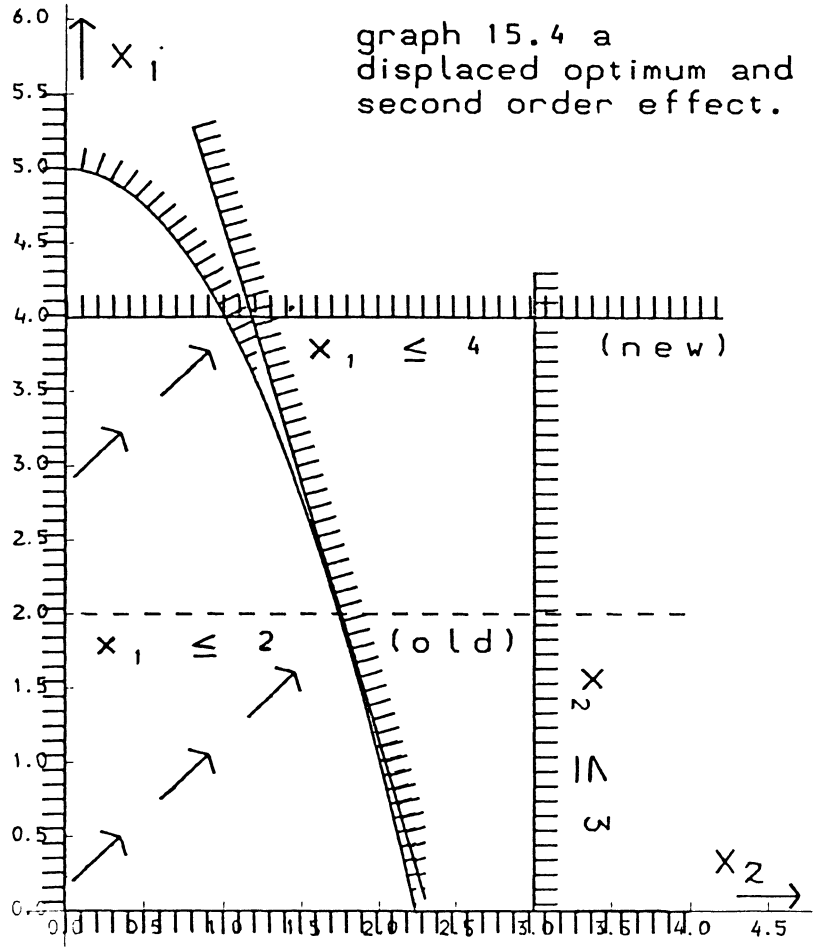
(b) $x_1 + 3.464\, x_2 \leq 8$ has been substituted for $x_1 + x_2^2 \leq 5$.

The dual variable gives the cost in terms of the stated objective function, should a unit of some resource be lost or diverted to other purposes. This is typically the economist's notion of the (marginal) opportunity cost of a unit of the resource. For this reason dual variables are also indicated as imputed prices or shadow prices.

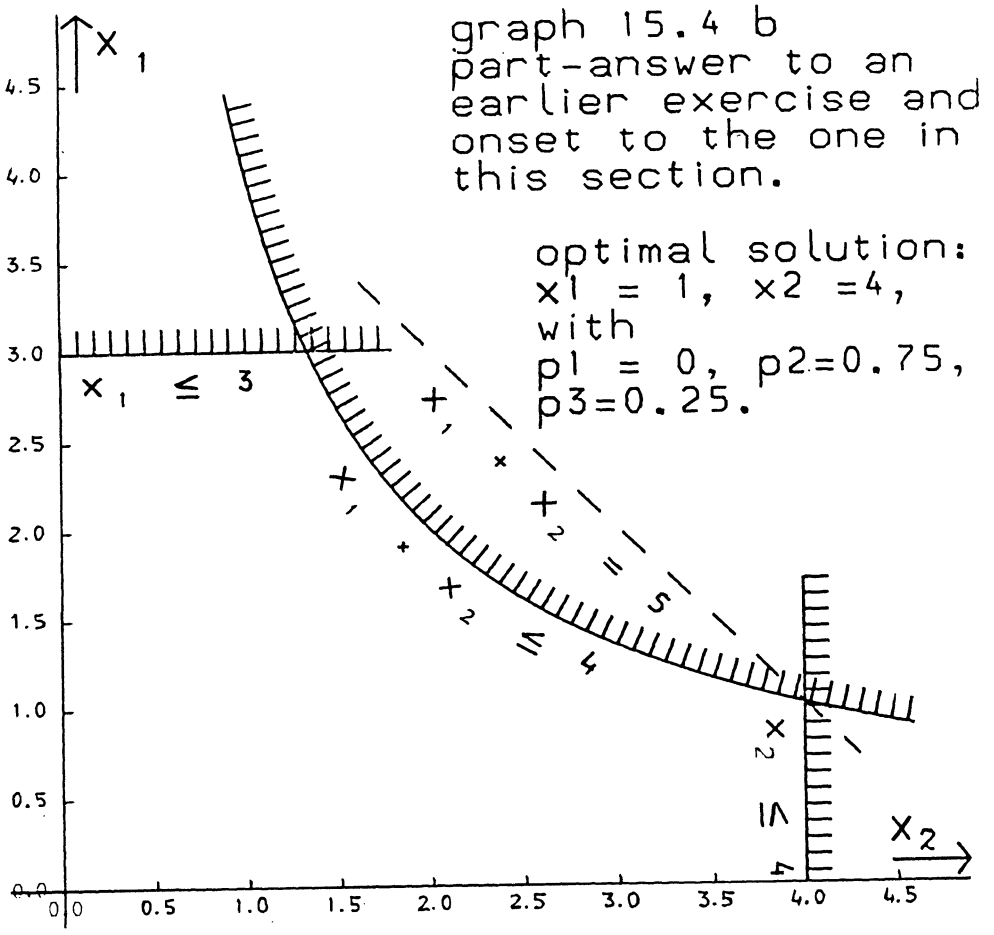
The linear restriction $x_1 + 3.464\, x_2 \leq 8$ is the tangential equivalent of the originally specified nonlinear restriction $x_1 + x_2^2 \leq 5$, the tangential being taken at $x_1=2$, $x_2=\sqrt{3}=1.732$.

The optimal solution of this linear problem is $x_1=4$, $x_2=1.15$. However, the solution follows the inwardly curved surface of the restriction $x_1 + x_2^2 \leq 5$ and the new nonlinear optimum is $x_1=4$, $x_2=1$, and the optimal function value is only 5.00 instead of 5.15. Thus the second-order effect is found to be (minus) 0.15.

At this point, it is appropriate to comment on the economic interpretation of dual variables. In economic allocation problems, the maximand is either a utility function, i.e. the problem is consumers' maximization of their personal satisfaction from a given set of availabilities, or a profit function, i.e. the problem is the firm's maximization of profit within the technical limits of its production installations, or a social welfare function, i.e. the problem is society's maximization of the collective supply of goods and services, within a given set of natural and other resource limits.



In each case the dual variables indicate by how much the specified objection function would increase, if the supply of a particular resource were increased by one unit.



Exercise and answer to a previous exercise

The optimal solution of the exercise at the end of section 15.3 (but not the optimality conditions), is given in Graph 15.4b. Reperform that exercise, changing

$$x_1 \cdot x_2 \leq 4, \text{ to } x_1 \cdot x_2 \leq 8.$$

In this particular case the second-order term in (15.4.10) is systematically zero.

Check that the change in solution value between the two versions of the exercise matches the product of the dual variable of the $x_1 \cdot x_2 \leq 4$ restriction (0.25), and the shift in its constraint (there being no second-order effect).

15.5 Constrained second order conditions: subspace convexity

The necessary first-order conditions discussed in section 15.3 may, or may not indicate a true optimum. Only in convex problems may this be assumed without further investigation, as may be illustrated by reference to the example below.

Example 15.5a

Maximise $\tau = - x_1^2 - x_2^2$

Subject to $(x_1 - 3)^2 + (x_2 - 4)^2 \geq 25$,

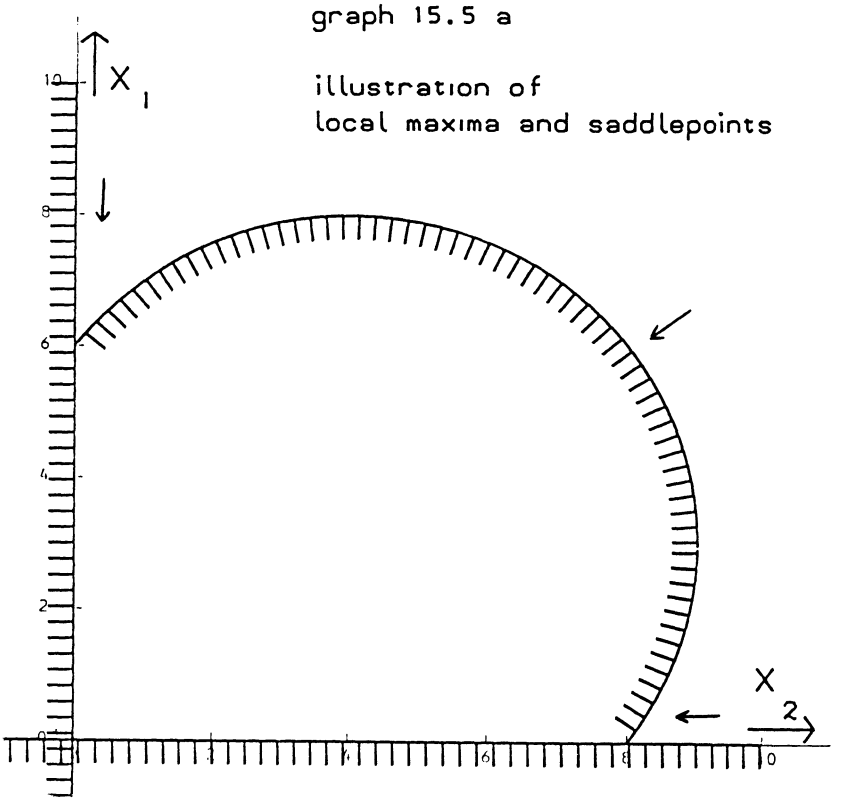
$(x_1, x_2) \geq 0$

In this problem there are four points for which the first-order conditions may be satisfied, viz. ($x_1 = 6$, $x_2 = 0$); ($x_1 = 0$, $x_2 = 8$), ($x_1 = 0$, $x_2 = 8$), and the origin, ($x_1 = 0$, $x_2 = 0$). If we add a restriction to exclude the origin, e.g. $x_1 + x_2 \geq 1$, ($x_1 = 6, x_2 = 0$) is the true optimum with $\tau = -36$. The objective function is in fact the minimization of (the square of) the distance from the origin, and the non-linear restriction is the non-convex outside of a circle with radius 5 and its centre in the point (3,4).

The points (6,0) and (0,8) - and without the additional restriction also (0,0) - , are <u>local maxima</u> i.e. points where any change in the solution vector is either associated with losing objective function value or else with moving out of the feasible space area. A formal definition is as follows:

graph 15.5 a

illustration of
local maxima and saddlepoints



A point $\underline{x} = \underline{x}^*$ is a (strict) local maximum, if it is true for all $\Delta\underline{x} \neq 0$, and $\varepsilon > 0$, but approaching zero, that either,

$$\tau(\underline{x} + \varepsilon \ \Delta\underline{x}) < \tau(\underline{x}) \qquad (15.5.1)$$

or, for some i:

$$f_i(\underline{x} + \varepsilon \ \Delta\underline{x}) < 0 \qquad (15.5.2)$$

($\underline{x} = \underline{x}^*$ being a feasoble solution)

Thus, at (0,8) we may find $\Delta\underline{x}' = 6,-8$ satisfying

$$\tau(\underline{x} + \Delta\underline{x}) = -36 > -64, \text{ and}$$

$f_1(\underline{x} + \Delta\underline{x}) = 0$, but that point is nevertheless a local maximum because when moving from (0,8) to (6,0) we must go through the inadmissable inside of the circle or alternatively, when going around it, begin with moving away from the origin. (The global optimum is also a local maximum, but the use of the term is usually reserved for non-convex problems). The conditions to be discussed in this section do not necessarily distinguish between different local maxima in terms of identifying the true optimum, but they distinguish betweel local maxima and constrained saddle points, where we may start to "climb" at once e.g. at ($x_1 = 6$, $x_2 = 8$). (The first order conditions generally are sufficient to exclude points where one may at once start to climb "steeply"). A constrained saddle point is thus a point where the first order conditions are satisfied, but which is not a local maximum.

To distinguish local maxima from constrained saddle points, we first formulate a first and second order approximation of the Lagrangean.

The following expression is therefore exact if all functions are quadratic, but an approximation - which is generally quite adequate in the vicinity of a first-order solution $\underline{x} = \underline{x}^*$ - if there are non-zero third and higher order derivatives.

$$L(\underline{x}) = L(\underline{x}^* + \Delta\underline{x}) =$$

$$L(\underline{x}^*) + \left(\frac{\partial L}{\partial \underline{x}}(\underline{x}^*)\right)' \Delta\underline{x} + \tfrac{1}{2}\Delta\underline{x}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}(\underline{x}^*)\right] \Delta\underline{x}$$

$$= \tau(\underline{x}^*) + \left(\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right)' \Delta\underline{x} + \tfrac{1}{2}\Delta\underline{x}' \left[\frac{\partial^2 \tau}{\partial^2 \underline{x}}(\underline{x}^*)\right] \Delta\underline{x}$$

$$+ \sum_{i=i}^{m} p_i f_i(\underline{x}^*) + \sum_{t=1}^{m} p_i \left(\frac{\partial f_i}{\partial \underline{x}}(\underline{x}^*)\right)' \Delta\underline{x}$$

$$+ \tfrac{1}{2} \sum_{i=i}^{m} p_i \Delta\underline{x}' \left|\frac{\partial^2 f_i}{\partial^2 \underline{x}}\right| (\underline{x}^*) \Delta\underline{x} \qquad (15.5.3)$$

For the sake of uniformity and simplicity of notation we will assume that the non-negativity restrictions have also been listed

as explicit restrictions. Otherwise it should be understood that $\underline{x}$ is the vector of non-zero variables only - or that non-negativity restrictions do not apply.

If $\underline{x} = \underline{x}^*$ satisfies the first order conditions, the first two members of (15.5.3) collapse to

$$L(\underline{x}) = L(\underline{x}^* + \Delta\underline{x}) = L(\underline{x}^*) + \tfrac{1}{2}\,\Delta\underline{x}'\left[\frac{\partial^2 L}{\partial^2 \underline{x}}\,(\underline{x}^*)\right]\Delta\underline{x} \qquad (15.5.4)$$

(Refer also to section 2.18, concerning the rules for differentiating matrix expressions).

If the quadratic form $\tfrac{1}{2}\,\Delta\underline{x}'\left[\frac{\partial^2 L}{\partial^2 \underline{x}}\,(\underline{x}^*)\right]\Delta\underline{x}$ is non-positive

for all $\Delta\underline{x}$, this confirms that $\underline{x} = \underline{x}^*$ is the optimum. Note that this conclusion does not depend on the convexity of any individual function, as may be illustrated by referring to the example below.

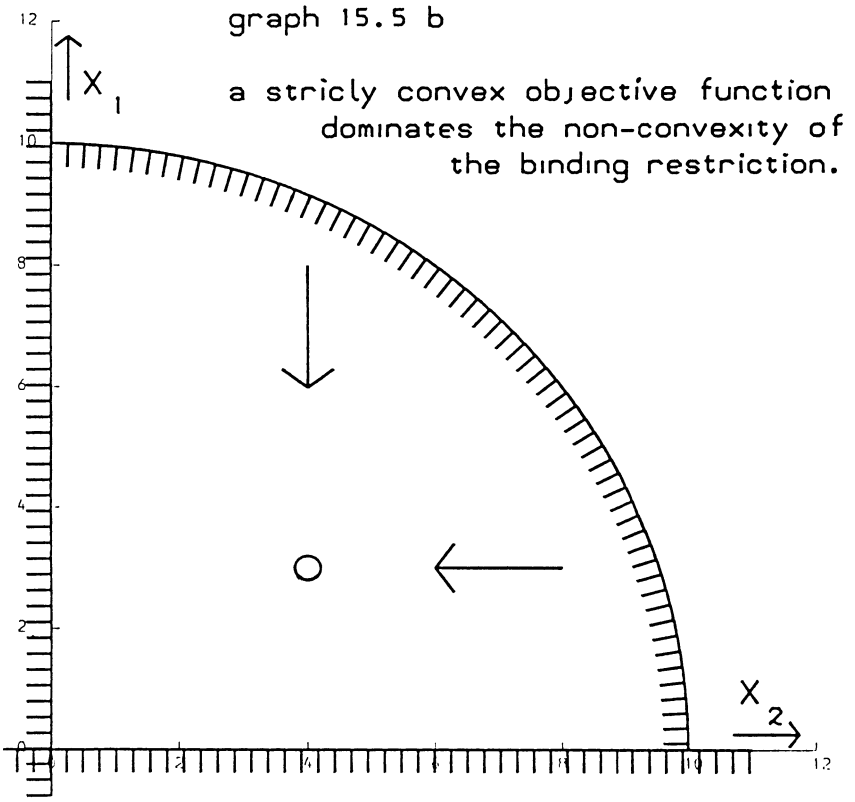
Example 15.5b

Maximise $\quad \tau = -(x_1 - 3)^2 - (x_2 - 4)^2$

Subject to $\quad x_1^2 + x_2^2 \geq 100, \quad x_1, x_2 \geq 0.$

Here the objective function is the minimisation of (the square of) the distance from the point ($x_1 = 3$, $x_2 = 4$), and the non-linear restriction is the non-convex outside of a circle with a radius 10 and the centre at the origin.

Compared with example 15.5a, there has been a kind of interchange between the objective function and the binding restriction and ($x_1 = 6$, $x_2 = 8$) is now the global optimum, as may be shown by developing the Lagrangean expression:



$$L = -(x_1 - 3)^2 - (x_2 - 4)^2 + p_1 (x_1^2 + x_2^2 - 100)$$

$$\frac{\partial L}{\partial x_1} = -2 (x_1 - 3) + 2 p_1 x_1 = 0 \qquad (x_1 = 6 > 0)$$

$$\frac{\partial L}{\partial x_2} = -2 (x_2 - 4) + 2 p_1 x_2 = 0 \qquad (x_2 = 8 > 0)$$

From which we solve for $x_1 = 6$, $x_2 = 8$, p_1 as $p_1 = 0.5$.

This permits us to numerically evaluate the Lagrangean as:

$$L(x_1,x_2) = - (x_1 - 3)^2 - (x_2 - 4)^2 + 0.5\, x_1^2 + 0.5\, x_2^2 - 50$$

$$= - \tfrac{1}{2} (x_1 - 6)^2 - \tfrac{1}{2} (x_2 - 8)^2 - 25.$$

This is a strictly convex function which attains a unique maximum of -25 at ($x_1 = 6$, $x_2 = 8$).

That no feasible solution vector can assign any higher value to τ, then follows immediately from the definition of a Lagrangean function i.e. for feasible solutions the value of the objective function does not exceed that of the Lagrangean.

Convexity of the Lagrangean throughout the coordinate space is not, however a necessary condition for a constrained maximum, certainly not if the term maximum includes local maxima. (see also the example in section 15.3).

To derive necessary and sufficient conditions we must analyze the possibility that the quadratic form in (15.5.4) is restricted to non-negative values for those vectors $\Delta\underline{x}$ which respect the side-conditions.

To this purpose we describe any vector $\Delta\underline{x}$ as

$$\Delta \underline{x} = \lambda \underline{v} + \delta \underline{q} \qquad (15.5.5)$$

where $\underline{v}$ is required to satisfy (for all i)

$$\left[\frac{\partial f_i}{\partial \underline{x}} (\underline{x}^*)\right]' \underline{v} = 0 \qquad (15.5.6)$$

and therefore, if $\underline{x} = \underline{x}^*$ satisfies the first order conditions, also,

$$\left[\frac{\partial \tau}{\partial \underline{x}} (\underline{x}^*)\right]' \underline{v} = 0 \qquad (15.5.7)$$

The presence of the scalars λ and δ permits us to normalize $\underline{v}$ and $\underline{q}$, requiring:

$$\left.\begin{array}{l} \underline{v}'\,\underline{v} = 1 \\ \\ \text{and} \\ \\ \underline{q}'\,\underline{q} = 1 \end{array}\right) \qquad (15.5.8)$$

It will be noted that, in the absence of any restriction on δ, or on the relative values of the elements of q (nor for that matter, so far on λ), no requirement on $\Delta\underline{x}$ has so far been imposed.

To test whether $\underline{x} = \underline{x}^*$ is optimal, we now require that $\underline{q}$ does not satisfy the equivalent of (15.5.7), and that both λ and δ approach zero, not both of them actually being zero.

These conditions make $\lambda\underline{v} + \delta\underline{q}$ equivalent to $\varepsilon\,\Delta\,\underline{x} \neq 0$ as used in (15.5.1) and (15.5.2). (The difference between the condition $\varepsilon > 0$ and $\lambda \neq 0$ or $\delta \neq 0$ is immaterial as no restriction on the signs of $\Delta\underline{x}$ or of $\underline{v}$ and $\underline{q}$ has been put. The condition that $\underline{q}$ is not merely another vector $\underline{v}$ is not actually necessary but restricts the substantial case $\delta = 0$, to the formal case $\delta = 0$.

We now re-formulate the condition for a local maximum by substituting $\lambda\underline{v} + \delta\underline{q}$ for $\varepsilon\,\Delta\,\underline{x}$ in (15.5.1) and (15.5.2) and putting the quadratic approximation used in (15.5.3). $\underline{x} = \underline{x}^*$ is a strict local maximum, if
either:

$$\Delta\tau = \left(\frac{\partial\tau}{\partial\underline{x}}(\underline{x}^*)\right)'(\lambda\underline{v} + \delta\underline{q}) + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})'\left[\frac{\partial^2\tau}{\partial^2\underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta\underline{q}) < 0 \tag{15.5.9}$$

or

$$\Delta f_i = \left(\frac{\partial f_i}{\partial\underline{x}}(\underline{x}^*)\right)'(\lambda\underline{v} + \delta\underline{q}) + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})'\left[\frac{\partial^2 f_i}{\delta^2\underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta\underline{q}) < 0 \quad \text{(some i)} \tag{15.5.10}$$

applies for all $\underline{v}$ and $\underline{q}$ satisfying (15.5.6) and (15.5.8), but $\underline{q}$ not satisfying the equivalent of (15.5.6) with both λ and δ approaching zero (but at least one of them not actually being zero).

By the first-order conditions (15.5.9) and (15.5.10) reduce to (15.5.11) and (15.5.12)

$\underline{x} = \underline{x}^*$ is a local maximum, if

$$\Delta\tau = \left(\frac{\partial\tau}{\partial\underline{x}}(\underline{x}^*)\right)'\,\delta\underline{q} + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})'\left[\frac{\partial^2\tau}{\partial^2\underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta\underline{q}) < 0 \tag{15.5.11}$$

or

$$\Delta f = \left(\frac{\partial f_i}{\partial \underline{x}}(\underline{x}^*)\right)' \delta\underline{q} + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})' \left[\frac{\partial^2 f_i}{\partial^2 \underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta\underline{q}) < 0 \qquad \text{(some i)} \tag{15.5.12}$$

applies for all $\underline{v}$ and $\underline{q}$ satisfying (15.5.6) and (15.5.8), $\underline{q}$ not satisfying the equivalent of (15.5.6), λ and δ both approaching zero, but at least one of them not actually being zero.

These conditions may also be formulated equivalently, by their negation:

$\underline{x} = \underline{x}^*$ is not a (strict) local maximum if

$$\Delta \tau = \left(\frac{\partial t}{\partial \underline{x}}(\underline{x}^*)\right)' \delta\underline{q} + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})' \left[\frac{\partial^2 \tau}{\partial^2 \underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta q) \geq 0 \tag{15.5.13}$$

and

$$\Delta f_i = \left(\frac{\partial f_i}{\partial \underline{x}}(\underline{x}^*)\right)' \delta\underline{q} + \tfrac{1}{2}(\lambda\underline{v} + \delta\underline{q})' \left[\frac{\partial^2 f_i}{\partial^2 \underline{x}}(\underline{x}^*)\right](\lambda\underline{v} + \delta\underline{q}) \geq 0 \qquad \text{(all i)} \tag{15.5.14}$$

applies for some $\underline{v}$ and $\underline{q}$ satisfying (15.5.6) and (15.5.8) λ and δ both approaching zero, not both of them actually being zero.

Note that this definition does not accept non-unique local maxima where the solution value is the same when going for some length along a combination of binding restrictions, hence the word strict local maximum.

We now first of all have the following

Theorem (corner theorem)

If all functions are anti-convex (the objective function as well as the restricting functions, the matrix

$$\left[\frac{\partial \underline{f}}{\partial \underline{x}}(\underline{x}^*)\right]$$

may be required to be square, of the same order as the number of elements of $\underline{x}$ (counting non-negativity restrictions as explicit restricting functions), and non-singular.

Proof

For $\delta = 0$, $\lambda \neq 0$ (15.5.11) and (15.5.12) cannot be satisfied if all the quadratic forms are positive semi-definite. Therefore the assumption that $\underline{x} = \underline{x}^*$ is the optimum implies that the set of non-zero vectors $\underline{v}$ satisfying (15.5.6) is empty,

q.e.d.

(N.B. The words "may be required" relate to the possibility that more than n restrictions are binding on the optimum solution.)

The reader will note that the familiar LP situation, where we only need to investigate the corners of the feasible space area, is covered by this theorem.

On the other hand, if the solution is not in a corner we have the following

Lemma

If $\underline{x} = x^*$ satisfies the first order conditions then

$$\left[\frac{\partial \underline{f}}{\partial \underline{x}}(\underline{x}^*)\right] \underline{q} \geq 0$$

implies

$$\left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right]' \underline{q} \leq 0$$

Proof

By the first order condition $\left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right]'$ may be expressed as

$\left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^*)\right]' = -\underline{p}'\left[\frac{\partial \underline{f}}{\partial \underline{x}}(\underline{x}^*)\right]$, from which the result, given the non-negativity of both $\underline{p}'$ and $\left[\frac{\partial \underline{f}}{\partial \underline{x}}(\underline{x}^*)\right] \underline{q}$.

q.e.d.

Therefore except in the excluded trivial case that $\underline{q}$ is in fact another vector $\underline{v}$, the negative formulation of the definition of a local maximum (15.5.13) and (15.5.14) can only be met in the case of a non-zero contribution of a quadratic form.

However, if λ approaches zero, we can only assume that (15.5.13) and (15.5.14) are met on account of a non-zero quadratic form (and violated if a quadratic form is deleted) if the ratio δ/λ also approaches (or becomes) zero.

Therefore a necessary condition for non-optimality is that (for $\delta = 0$)

$$\Delta L = \tfrac{1}{2}\, \underline{v}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}\ (\underline{x}^*)\right] \underline{v} \geq 0$$

for some $\underline{v}$ satisfying (15.5.6).

Conversely for $\delta = 0$ the positive formulation (15.5.11) and (15.5.12) shows that

$$\Delta L = \tfrac{1}{2}\, \underline{v}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}\ (\underline{x}^*)\right] \underline{v} < 0$$

to hold for all $\underline{v}$ satisfying (15.5.6) is a necessary condition for optimality. The latter condition has therefore been shown to be both necessary and sufficient for optimality.

It is stated here without further proof that one may normally assume that

$$\Delta L = \tfrac{1}{2}\, \underline{v}' \left[\frac{\partial^2 L}{\partial^2 \underline{x}}\ (\underline{x}^*)\right] \underline{v} \leq 0$$

for all $\underline{v}$ satisfying (15.5.6) is a necessary (and normally sufficient) condition for a local maximum without the word "strict" added to it. Complications with non-zero third higher order derivative may, however, arise.

Example 15.5c

(Of the constrained second order conditions)

Maximise $\quad \tau = x_1^{\,2} + x_2^{\,2}$

Subject to $\quad (x_1 - 10)(x_2 - 10) \geq 4$

$$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \; x_1,\ x_2 \geq 0$$

$(x_1 \leq 9, \quad x_2 \leq 9$

In this problem, the objective function is anti-convex, and the one non-linear restriction is peripherally convex in the $x_1 \leq 10$, $x_2 \leq 10$ domain.

Graphical analysis (the actual drawing is left to the reader; the non-linear restriction is a hyperpola with rectangular axes $x_1 = 10$, and $x_2 = 10$, the objective function is the (square of) the distance from the origin) indicates that the optimum is $x_1 = x_2 = 8$.

This is obviously a method which is not generally available in the n-dimentional case, and we seek algebraic confirmation.

We develop the Lagrangean

$$L = x_1^2 + x_2^2 + p_1 ((x_1 - 10)(x_2 - 10) - 4)$$

The zero valued multipliers associated with the amply fulfilled linear restrictions have been left out from this expression already.

The first order conditions are

$$\frac{\partial L}{\partial x_1} = 2x_1 + p_1 (x_2 - 10) = 0 \quad (x_1 = 8 > 0)$$

$$\frac{\partial L}{\partial x_2} = 2x_2 + p_1 (x_1 - 10) = 0 \quad (x_2 = 8 > 0)$$

and we solve the multiplier as $p_1 = 8$.

The Lagrangean may now be evaluated numerically as

$$L = x_1^2 + x_2^2 + 8 ((x_1 - 10) (x_2 - 10) - 4)$$

and the associated matrix of second order derivatives as

$$\left[\frac{\partial^2 L}{\partial^2 \underline{x}}\right] = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}$$

This is an indefinite matrix. To verify whether or not it is associated with a constrained maximum, we need to define the set of admissable vectors $\underline{v}$ to figure in (15.5.6).

The linear approximation of the side conditions at $x_1 = x_2 = 8$ is

$$- 2\, dx_1 - 2\, dx_2 = 0$$

i.e. can only be satisfied if the changes in x_1 and x_2 are of the same absolute value but opposite sign.

We evaluate the quadratic form as

$$\begin{bmatrix}1 & -1\end{bmatrix}\begin{bmatrix}2 & 4\\ 4 & 2\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix} = \begin{bmatrix}-2 & 2\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix} = -4 < 0$$

and conclude that $x_1 = 8$, $x_2 = 8$ is indeed a constrained (local) maximum. Note however, that we cannot by this calculation alone deduce that it is the global maximum:

f the restrictions on x_1 and x_2 separately are replaced by $x_1 \leq 20$, $x_2 \leq 20$, the global optimum is $x_1 = x_2 = 20$, but the local maximum at $x_1 = x_2 = 8$ is not affected by such a re-definition of the problem, it still satisfies the conditions for a constrained local maximum.

Before finalizing this section, it is useful to mention an equivalent second order condition for a constrained local maximum. There is actually an older tradition in analysis and mathematical economics (see for example P.A. Samuelson [32] pp 61-69), which tends to formulate the second order conditions in terms of determinants rather than quadratic forms.

To that purpose we form the composite matrix

$$M = \left[\begin{array}{c|c} -\frac{\partial^2 L}{\partial^2 \underline{x}} & (-\frac{\partial \underline{f}}{\partial \underline{x}})' \\ \hline \frac{\partial \underline{f}}{\partial \underline{x}} & \end{array}\right]$$

The second order condition then relates to the principal minors (= determinants of square diagonal blocks) of M, insofar as the corresponding minor-matrices include the zero bottom righthand block. The second order condition for a constrained maximum then amounts to the non-negativity of each of these principal minors including $|M|$ itself.

For a unique maximum we may also require $|M| > 0$.

Example

Maximise $\tau = x_1 \cdot x_2$

Subject to $x_1 + x_2 \leq 10 \quad (x_1, x_2 \geq 0)$

This problem has an optimum of $x_1 = x_2 = 5$, with $p_1 = 5$.

The corresponding matrix M for that example is as follows:

$$M = \left[\begin{array}{cc|c} 0 & -1 & 1 \\ -1 & 0 & 1 \\ \hline -1 & -1 & 0 \end{array}\right]$$

and the second order condition is verified as follows:

$$|M| = 2 > 0.$$

Apart from the trivial case of the 3,3 element (and in general the bottom righthand block itself) which obviously gives rise to a zero principal minor, M has 5 principal minors viz: the 1,1 diagonal element, the 2,2 diagonal element the determinant of top lefthand block $\left|-\frac{\partial^2 L}{\partial^2 \underline{x}}\right| = \begin{vmatrix} 0 & -1 \\ -1 & 0 \end{vmatrix}$

and the blocks which arise from deleting the first column and row, and similarly the second column and row. The first three of these do not count, as their minor matrices do not include the bottom righthand element.

The remaining relevant minors are:

$$M_{1,1} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} = 1 > 0, \text{ and}$$

$$M_{2,2} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} = 1 > 0,$$

Confirming that $x_1 = x_2 = 5$ is a local maximum.

Note that we do <u>not</u> require $|M_{33}| = \left|-\frac{\partial^2 L}{\partial^2 \underline{x}}\right| \geq 0$ as indeed, is not the case. Such a condition would imply positive semi-definiteness of $\left[-\frac{\partial^2 L}{\partial^2 \underline{x}}\right]$ (= negative semi-definiteness of

$\left[\frac{\partial^2 L}{\partial^2 \underline{x}}\right]$, the Lagrangean to be convex throughout the coordinate space, that would be a too strong condition. We also observe that the stated condition becomes trivial in the case of a corner solution. If $\left[\frac{\partial \underline{f}}{\partial \underline{x}}\right]$ is square and non-singular, $|M|$ is evaluated as $|M| = \left|\frac{\partial \underline{f}}{\partial \underline{x}}\right|^2$, and all the relevant minors vanish on account of the block triangular structure of the minor matrix.

A proof of the equivalence between the determinantal condition and the one on the constrained quadratic form may be obtained by means of analyzing the possible pivots which might be applied to invert M. (Readers who might wish to derive this proof may find it helpful to re-read sections 5.8, 14.4 and 14.6, and to read section 16.6 before pursuing this point.)

Exercise 15.5.a

The following problem is formulated:

Maximise $\tau = x_1^2 + x_1 x_2 + x_2$

Subject to $4x_1 + x_1 x_2 \leq 18$

$(x_1, x_2 \geq 0)$

It is suggested that $x_1 = 3$, $x_2 = 2$ is the optimum solution of this problem, or at least a constrained local maximum.

Verify the first and second order conditions, i.e. whether this is so.

(Answer 15.5.a at the end of the chapter)

15.6 Non-linear duality

In this section we discuss a theorem which is originally due to Dantzig, Eisenberg and Cottle [9], and is reported here on the authority of A. Whinston. [40]

To conform with presentation conventions in this book, the theorem has been re-formulated. These presentation conventions require, inter alia,

a) that the original problem (i.e. the "primal") problem is a maximisation problem, and that

b) the specified primal variables are indicated as $\underline{x}$, and the specified dual variables are indicated as $\underline{p}$.

We put the usual mathematical programming problem, as follows:

Maximise

$$\tau = \tau(x) \tag{14.1.1}$$

Subject to

$$f_i(\underline{x}) \geq 0 \qquad (i = 1, 2 \ldots\ldots m) \tag{14.1.2}$$

$$(x \geq 0)$$

The associated Lagrangean expression therefore is:

$$L(\underline{x},\underline{p}) = \tau(\underline{x}) + \sum_{i=1}^{m} p_i f_i(\underline{x}) = \tau(\underline{x}) + \underline{p}'\underline{f} \tag{15.6.1}$$

We now formulate this primal problem as

Maximise

$$\lambda = L(\underline{x}, \underline{p}) - \underline{p}' \frac{\partial L}{\partial \underline{p}} \tag{15.6.2}$$

Subject to

$$\frac{\partial L}{\partial \underline{p}} \geq 0 \tag{15.6.3}$$

$$(\underline{x} \geq 0, \underline{p} \geq 0)$$

This is the same programming problem, because $\frac{\partial L}{\partial \underline{p}}$ is the vector of function-values $f_i(\underline{x})$

$$\frac{\partial L}{\partial \underline{p}} = \underline{f} \tag{15.6.4}$$

Hence the term $-\underline{p}' \frac{\partial L}{\partial \underline{p}}$ in (15. .2) reduces the Lagrangean to the objective function

$$L(\underline{x}, \underline{p}) - \underline{p}' \frac{\partial L}{\partial \underline{p}} = \tau(\underline{x}) \tag{15.6.5}$$

and (15.6.3) are simply the primal requirements

$$\underline{f} \geq 0 \tag{15.6.6}$$

which is (14.1.2) in vector notation.

The dual problem then is

Minimise

$$\mu = L(\underline{x}, \underline{p}) - \underline{x}' \frac{\partial L}{\partial \underline{x}} \tag{15.6.7}$$

Subject to

$$\frac{\partial L}{\partial \underline{x}} \leq 0 \tag{15.6.8}$$

$$(\underline{x} \geq 0, \ \underline{p} \geq 0)$$

The theorem then states that the optima of these two problems are characterized by (the complementary slackness conditions) $\underline{p}' \frac{\partial L}{\partial \underline{p}} = 0$ and $\underline{x}' \frac{\partial L}{\partial \underline{x}} = 0$, with the same optimal solution vectors $\underline{x}$, $\underline{p}$ figuring in both problems.

Answer 15.5a

We first form the Lagrangean

$$L = x_1^2 + x_1 x_2 + x_2 + p_1 (18 - 4 x_1 - x_1 x_2)$$

The first order conditions are

$$\frac{\partial L}{\partial x_1} = 2 x_1 + x_2 - 4 p_1 - p_1 x_2 = 0 \quad (x_1 = 3 > 0)$$

$$\frac{\partial L}{\partial x_2} = x_1 + 1 - p_1 x_1 = 0 \quad (x_2 = 2 > 0)$$

Therefore, optimality of this solution requires (by the condition on x_2): $p_1 = (x_1+1)/x_1 = 4/3 = 1\ 1/3$

The solution $x_1 = 3$, $x_2 = 2$, $p_1 = 1\ 1/3$ also satisfies the condition on x_1,

$$2x_1 + x_2 - 4p_1 - p_1 x_2 = 6 + 2 - 16/3 - 8/3 = 0$$

therefore the first order conditions are satisfied

We now evaluate the Lagrangean numerically as

$$L = x_1^2 + x_1 x_2 + x_2 + 1\ 1/3(18 - 4 x_1 - x_1 x_2)$$

and the matrix of second order derivatives as

$$\left[\frac{\partial^2 L}{\partial \underline{x}}\right] = \left[\begin{array}{c|c} 2 & \frac{1}{2}(1-p_1) \\ \hline \frac{1}{2}(1-p_1) & - \end{array}\right] = \begin{bmatrix} 2 & -1/6 \\ -1/6 & - \end{bmatrix}$$

and the side-conditon $\left[\frac{\partial f}{\partial \underline{x}}(\underline{x}^*)\right] d\underline{x} = 0$ as

$$-(4 + x_2)\, dx_1 - x_1\, dx_2 = -6\, dx_1 - 3\, dx_2 = 0$$

This condition is satisfied only by changes in x_1 and x_2 being in the proportions vector $\underline{v}' = [1 \quad -2]$.

The quadratic form $\underline{v}' \frac{\partial^2 L}{\partial \underline{x}}(\underline{x}^*)\, \underline{v}$ is now evaluated as:

$$[1 \quad -2] \begin{bmatrix} 2 & -1/6 \\ -1/6 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \end{bmatrix} = [2\ 1/3 \quad -1/6] \begin{bmatrix} 1 \\ -2 \end{bmatrix} = 2\ 2/3 > 0$$

showing non-optimality of $x_1 = 3$, $x_2 = 2$.

Part IV

QUADRATIC PROGRAMMING

CHAPTER XVI

QUADRATIC PROGRAMMING WITH LINEAR RESTRICTIONS

16.1 Statement of the Problem

Let $\underline{x}$ be an n by 1 column vector of unknown variables. $\underline{w}'$ a 1 by n row-vector of known linear preference coefficients, D a known matrix of quadratic preference coefficients. Let τ be a scalar variable indicating the value of the preference-function, or objective function.

A general quadratic objective function is then specified as

$$\tau = \underline{w}' \underline{x} + \tfrac{1}{2} \underline{x}' D\underline{x} \qquad (16.1.1)$$

where the inner product $\underline{w}'\underline{x}$ is the linear component, the quadratic form $\frac{1}{2} \underline{x}' D\underline{x}$ the non-linear component. Following section 14.4, we assume that D is symmetric. If we had originally specified a non-symmetric preference matrix we may always replace this by a symmetric one, by taking the average between the matrix itself and its transpose. For example $x_1^2 + 2x_1x_2 + x_2^2$ may be written as

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{or as}$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{and we might as well write}$$

$$D = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Generally,

$$\underline{x}' D \underline{x} = \underline{x}' \left[\tfrac{1}{2}D + \tfrac{1}{2}D'\right] \underline{x} \qquad (16.1.2)$$

is true by the definition of a quadratic form. (see section 14.4). To ensure that a first-order solution actually is the optimum, we may (or may not) require that D is negative semi-definite.

$$\underline{x}' D \underline{x} \leq 0 \qquad (16.1.3)$$

for all $\underline{x}$

This requirement of negative semi-definiteness implies that we are dealing with a convex preference function. Some implications of relaxing the requirement of negative semi-definiteness will be discussed at a later stage.

The algorithm to be discussed in this chapter, deals with a quadratic objective function, but only with linear side-conditions. These linear side-conditions are the usual block of inequalities, similar to the restrictions in linear programming.

$$A\,\underline{x} \leq \underline{b} \qquad (7.2.2)$$

and the specified programming problem is

Maximise

$$\tau = \underline{w}'\,\underline{x} + \tfrac{1}{2}\,\underline{x}'\,D\,\underline{x} \qquad (16.1.1)$$

Subject to

$$A\,\underline{x} \leq \underline{b} \qquad (7.2.2)$$

In (7.2.2), A will be an m by n matrix of known coefficients, $\underline{b}$ an m by 1 vector of known constants, and we normally assume $\underline{x} \geq 0$.

16.2 The Optimality Conditions

Following Dantzig [8], section 24.4, Van de Panne and Whinston [36], and Cottle [6], we first state the optimality conditions. An m by 1 vector of Lagrangean multipliers is denoted by $\underline{p}$ and the Lagrangean expression is:

$$L = \underline{w}'\,\underline{x} + \tfrac{1}{2}\underline{x}'\,D\underline{x} + \underline{p}''\,(\underline{b} - A\underline{x}) \qquad (16.2.1)$$

The necessary first-order conditions are then, besides (7.2.2) and the complementary slackness condition

$$\frac{\partial L}{\partial \underline{x}} = \underline{w}' + \underline{x}'\,D - \underline{p}'A \leq 0 \quad (\underline{p}' \geq 0) \qquad (16.2.2)$$

or,
presented column-wise as a block-inequality

$$D\underline{x} - A'\underline{p} \leq -\underline{w} \qquad (16.2.3)$$

We will assume that $\underline{x}$ is restricted to non-negative values only.

The quadratic programming problem can now be presented in tabular form. We seek a permissable solution to the combined system.

$$\left.\begin{array}{ll} D\underline{x} - A'\underline{p} \leq -\underline{w} & (16.2.3) \\ A\underline{x} \leq \underline{b} & (7.2.2) \end{array}\right) \underline{x} \geq 0,\ \underline{p} \geq 0$$

The solution to this system is also required to satisfy the remainder of the Kuhn-Tucker conditions, i.e. the complementary slackness condition that an element of $\underline{p}$, a dual variable, is only allowed to attain a non-zero value if the corresponding primal restriction in (7.2.2) is exactly binding, and an element of $\underline{x}$ is only allowed to attain a non-zero value if the corresponding dual restriction in (16.2.3) is binding.

At this point it is useful to again discuss the second-order conditions for a constrained maximum.

The requirement that D is negative semi-definite combined with the necessary first-order conditions, is a sufficient condition to show that a particular vector $\underline{x}$ corresponds to a maximum. But this condition is <u>not</u> a necessary condition.

Consider the following example

Maximise $\tau = x_1 \cdot x_2$

Subject to $x_1 + x_2 \leq 10$

$(x_1, x_2 \geq 0)$

This problem <u>has</u> a unique optimal and feasible solution, $x_1 = x_2 = 5$. At this point $x_1 \cdot x_2 = 25$, whereas for example $x_1 = 4$, $x_2 = 6$ yields $x_1 \cdot x_2 = 24$.

Yet $D = \begin{bmatrix} - & 1 \\ 1 & - \end{bmatrix}$ is not a negative semi-definite matrix since

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} - & 1 \\ 1 & - \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 > 0$$

It is an <u>in</u>definite quadratic form.

However, the only variation in the x_1, x_2 vector which we should investigate is $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ i.e. x_1 increasing at the cost of x_2 or

vice-versa; other directions violate the binding side-condition.

Under that side-condition, the quadratic form $\begin{bmatrix} \Delta x_1 & \Delta x_2 \end{bmatrix} \begin{bmatrix} - & \frac{1}{2} \\ \frac{1}{2} & - \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}$

in indeed negative definite.

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} - & \frac{1}{2} \\ \frac{1}{2} & - \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 < 0$$

or, if the sign is reversed (x_2 increasing at the cost of x_1),

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} - & \frac{1}{2} \\ \frac{1}{2} & - \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = -1 < 0$$

To avoid describing local minima or constrained saddle-points, we introduce the notion of subspace convexity.

The existence of an optimal solution to the problem as a whole implies the existence of an optimal solution to the sub-problem.

Maximise $$\tau_1(x_1) = \underline{w}_1' \underline{x}_1 + \tfrac{1}{2} \underline{x}_1' D_{11} \underline{x}_1 \qquad (16.2.4)$$

Subject to $$A_{11} \underline{x}_1 = \underline{b}_1 \qquad (16.2.5)$$

where $\underline{x}_1$ is the vector of variables which the optimal solution describes as being non-zero, $\underline{w}_1'$ the corresponding vector of linear preference coefficients, D_{11} the corresponding block of quadratic preference coefficients, A_{11} the intersection of the block-row which describes the binding restrictions, with the block column which relates to $\underline{x}_1$, while $\underline{b}_1$ are the corresponding elements of $\underline{b}$.

Now refer to section 15.5 and note that the second-order derivatives of the Lagrangean are (with linear side-conditions), simply those of the objective function.

The second order conditions for a constrained maximum of this sub-problem therefore are

$$\underline{v}_1' D_{11} \underline{v}_1 \leq 0, \qquad (16.2.6)$$

for all $\underline{v}_1$ satisfying

$$A_{11}\,\underline{v}_1 = 0 \qquad (16.2.7)$$

We will express this condition by saying that the objective function is convex within the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace.

In anticipation of finding an optimal solution which satisfies this condition, we impose it also on each current solution, at least as far as this concerns solutions for which the complementary slackness conditions has been verified. Moreover, it will be shown in section 16.6, that once we require subspace convexity, the existence of an invertable block-pivot also imples the more stringent condition of strict subspace convexity.

$$\underline{v}_1'\,D_{11}\,\underline{v}_1 < 0,$$

for all $\underline{v}_1 \neq 0$, satisfying

$$A_{11}\,\underline{v}_1 = 0.$$

(the objective function is strictly convex within the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace).

At this point it is useful to put the second-order condition for a constrained maximum, as developed in section 15.5, in the form in which this condition appears in the case of a quadratic objective function and linear side-conditions. This permits simpler proofs.

Theorem

Let
$\underline{w}$ be an n-dimensional vector of known coefficients, D a square and symmetric matrix of known coefficients, D a square and symmetric matrix of known coefficients, A an m by n matrix of known coefficients, and $\underline{b}$ an m by 1 vector of known coefficients.

Then,
regardless of the content of the vector $\underline{b}$

$$\tau(\underline{x}) = \underline{w}'\underline{x} + \tfrac{1}{2}\underline{x}'D\underline{x}$$

attains a finite maximum in the $A\underline{x} = \underline{b}$ subspace,
only if

$$\underline{v}'D\underline{v} \leq 0$$

is true for all $\underline{v}$ satisfying $A\underline{v} = 0$.

Proof

If a finite constrained maximum exists, we may express any actual vector $\underline{x}$ as

$$\underline{x} = \underline{x}^* + \underline{v} = \underline{x}^* + \Delta\underline{x} \tag{16.2.8}$$

where $\underline{x}^*$ is a vector at which the constrained maximum is attained.

Furthermore, if maximum exists, we may form the Lagrangean expression

$$L = \underline{w}'\underline{x} + \tfrac{1}{2}\underline{x}'D\underline{x} + \underline{p}'(\underline{b}-A\underline{x}) \tag{16.2.9}$$

where on account of the equation-nature of the side-restriction, $\underline{p}$ need not be non-negative. We may then (in the absence of non-negativity restrictions) require, without regard to complementary slackness

$$\frac{\partial L}{\partial \underline{x}}(\underline{x}^*) = \underline{w} + D\underline{x}^* - A'\underline{p} = 0 \tag{16.2.10}$$

We now substitute $\underline{x}^* + \underline{v}$ for $\underline{x}$ by (16.2.8) into (16.2.9), to obtain

$$\begin{aligned} L &= \underline{w}'(\underline{x}^*+\underline{v}) + \tfrac{1}{2}(\underline{x}^*+\underline{v})'D(\underline{x}^*+\underline{v}) + \underline{p}'(\underline{b}-A\underline{x}^*) - \underline{p}'A\underline{v} \\ &= \underline{w}'\underline{x}^* + \tfrac{1}{2}\underline{x}^{*\prime}D\underline{x}^* + \underline{p}'(\underline{b}-A\underline{x}^*) \\ &\quad + (\underline{w}' + \underline{x}^{*\prime}D - \underline{p}'A')\underline{v} + \tfrac{1}{2}\underline{v}'D\underline{v} \end{aligned} \tag{16.2.11}$$

However on account of the first-order-conditions (16.2.10), and the assumed equality between $A\underline{x}^*$ and $\underline{b}$ (16.2.11) reduces to

$$L = \tau(\underline{x}^*) + \tfrac{1}{2}\underline{v}'D\underline{v} \tag{16.2.12}$$

Since for any $A\underline{v} = 0$ the side-conditions remain satisfied and the value of L remains equal to τ, the result follows. q.e.d.

N.B.:
Recapitulation of the similar theorem in the case of non-linear side-conditions, as stated in section 15.5:

A constrained maximum exists, only if

$$\underline{v}'\left[\frac{\partial^2 L}{\partial \underline{x}}(\underline{x}^*)\right]\underline{v} \leq 0$$

is true for all $\underline{v}$ satisfying

$$\left[\frac{\partial f}{\partial \underline{x}}(\underline{x}^*)\right]\underline{v} = 0$$

The theorem stated here, relating to linear side-conditions is included in that theorem as a special case

$$\left[\frac{\partial^2 L}{\partial \underline{x}}\right] = \left[\frac{\partial^2 \tau}{\partial \underline{x}}\right] = D; \quad \left[\frac{\partial \underline{f}}{\partial \underline{x}}\right] = -A.$$

As in the case of convexity throughout the coordinate space the existence of a maximum implies (subspace-)convexity but (subspace-) convexity does not imply the existence of a maximum.

Example

Maximise $\tau = x_1 + x_2$

Subject to $x_1 = x_2$

The objective function is convex throughout the coordinate space, therefore also convex in the $x_1=x_2$ subspace, but there is no finite maximum.

Just as in the case of convexity throughout the coordinate space (see section 14.4) a stronger theorem in the if and only if form applies in the presence of strict convexity.

There is, however, one obvious caveat, relating to non-emptiness. We must begin by assuming that the set of vectors satisfying $A\underline{x} = \underline{b}$ is not an empty set. This point is attended to by putting additional conditions on the order and rank of A.

Theorem

Let A be an m by n matrix of rank m ($m \leq n$), while $\underline{b}$ is an m-dimensional vector, D an n by n symmetric matrix, and $\underline{w}$ an n-dimensional vector.

Then

$$\tau(\underline{x}) = \underline{w}'\underline{x} + \tfrac{1}{2}\underline{x}'D\underline{x}$$

attains a unique maximum in the $A\underline{x} = \underline{b}$ subspace if and only if

$$\underline{v}'D\underline{v} < 0$$

is true for all $\underline{v}$ satisfying $A\underline{v} = 0$, and $\underline{v} \neq 0$.

Proof

We first state and prove the following lemma

The assumed

$$\underline{v}'D\underline{v} < 0$$

for all $\underline{v} \neq 0$ satisfying $A\underline{v} = 0$
implies the non-singularity of

$$M = \left[\begin{array}{c|c} D & -A' \\ \hline A & \end{array}\right]$$

Proof (of the lemma)

Assume by contra-assumption that for some $[\underline{v}',\underline{z}'] \neq 0$, we have

$$\left.\begin{array}{ll} D\underline{v} - A'\underline{z} = 0 &) \\ &) \\ A\underline{v} = 0 &) \end{array}\right. \qquad (16.2.13)$$

For $[\underline{v}',\underline{z}'] \neq 0$, $\underline{v} = 0$, (16.2.13) implies, by its first block-row

$$A'\underline{z} = 0$$

and therefore for $m \leq n$
a contradiction of the assumed full rank of A.
We must therefore assume $\underline{v} \neq 0$.

From (16.2.13) as a whole we then obtain

$$[\underline{v}',\underline{z}'] \left[\begin{array}{c|c} D & -A' \\ \hline A & \end{array}\right] \begin{bmatrix} \underline{v} \\ \underline{z} \end{bmatrix}$$

$$= \underline{v}'D\underline{v} + \underline{z}'A\underline{v} - \underline{v}'A\underline{z} = \underline{v}'D\underline{v} = 0$$

contradicting the assumed strict subspace-convexity, in combination with the second block-row of (16.2.13).

Therefore we must assume (16.2.13) to be untrue for any $[\underline{v}',\underline{z}'] \neq 0$.
q.e.d. for the lemma.

The lemma proves that the composite system

$$\left.\begin{array}{l} D\underline{x}^* - A'\underline{p} = -\underline{w} \\ \\ A\underline{x}^* \qquad\quad = \underline{b} \end{array}\right) \qquad (16.2.14)$$

can be solved by inversion, yielding a solution to (16.2.8).

Therefore even if we do not initially assume that $\underline{x} = \underline{x}^*$ is an optimum, this is proved by (16.2.12), and in the case of strict subspace-convexity, $\underline{x} = \underline{x}^*$ is shown to be a unique constrained maximum.
q.e.d.

16.3 Outline of the Simplex Algorithm for Quadratic Programming

The algorithm discussed in this section is essentially Cottle's [6] , but features which are more akin to Van de Panne and Whinston [36] , [37] , are introduced later in this chapter. i.e. section 16.9 (flying through dual restrictions), and section 16.11 (the introduction of a separate Phase I).

A difference between the algorithm offered here and the ones suggested by these authors is that they include unit vectors in their tableaux, while the "shortened" tableau discussed in section 8.8 is used here, and indeed, even more condensed tableau-presentations will be introduced later in this chapter.

The algorithm involves the use of a tableau which is rather similar to a linear programming tableau, but the rules for choosing pivots, especially the rule for choosing the pivotal column, are different from those which apply in the case of linear programming.

In a quadratic programming tableau the objective function row is needed only for calculating the solution value, not for search operations. We "present" the quadratic programming tableau as a linear programming problem, as follows:

Maximise

$$\mu = \underline{w}' \underline{x} + \underline{b}' \underline{p} \qquad (16.3.1)$$

Subject to

$$\left.\begin{array}{l} D\underline{x} - A'\underline{p} \leq -\underline{w} \\ \qquad\qquad\quad) \quad \underline{x} \geq 0, \quad \underline{p} \geq 0 \\ A\underline{x} \qquad\quad \leq \underline{b} \end{array}\right) \qquad \begin{array}{l} (16.2.3) \\ \\ (7.2.2) \end{array}$$

It will be shown in section 16.5, that, for solutions which satisfy complementary slackness condition, the value of this pseudo objective function is twice that of the specified objective function, and we shall write 2τ instead of μ.

We write the above system in tabular form, representing an initial tableau of the trivial basis. We use the "shortened" presentation of the simplex tableau as discussed for linear programming in section 8.8, and tabulate the system as follows:

Name	$\underline{x}$	$\underline{p}$	$\leq$	Value
$\underline{d}$	D	-A'		$-\underline{w}$
$\underline{s}$	A			$\underline{b}$
2τ	$-\underline{w}'$	-b'		0

Here $\underline{d}$ are the dual slack-variables, associated with (16.2.3) and $\underline{s}$ are the primal slack-variables associated with (7.2.2).

The trivial basis is $\underline{d} = -\underline{w}$ and $\underline{s} = \underline{b}$.

The requirement of complementary slackness is represented by the name-codes of the variables. The elements of $\underline{x}$ are numbered 1 to n, the complementary dual slacks, the elements of $\underline{d}$ from -1 to -n. The elements of $\underline{s}$ are numbered from 1000 to 1000 + m, the associated dual variables from -1000 to -1000 - m.

Example

Minimise $\quad x_1 + x_2 + x_1^2 + x_2^2$

i.e. Maximise $\quad -x_1 - x_2 - x_1^2 - x_2^2$

Subject to $\quad x_1 + 2x_2 \geq 3 \qquad (\text{i.e. } -x_1 - 2x_2 \leq -3)$

$\qquad (x_1, x_2 \geq 0)$

Formulate the Lagrangean expression

$$L = -x_1 - x_2 - x_1^2 - x_2^2 + p_1 (x_1 + 2x_2 - 3)$$

and obtain the dual requirements

$$\frac{\partial L}{\partial x_1} = -1 - 2x_1 + p_1 \leq 0$$

$$\frac{\partial L}{\partial x_2} = -1 - 2x_2 + 2p_1 \leq 0$$

or in the tabular presentation with all inequalities in the $\leq$ form:

TABLEAU 16.3 A

THE SET-UP TABLEAU OF A QUADRATIC PROGRAMMING PROBLEM.

NAME	CODE	X 1	X 2	P 1	VALUE
	CODE	1	2	-1001	
D 1	-1	-2	-	1	1
D 2	-2	-	-2	2	1
S 1	1001	-1	-2	-	-3
2T		1	1	3	-

The complementary slackness condition is now given as not permitting two names of the same absolute value to be in the list of basic variables at the same time.

If x_1 (name-code 1) enters the basis, then d_1 (name-code-1), being the slack in the dual restriction which refers to x_1, has to leave. Likewise p_1 (name-code -1001) is not allowed in the list of basic variables, unless s_1, the slack variable associated with the corresponding primal restriction (name-code 1001) leaves the list of basic variables.

The methods for actually finding a solution permits some tableaux to be "not in standard form", in which case one name is in the list of basic variables together with its complementary name. (The complementary slackness condition is then temporarily violated.)

The algorithm begins with selecting a "badname", a variable which we wish to eliminate from the list of basic variables. (Or to use Cottle's terminology, a distinguished variable.) We initially assume that badname-variable selection is according to the rule of the steepest ascent, i.e. we select the name of the variable for which the most negative entry occurs on the right-hand side. In our example this would be badname = 1001,

i.e. we seek to eliminate the (negative valued) slack of the first primal restriction. The elimination of the badname-variable is anticipated, by introducing the complementary name in the list of basic variables. Hoping to eliminate s_1, we introduce the dual variable p_1. The complementary variable associated with the selected badname variable, is known as the driving variable.

The choice of the pivotal row is more or less by the same method as in linear programming, i.e. by the rule of the smallest quotient. However, there are differences with that rule, as known from LP. We accept a negative pivot in the badname-variable row, but not in any other row.

TABLEAU 16.3 B

SET-UP TABLEAU OF A QUADRATIC PROGRAMMING ROBLEM, WITH PIVOT-MARKING.

NAME	!		!!	X 1	X 2	P 1	!!	VALUE
	!	CODE	!!	1	2	-1001	!!	
D 1	!	-1	!!	-2	-	1	!!	1
D 2	!	-2	!!	-	-2	②	!!	1
S 1	!	1001	!!	-1	-2	-	!!	-3
2T	!		!!	1	1	3	!!	-

In case of equal quotients, the badname-row has priority over another row in which an equal quotient might occur. Also, quite apart from degeneracy, there are restrictions on the acceptability of dual variables as leaving variables; sometimes they are allowed to become negative instead. We shall need to come back to this question, but at this stage, we will illustrate the algorithm, on the assumption, that the rule of the smallest quotient is applied in the usual way. We find d_2 as leaving variable, and develop tableau 16.3c.

TABLEAU 16.3 C

QP TABLEAU, AFTER ONE STEP, NON-STANDARD FORM, WITH NEW PIVOT-MARKING.

NAME	!		!!	X 1	X 2	D 2	!!	VALUE
	!	CODE	!!	1	2	-2	!!	
D 1	!	-1	!!	-2	①	-0.50	!!	0.50
P 1	!	-1001	!!	-	-1	0.50	!!	0.50
S 1	!	1001	!!	-1	-2	-	!!	-3
2T	!		!!	1	4	-1.50	!!	-1.50

Tableau 16.3c is not in standard form, i.e. fails to satisfy the complementary slackness requirements. Since d_1 (name-code -1) is a basic variable, x_1 is excluded as new pivot column, or we would risk a second pair of alternative variables.

Further violation of the complementary slackness requirement needs to be avoided, and we introduce the complementary name to the just eliminated d_2 variable. (Re-introduction of d_2 as basic variable would merely reverse the previous step.) We would wish to get back to standard form.

Since d_2 (name-code -2) was just eliminated, x_2 (name-code 2) is the new variable to enter the list of basic variables. The new pivot-row is the d_1 row, (by the rule of the smallest quotient). (That pivot was already marked in tableau 16.3c. The new tableau is:

TABLEAU 16.3 D

THE SAME QP PROBLEM, AFTER TWO STEPS, STILL NOT BACK IN STANDARD FORM, WITH NEW PIVOT MARKED.

NAME		X 1	D 1	D 2	VALUE
	CODE	1	-1	-2	
X 2	2	-2	1	-0.50	0.50
P 1	-1001	-2	1	-	1
S 1	1001	(-5)	2	-1	-2
2T		9	-4	0.50	-3.50

Note that we have two distinct rules for selecting the incoming variable. In standard form tableaux we select the driving variable, otherwise column-selection is by the complementarity rule which says that the incoming variable is the complementary variable associated with the just-eliminated variable.

In tableau 16.3d the complementary variable to the just-eliminated d_1 variable is x_1. The x_1 column indicates 3 ratios, of which both $\frac{1}{2}/-2$ in the x_2 row, and $1/-2$ in the p_1 row are of the wrong sign. Therefore, $-2/-5$ in the s_1 row indicates the pivot-row. Note the acceptance of a negative pivot, in the badname row. We refuse negative pivots in other rows, "flying through" violated restrictions instead.

The s_1,x_1 element has been marked in tableau 16.3d. The step is made and the next tableau is tableau 16.3e.

The tableau is now back in standard form, and the right-hand side is non-negative. Therefore $x_1 = 2/3$, $x_2 = 1\ 3/10$ is the optimum.

TABLEAU 16.3 E

THE SAME QP PROBLEM, AFTER THREE STEPS, NOW BACK IN STANDARD FORM, THIS ALSO BEING THE OPTIMUM.

NAME	!		!!	S 1	D 1	D 2	!!	VALUE
	!	CODE	!!	1001	-1	-2	!!	
X 2	!	2	!!	-0.40	0.20	-0.10	!!	1.30
P 1	!	-1001	!!	-0.40	0.20	0.40	!!	1.80
X 1	!	1	!!	-0.20	-0.40	0.20	!!	0.40
2T	!		!!	1.80	-0.40	-1.30	!!	-7.10

Exercise 16.3a

Consider the quadratic programming problem

Maximise $10x_1 + 10x_2 - x_1^2 - x_2^2$ (= minimise $(x_1-5)^2 + (x_2-5)^2-50$)

Subject to $2x_1 + x_2 \leq 6$

$(x_1, x_2 \geq 0)$

Solve the problem by applying the algorithm, as outlined so far.

To verify the correctness of your results, also map the one restriction and draw the $(x_1-5)^2 + (x_2-5)^2 = 16.2$ circle, $(\sqrt{16.2} \simeq 4.02)$, which is the optimal iso-objective function line.

Exercise 16.3b

If you had any difficulty in making the actual steps for this qp problem, (as distinct from sorting out which steps to make) it's time you refresh your competence at tableau-manipulation.

You should:
1) For each tableau developed in this section (which are all shortened tableaux), write the corresponding full explicit tableau, with the unit vectors.
2) For tableaux 16.3a, 16.3c and 16.3d, write the corresponding systems of equations explicitly.

16.4 The Symmetry Property of the Tableau in its Standard Form

Some properties of the tableau are more clearly illustrated if we re-order the tableau according to the name-codes of the various columns and rows.

We put each column in the position indicated by a name of the same absolute value in the trivial basis, and re-order the rows on the same principle (see tableau 16.4a).

TABLEAU 16.4 A

THE OPTIMUM TABLEAU OF THE QP PROBLEM FROM SECTION 16.3, RE-ORDERED ACCORDING TO CODES.

NAME	!	!!	D 1	D 2	S 1	!!	VALUE
	! CODE	!!	-1	-2	1001	!!	
X 1	! 1	!!	-0.40	0.20	-0.20	!!	0.40
X 2	! 2	!!	0.20	-0.10	-0.40	!!	1.30
P 1	!-1001	!!	0.20	0.40	-0.40	!!	1.80
2T	!	!!	-0.40	-1.30	1.80	!!	-7.10

Note the semi-symmetry property. If the signs were suppressed, the tableau would be a symmetric matrix.

To describe the signs, it is useful to classify all the names in the tableau in two groups, primal names and dual names. Primal names are the positive names corresponding to the elements of $\underline{x}$ and $\underline{s}$ with positive name-codes, as well as the value-column.

Dual names are the ones corresponding to elements of $\underline{p}$ and $\underline{d}$, with negative name-codes, as well as the quasi objective function.

It appears that a cell of the tableau, where the row and the column belong to the same class, has the same absolute value but the opposite sign as the symmetric entry across the diagonal. If the row- and column-names belong to different classes, there is full symmetry.

The bottom left-hand block with dual row-names $\underline{p}$ and μ and dual column-names d_1 and d_2 is symmetric but opposite in sign to the top right-hand block with primal row-names x_1 and x_2, the primal column-names, s_1 and the value column.

The rest of the tableau (in this small example only the diagonal itself) is fully symmetric. The rule applies in that trivial case as well. Thus the rowname of the top left-hand cell is 1, the corresponding column-name is -1, and the cell is its own full symmetry.

This symmetry property is systematic as we first show for the (quasi) target-row and the value-column.

Consider the dual of the linear problem formulated in Section 16.3. This dual problem is:

Maximise

$$\lambda = \underline{w}' \underline{y} - \underline{b}' \underline{z} \qquad (16.4.1)$$

Subject to

$$-D\underline{y} - A' \underline{z} \leq -\underline{w} \qquad (16.4.2)$$

and

$$A\underline{y} \geq -\underline{b} \qquad (16.4.3)$$

if $\underline{x}$, $\underline{p}$ is a solution to the primal problem, then

$$\underline{y} = -\underline{x} \qquad (16.4.4)$$

and

$$\underline{z} = \underline{p} \qquad (16.4.5)$$

will be a solution of the dual problem, as may be seen from comparing (16.4.2) with (16.2.3) and (16.4.3) with (7.2.2). (That "dual" solution will not usually be a non-negative vector, but this does not concern us here).

We may not generally require that the target-row indicates the same combination of binding restrictions for the dual, as does the value column for the primal.

But the complementary slackness condition, i.e. the requirement of standard form, implies that in the case at hand. Therefore we will find $\underline{y}$ as defined by (16.4.4) and $\underline{z}$ as defined by (16.4.5) in the target-row, when ever the tableau is in standard form.

For elements of the pivot-inverse, we also have a more direct proof.

Consider the symmetric matrix

$$S = \left[\begin{array}{c|c} D_{11} & A'_{11} \\ \hline A_{11} & \end{array}\right] \qquad (16.4.6)$$

which is the block-pivot with the sign of the second block-column changed round.

The inverse of this matrix is obviously an entirely symmetric matrix, and we find the inverse of the block-pivot by changing the sign of the bottom block-row once more.

Denote

$$C = S^{-1} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

with associated name-codes

$$\begin{array}{c} \\ \underline{x}_1 \\ \underline{p}_1 \end{array} \begin{array}{c} \begin{array}{cc} \underline{d}_1 & \underline{s}_1 \end{array} \\ \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \end{array}$$

and the correct inverse of the block-pivot is

$$\begin{array}{c} \\ x_1 \\ p_1 \end{array} \begin{array}{c} \begin{array}{cc} \underline{d}_1 & \underline{s}_1 \end{array} \\ \begin{bmatrix} C_{11} & C_{12} \\ -C_{21} & -C_{22} \end{bmatrix} \end{array}$$

The proof can in fact be generalized to the full tableau. The semi-symmetric matrix associated with the set-up tableau would in that case be an extended basis-matrix, the difference with the usual basis-matrix being the addition of two - I block-rows representing the non-negatively restrictions. When these restrictions are explicitly written as $\underline{x} \geq o$ and $\underline{p} \geq o$, these vectors match the unit vectors which represent dual and primal slack variables.

The curious sign-inversion rules now follow and may be recapitulated as follows.

If the row and column name which are associated with a tableau-cell belong to the same class e.g. p_i/d_j in $-C_{21}$ with an associated cell on the other side with x_j/s_i in C_{12} we have sign-inversion.

Inside the diagonal blocks we have cells for which the row and column-names belong to different classes e.g. x_j/d_j in C_{11} and p_i/s_i in $-C_{22}$ and we have full symmetry.

Exercise

The optimum tableau which you should have obtained when doing the exercise at the end of the previous section 16.3 is given here in part i.e. the top right-hand part only.

TABLEAU 16.4 EX

THE OPTIMUM TABLEAU OF THE
QP EXERCISE FROM SECTION 16.3,
FOR YOU TO COMPLETE.

NAME		!!	D 1	D 2	S 1	!!	VALUE
	CODE	!!	-1	-2	1001	!!	
X 1	1	!!	-0.10	0.20	0.40	!!	1.40
X 2	2	!!	??	-0.40	0.20	!!	3.20
P 1	-1001	!!	??	??	-0.40	!!	3.60
2T		!!	??	??	??	!!	67.60

Find the correct entries to be placed in the bottom lefthand cells marked as ??

Do not initially refer back to section 16.3, using the symmetry properties explained in this section instead.

Then afterwards compare with section 16.3 and attend to any corrections which may be needed in your answers to either of the exercises.

16.5 The Solution Value

Recall, from section 16.3, the linear quasi-objective function

$$\mu = \underline{w}' \underline{x} + \underline{b}' \underline{p} \qquad (16.3.1)$$

We will now show that, for a solution which is described by a standard form tableau, this function indeed is twice the value of the objective function.

We introduce equation-type equivalents of (7.2.2) and (16.2.3).

Following what we already did for individual variables, we indicate the vector of primal slack variables as $\underline{s}$, and the dual slack-variables as $\underline{d}$. Then

$$A\underline{x} + \underline{s} = \underline{b} \qquad (16.5.1)$$

is equivalent to (7.2.2), and

$$D\underline{x} - A'\underline{p} + \underline{d} = -\underline{w} \qquad (16.5.2)$$

is equivalent to (16.2.3). Substitute $A\underline{x} + \underline{s}$ for $\underline{b}$ into (16.3.1), to obtain:

$$\mu = \underline{w}'\ \underline{x} + \underline{x}'\ A'\underline{p} + \underline{s}'\ \underline{p} \qquad (16.5.3)$$

The complementary slackness condition requires the third term of (16.5.3) to vanish. Substitute $Dx + d + \underline{w}$ for $A'\underline{p}$ by (16.5.2) into (16.5.3), and suppress the last term of (16.5.3), to obtain

$$\mu = \underline{w}'\ \underline{x} + \underline{x}'D\underline{x} + \underline{x}'\underline{d} + \underline{x}'\underline{w} \qquad (16.5.4)$$

The complementary slackness condition requires the vanishment of yet another term, $\underline{x}'\underline{d}$. Therefore when the tableau is in standard form

$$\mu = \underline{w}'\ \underline{x} + \underline{x}'D\underline{x} + \underline{x}'\underline{w} = 2\ \underline{w}'\ \underline{x} + \underline{x}'D\underline{x} \qquad (16.5.5)$$

Comparing (16.5.5) with (16.1.1), it is clear that (when the tableau is in standard form) $\mu = 2\tau$ is indeed correct.

Note, however that for a non-standard form tableau,

$$\mu = 2\tau + \underline{x}'\underline{d} + \underline{p}'\underline{s} \qquad (16.5.6)$$

is the true relation.

16.6 The negative diagonal

In preparation for the next section on non-standard form tableaux, we develop some theorems, concerning the pivot inverse and its main diagonal.

These theorems and their proofs are rather detailed and a summary of the main results is given here first, together with some earlier results, but in advance of more detailed discussion.

Our interest in this section concentrates on the d_i/x_i, x_i/d_i, s_i/p_i, and p_i/s_i cells of the tableau. We assume that the tableau is ordered according to the (absolute values of the) name codes and these cells therefore all occur on the main diagonal of the tableau.

We collectively refer to them as "the diagonal cells". Note, however, that no properties concerning the 2τ/value cell is stated in this section, even where that cell too may be written on the main diagonal.

The following properties apply

Property 1

If the problem is properly convex ($\underline{v}'D\underline{v} \leq 0$ for all $\underline{v}$),

all conceivable pivot inverses $\begin{bmatrix} D_{11} & -A_{11}' \\ A_{11} & \end{bmatrix}^{-1}$

represent constrained maxima in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace and all diagonal cells are non-positive, and the fully symmetric part of the tableau forms a negative semi-definite matrix.

Property 2

Regardless of the definiteness or indefiniteness of D, standard form tableaux developed by the algorithm, describe $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspaces, in which the more restrictive condition of strict convexity ($\underline{v}_1'\,D_{11}\,\underline{v}_1 < 0$ for all $\underline{v}_1$ satisfying $A_{11}\,\underline{v}_1 = 0$ and $\underline{v}_1 \neq 0$) is satisfied.

(A subspace is the set of points satisfying some listed restrictions, see section 6.4).

In such tableaux the diagonal cells in dual variables' columns (x_i/d_i and s_i/p_i cells) are negative non-zero, except in those dual variables' columns in which all primal variables' row-entries are zero.

We may in fact interpret a positive diagonal entry as a sympton of non-convexity, and we say that non-convexity, if present, becomes manifest only in primal variables' columns.

An essential feature of the algorithm is therefore the restriction of solved solution-vectors to those where the problem may be described as being strictly convex in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace. This restriction is not nearly as severe as the word convexity might suggest.

In particular, if the objective function is anti-convex, we know (see section 14.7), that the optimum (if one exists, will be in a corner of the feasible space area.

In that case, the set of vectors $\underline{v}_1$, which satisfy $A_{11}\,\underline{v}_1 = 0$ for $\underline{v}_1 \neq 0$, is an empty set, A_{11} being square and non-singular.

Thus, if the problem is anti-convex, (the matrix D is positive semidefinite), the algorithm only develops solutions for which A_{11} is square and non-singular.

The other obvious case where we would expect this property to be revealed is with a directionally convex objective function.

For example if we seek to maximise $\tau = -x_1 + x_1 \cdot x_2$, subject to $x_1 + x_2 \geq 2$ $(x_1, x_2 \geq 0)$, we will exchange x_2 and the one dual variable p_1 against the one side-restriction and the dual restriction on x_2.

We then operate in the $x_1 + x_2 = 1$ subspace and substitution of $2-x_1$ for x_2 now gives $\tau = x_1 - x_1^2$, which is a properly and strictly convex function of x_1.

We now proceed with a more detailed discussion of the theorems.

The pivot-matrix of a quadratic programming tableau in standard form will be

$$P = \left[\begin{array}{c|c} D_{11} & -A_{11}' \\ \hline A_{11} & \end{array}\right] \qquad (16.6.1)$$

Above, D_{11} is the block of D which corresponds to the basic primal variables $\underline{x}_1$ and the associated dual slacks $\underline{d}_1$. This block will be square (of order n_1 by n_1), and symmetric, it may, or may not be negative (semi) definite, like the full matrix D. A_{11} is the intersection between the block-row A_1', the rows of which represent binding restrictions, and the block-column A_1, the columns of which represent basic variables.

The block A_{11} need not be square, it can contain more columns than rows.

Theorem

Let a vertex correspond to optimal and feasible solution to the sub-problem.

Maximise

$$\tau_1(\underline{x}_1) = \underline{w}_1' \underline{x}_1 + \tfrac{1}{2} \underline{x}' D_{11} \underline{x}_1 \qquad (16.6.2)$$

Subject to

$$A_{11} \underline{x}_1 = \underline{b}_1 \qquad (16.6.3)$$

then, irrespective of the definiteness of D_{11}

the first n_1 leading diagonal elements of the pivot inverse will be non-positive.

Proof

Consider any e.g. the j^{th} column of the pivot-inverse. We will denote the pivot inverse as Z and the j^{th} leading column of the inverse is indicated as $\underline{z}_j$. By the definition of an inverse, post-multiplication of P by $\underline{z}_j$ yield a unit vector.

$$P\ \underline{z}_j = \underline{e}_j \tag{16.6.4}$$

Therefore the quadratic form $\underline{z}_1'\ P\ \underline{z}_1$ is evaluated as

$$\underline{z}_j'\ P\ \underline{z}_j = z_{j,j} \tag{16.6.5}$$

However, for $j \leq n_1$, $\underline{z}_j$ yields a zero product with the bottom block-row of the pivot-matrix.

$$\left[A_{11}\middle|\ \right]\ \underline{z}_j = 0 \tag{16.6.6}$$

correspondingly, for $j \leq n_1$, (16.6.5) reduces to

$$\underline{z}_{1,j}\ D_{11}\ \underline{z}_{1,j} = z_{j,j} \tag{16.6.7}$$

where $\underline{z}_{1,j}$ indicates the leading sub-vector of $\underline{z}_j$, containing the n_1 elements which refer to the elements of $\underline{x}_1$ rather than $\underline{p}_1$.

Therefore, $z_{jj} > 0$ would contradict the second-order conditions for a constrained maximum, as discussed in section 15.5 and recapitulated in section 16.2.
q.e.d.

The one point still to be noted here, is that for certain columns $\underline{z}_{1,j}$ may be a zero vector, i.e. certain dual variables may not affect $\underline{x}$ at all.

Otherwise, for $\underline{z}_{1,j} \neq 0$, a top left-hand diagonal element of the pivot inverse is expressed as a quadratic form by (16.6.5) and a unique maximum is shown to imply

$$z_{jj} < 0 \tag{16.6.8}$$

We now consider the question how we may recognise standard form tableaux which correspond to solutions that are properly and strictly convex in the $A_{1,1}\ \underline{x}_1 = \underline{b}_1$ sub-space.

We postulate the existence of a symmetric ordering and partitioning of the pivot matrix P, as follows:

$$P = \left[\begin{array}{c|c|c} P_{1,1} & P_{1,2} & P_{1,3} \\ \hline P_{2,1} & P_{2,2} & P_{2,3} \\ \hline P_{3,1} & P_{3,2} & \end{array}\right] = \left[\begin{array}{c|c|c} D^*_{1,1} & D^*_{1,2} & -A^{*\prime}_{1,1} \\ \hline D^*_{2,1} & D^*_{2,2} & -A^{*\prime}_{1,2} \\ \hline A^*_{1,1} & A^*_{1,2} & \end{array}\right] \qquad (16.6.9)$$

for which all the diagonal blocks are square, while $P_{3,1} = A^*_{1,1}$ is square and non-singular.

This partitioning is related to the previous given by (16.6.1), as:

$$\left[P_{3,1} \quad P_{3,2}\right] = A_{1,1} \; ; \quad \begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{bmatrix} = D_{1,1} \; .$$

In the trivial case where no primal restriction is binding and P is identical to $D_{1,1}$, the third-block-row and block-column of P do not exist at all, but otherwise the existence of a square and non-singular block $A^*_{1,1}$ is a condition for the invertability of P. (see Section 5.7).

We are, incidentally, also in a position at this point to make an inference concerning the number of non-zero variables in a QP problem.

We know from section 5.7, that (if P is invertable) the rank of $D_{1,1}$ must be at least equal to the number of columns of $A^*_{1,2}$. Or, to put the same statement the other way round, the number of non-zero variables in excess of the number of binding restrictions is at most equal to the rank of $D_{1,1}$.

The following tableau-extract from the partitioned set-up tableau relates some notation of vectors to the partitioning given by (16.6.9.)

Name	$\underline{x}^*_1$	$\underline{x}^*_2$	$\underline{p}_1$	Value
$\underline{d}^*_1$	$D^*_{1,1}$	$D^*_{1,2}$	$-A^{*\prime}_{1,1}$	$-\underline{w}^*_1$
$\underline{d}^*_2$	$D^*_{2,1}$	$D^*_{2,2}$	$-A^{*\prime}_{1,2}$	$-\underline{w}^*_2$
$\underline{s}_1$	$A^*_{1,1}$	$A^*_{1,2}$		$\underline{b}_1$
2τ	$-\underline{w}^{*\prime}_1$	$-\underline{w}^{*\prime}_2$	$-\underline{b}'_2$	

A corresponding extract from the current tableau, named T, is summarized as follows:

Name	$\underline{d}_1^*$	$\underline{d}_2^*$	$\underline{s}_1$
$\underline{x}_1^*$	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$
$\underline{x}_2^*$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$
$\underline{p}_1$	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$

Theorem

A necessary and sufficient condition for P to be associated with a solution where τ is strictly convex in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace, is that (for some ordering of the variables) $T_{2,2}$ is negative definite, (or else to be of zero order).

Proof:

Consider the product of the two top block-rows of T, post-multiplied by the first two block-columns of p, postmultiplied by a vector $\Delta\underline{x}_1^*$, $\Delta\underline{x}_2^*$ (a change in the vector $\underline{x}$).

$$\begin{bmatrix} T_{1,1} & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} & T_{2,3} \end{bmatrix} \begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \\ P_{3,1} & P_{3,2} \end{bmatrix} \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} = \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} \tag{16.6.10}$$

(The two matrices on the left of the left-hand side expression are a block-row of P^{-1}, postmultiplied by a block-column of P, yielding a unit-matrix as product.)

Within the $A_{1,1}\,\underline{x}_1 = \underline{b}_1$ subspace, $\Delta\underline{x}_1$ is restricted to $A_{1,1}\,\Delta\underline{x}_1 = 0$, and the following relation applies, for vectors $\Delta\underline{x}_1$, which obey that side-restriction.

$$\begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \\ P_{3,1} & P_{3,2} \end{bmatrix} \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} = \begin{bmatrix} D_{1,1}^* & D_{1,2}^* \\ D_{2,1}^* & D_{2,2}^* \\ A_{1,1}^* & A_{1,2}^* \end{bmatrix} \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} = \begin{bmatrix} \underline{z}_1^* \\ \underline{z}_2^* \\ 0 \end{bmatrix} \tag{16.6.11}$$

In (16.6.11), $\underline{z}_1^*$ and $\underline{z}_2^*$ may be considered as arbitrary vectors. The abstract form T indicates that for $\Delta\underline{s}_1=0$ (and $\Delta\underline{p}_2=0$), we may interpret $\underline{z}_1^*$ and $\underline{z}_2^*$ as $-\underline{d}_1^*$ and $-\underline{d}_2^*$. We therefore reformulate the combination of (16.6.10) and (16.6.11) as:

$$\begin{bmatrix} T_{1,1} & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} & T_{2,3} \end{bmatrix} \begin{bmatrix} -\Delta\underline{d}_1^* \\ -\Delta\underline{d}_1^* \\ 0 \end{bmatrix} = \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} \qquad (16.6.12)$$

Note that the condition $\Delta\underline{s}_1 = 0$ in (16.6.12) implies that $\Delta\underline{x}_1^*$ and $\Delta\underline{x}_2^*$ obey the requirement $A_{1,1}\ \Delta\underline{x}_1 = 0$.

Note also that the block $\begin{bmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2} \end{bmatrix}$

is fully symmetric by the symmetry rules.

Therefore, for vectors $|\Delta\underline{x}_1^{*\prime}\ \ \Delta\underline{x}_2^*|$ which satisfy the $A_{1,1}\ \Delta\underline{x}_1 = 0$ condition, we find the following relation:

$$\begin{aligned} \begin{bmatrix} \Delta\underline{d}_1^{*\prime} & \Delta\underline{d}_2^{*\prime} \end{bmatrix} \begin{bmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2} \end{bmatrix} \begin{bmatrix} \Delta\underline{d}_1^* \\ \Delta\underline{d}_2^* \end{bmatrix} &= -\begin{bmatrix} \Delta\underline{d}_1^{*\prime} & \Delta\underline{d}_2^{*\prime} \end{bmatrix} \begin{bmatrix} \Delta\underline{x}_1^* \\ \Delta\underline{x}_2^* \end{bmatrix} \\ &= -\ \Delta\underline{d}_1^{*\prime}\ \Delta\underline{x}_1^* - \Delta\underline{d}_2^{*\prime}\ \Delta\underline{x}_2^* \end{aligned} \qquad (16.6.13)$$

However, $\Delta\underline{d}_1^*$, $\Delta\underline{d}_2^*$, $\Delta\underline{x}_1^*$, $\Delta\underline{x}_2^*$ are also related by a tableau associated with a smaller block-pivot. This smaller block-pivot is

$$P^* = \left[\begin{array}{c|c} D_{1,1}^* & -A_{1,1}^{*\prime} \\ \hline A_{1,1}^* & \end{array}\right] \qquad (16.6.14)$$

The tableau-extract corresponding to that block-pivot P* may be summarised as T* =

$$\begin{array}{c} \\ \underline{x}_1^* \\ \underline{d}_2^* \\ \underline{p}_1 \end{array} \begin{array}{c} \begin{array}{ccc} \underline{d}_1^* & \underline{x}_2^* & \underline{s}_1 \end{array} \\ \left[\begin{array}{c|c|c} & -T_{2,1}^{*\prime} & A_{1,1}^{*-1} \\ \hline T_{2,1}^* & T_{2,2}^* & T_{3,2}^{*\prime} \\ \hline -[A_{1,1}^{*\prime}]^{-1} & T_{3,2}^* & T_{3,3}^* \end{array}\right] \end{array} \quad \begin{array}{l} \text{where } -T_{2,1}^{*\prime} = T_{1,2}^* \\ \text{and} \quad T_{3,2}^{*\prime} = T_{2,3}^* \\ \text{conform the symmetry rules.} \end{array}$$

Note the zero top left-hand block, due to the block-triangular structure of P*

Tableaux T and T* are related by, among other relations,

$$T_{2,2} = T^{*-1}_{2,2} \tag{16.6.15}$$

($T^*_{2,2}$ is the third and last block-pivot used in inverting P to become T, T* being the preceding calculation tableau.)

We also note that both $T^*_{2,2}$ and $T_{2,2}$ are non-singular (if P is non-singular).

The first block-row of T* shows that the vectors $\Delta\underline{d}^*_1$, $\Delta\underline{d}^*_2$, $\Delta\underline{x}^*_1$ and $\Delta\underline{x}^*_1$ and Δx^*_2, as given in (16.6.12) are also related by

$$\Delta\underline{x}^*_1 = T^{*\prime}_{2,1}\ \Delta\underline{x}^*_2 \tag{16.6.16}$$

(again assuming $\Delta\underline{s}_1 = 0$)

Similarly for the second block-row

$$\Delta\underline{d}^*_2 = -T^*_{2,1}\ \Delta\underline{d}^*_1 - T^*_{2,2}\ \Delta\underline{x}^*_2 \tag{16.6.17}$$

Substitution of the right-hand sides of (16.6.16) and (16.6.17) for $\Delta\underline{x}^*_1$ and $\Delta\underline{d}^*_2$ into the right-hand side of (16.6.13) now yields

$$\left[\Delta\underline{d}^{*\prime}_1,\ \Delta\underline{d}^{*\prime}_2\right] \begin{bmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2} \end{bmatrix} \begin{bmatrix} \Delta\underline{d}^*_1 \\ \Delta\underline{d}^*_2 \end{bmatrix} =$$

$$- \Delta\underline{d}^{*\prime}_1\ T^{*\prime}_{2,1}\ \Delta\underline{x}^*_2 + (\Delta\underline{d}^{*\prime}_1\ T^{*\prime}_{2,1} + \Delta\underline{x}^{*\prime}_2\ T^{*\prime}_{2,2})\ \Delta\underline{x}^*_2$$

$$= \Delta\underline{d}^{*\prime}_1\ \Delta\underline{x}^{*\prime}_1 \begin{bmatrix} & -T^{*\prime}_{2,1} \\ T^*_{2,1} & T^*_{2,2} \end{bmatrix} \begin{bmatrix} \Delta\underline{d}^*_1 \\ \Delta\underline{x}^*_2 \end{bmatrix}$$

$$= \Delta\underline{x}^{*\prime}_2\ T^*_{2,2}\ \Delta\underline{x}^*_2 \tag{16.6.18}$$

Combining (16.6.12), (16.6.13) and (16.6.18), we find for every $\Delta\underline{x}_1' = \left[\Delta\underline{x}^{*\prime}_1,\ \Delta\underline{x}^{*\prime}_2\right]$, which satisfies the side-condition $A_{1,1}\ \Delta\underline{x}_1 = 0$,

$$\begin{bmatrix}\Delta\underline{x}_1^{*\prime} , \Delta\underline{x}_2^{*\prime}\end{bmatrix} \begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2}\end{bmatrix} \begin{bmatrix}\Delta\underline{x}_1^{*} \\ \Delta\underline{x}_2^{*}\end{bmatrix}$$

$$= \begin{bmatrix}\Delta\underline{d}_1^{*\prime} \Delta\underline{d}_2^{*}\end{bmatrix} \begin{bmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2}\end{bmatrix} \begin{bmatrix}\Delta\underline{d}_1^{*} \\ \Delta\underline{d}_2^{*}\end{bmatrix} = \Delta\underline{x}_2^{*\prime}\ T_{2,2}^{*}\ \Delta\underline{x}_2^{*} \qquad (16.6.19)$$

which shows that the negative definiteness of $T_{2,2}^{*}$ is a necessary condition for the far left-hand side expression to be negative for all $\Delta\underline{x}_2^{*} \neq 0$.

Furthermore, by (16.6.16) $\Delta\underline{x}_2^{*} = 0$ implies (for $A_{1,1}\ \Delta\underline{x}_1 = 0$) $\Delta\underline{x}_1^{*} = 0$, therefore $\Delta\underline{x}^{*}{}_1 \neq 0$ implies (for $A_{1,1}\ \Delta\underline{x}_1 \neq 0$) $\Delta\underline{x}_2^{*} \neq 0$.

Since $T_{2,2}^{*}$ is non-singular we must also exclude the possibility that both sides of (16.6.19) are zero on account of semi-definiteness of $T_{2,2}^{*}$.

The right-hand side is negative non-zero for all $\Delta\underline{x}_2^{*} \neq 0$ and there are no vectors $\Delta\underline{x}_1 \neq 0$ with $\Delta\underline{x}_2^{*} = 0$ and $A_{1,1}\ \Delta\underline{x}_1 = 0$. Therefore all the three equivalent expressions in (16.6.19) are negative non-zero for all $\Delta\underline{x}_1 \neq 0$ satisfying $A_{1,1}\ \Delta\underline{x}_1 = 0$.

Since $T_{2,2}^{*}$ is the inverse of $T2,2$ its negative definiteness is equivalent to the negative definiteness of $T_{2,2}$ which is therefore also shown, proving the necessity of the stated condition.

We also note, that for a vector $\Delta\underline{x}_1 \neq 0$ satisfying the side-condition $A_{11}\ \Delta\underline{x}_1 = 0$, the quadratic form $\Delta\underline{x}_1'\ D_{11}\ \Delta\underline{x}_1$ is evaluated by (16.6.19) as $\Delta\underline{x}_2^{*\prime}\ T_{2,2}^{*}\ \Delta\underline{x}_2^{*} < 0$, proving the sufficiency of the stated condition.
q.e.d.

Note that this theorem does not have the usual weaker semi-definite equivalent. The invertability of $T_{2,2}^{*}$ means that standard-form tableaux either correspond to solutions where τ is strictly convex in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, or else not convex at all!
($\Delta\underline{x}_2^{*\prime}\ T_{2,2}^{*}\ \Delta\underline{x}_2^{*} \leq 0$, all $\Delta\underline{x}_2'$; $\Delta\underline{x}_2^{*\prime}\ T_{2,2}^{*}\ \Delta\underline{x}_2^{*} = 0$, some $\Delta\underline{x}_2^{*} \neq 0$ implies $|T_{2,2}^{*}| = 0$, and therefore $|P| = 0$.)

It is, incidentally, quite possible to write standard form tableaux which correspond to solutions of both the primal and the dual restrictions, but which do not refer to convex subspaces, and which display manifest non-convexity in dual variables' columns, i.e. a positive diagonal cell.

Example

Maximise $\tau = -x_1 - x_2 + x_1 \cdot x_2$

subject to $x_1 + x_2 \leq 10$

TABLEAU 16.6 A

ILLUSTRATION OF NOT RESPECTING (SUBSPACE) CONVEXITY.

SET-UP TABLEAU

NAME	X1	X2	P1	VALUE
D1	-	1	-1	1
D2	1	-	-1	1
S1	1	1	-	10
2T	1	1	-10	-

INCORRECT 'OPTIMUM'

NAME	D1	D2	P1	VALUE
X1	-	1	-1	1
X2	1	-	-1	1
S1	-1	-1	2	8
2T	-1	-1	-8	-2

(To solve this non-convex problem, one would have to add restrictions, e.g. $x_1 \geq 2$, $x_2 \geq 2$. The objective function would then be directionally convex in the feasible space area.)

The property that non-convexity is manifest only in primal variables' columns is true for standard form tableaux developed by the algorithm. These tableaux correspond to solutions which are convex, and therefore strictly convex in the $A_{1,1}\,\underline{x}_1 = \underline{b}_1$ subspace.

One further comment on the precise significance of the theorem which we might call the "negative definiteness theorem", concerns the partitioning. It is normal that several partitionings will permit a square and non-singular block $A^*_{1,1}$. Correspondingly, there will be several negative-definite blocks $T_{2,2} = T^{*-1}_{2,2}$.

We may not, however, require that the whole top left-hand block of the tableau is negative definite, only negative semi-definiteness applies for the full intersection between primal variables' rows and dual variables' columns. And there may be zeros.

Theorem

If a standard form tableau refers to a solution which is convex in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, all dual variables' columns display one of the following characteristics:

Either: a) the diagonal cell associated with that column is negative non-zero

or b) all primal variables' row-entries in such a dual variables' column are zero.

Proof:

By the previous theorem, the assumed convexity in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, implies, in combination with the non-singularity of p, strict convexity (in the same subspace).

We now first consider dual slack variables d_j. If there are columns for these variables in the tableau the diagonal cell which we refer to is an element of the pivot inverse. Conform the notation earlier in this section we indicate the vector of primal variables' entries of the d_j column, as far as they refer to elements x_j rather than slack variables as $\underline{z}_{1,j}$.

The property of strict convexity, as proved above, now implies

$$\underline{z}'_{1,j}\ D_{1,1}\ \underline{z}_{1,j} = z_{j,j} < 0, \text{ for all } \underline{z}_{1,j} \neq 0 \qquad (16.6.20)$$

No variation in $\underline{x}_1$ implies no change in $\underline{s}$, the vector of slack-variables, therefore $\underline{z}_{1,j} = 0$ implies zero entries for all primal variables' rows in the column in question.

Concerning columns for non-basic dual variables p_i, we note that these are equivalent to dual slack-variables if primal slack-variables are listed as explicit variables, with explicit non-negativity restrictions. The same proof has then to be carried for an enlarged pivot matrix.

$$E = \left[\begin{array}{c|c|c} D_{1,1} & & A_{1,1} \\ \hline & & -I \\ \hline A_{1,1} & I & \end{array}\right] \qquad (16.6.21)$$

which completes the proof for dual variables p_i.
q.e.d.

The symmetry of the tableau now leads to the following:

Corollary:

If a standard form tableau refers to a solution which is convex in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, all primal variables' rows display one of the following characteristics:

Either: a) the diagonal cell associated with the row in question is negative non-zero

or b) all dual variables' column-entries in such a primal variables' row are zero.

Note:

The "matrix analogy" of the above theorem:

"A dual variables' block-column is either free of non-zero elements in primal variables' rows, or the diagonal block is negative definite", is not true, as may be illustrated by the following example:

Maximise $x_1 + x_2 - (x_1 - x_2)^2$

subject to $x_1 + x_2 \leq 1 \quad (x_1, x_2 \geq 0)$

The optimum tableau of this problem is given in tableau 16.6b below

TABLEAU 16.6 B

ILLUSTRATION OF THE NEGATIVE DIAGONAL.

NAME	!!	D 1	D 2	S 1	!!	VALUE
X 1	!!	-0.12	0.12	0.50	!!	0.50
X 2	!!	0.12	-0.12	0.50	!!	0.50
P 1	!!	-0.50	-0.50	-	!!	1
2T	!!	-0.50	-0.50	1	!!	2

There are two dual variables' columns in this tableau (d_1 and d_2). The first theorem of this section tells us that they contain non-positive entries on the main diagonal (the two entries of -1/8, printed as -0.12).

The second theorem tells us that, since the excess of the binding restrictions is 2-1 = 1, there should be a partitioning which yields a 1 by 1 negative-definite diagonal block $\underline{x}^*_2/\underline{d}^*_2$. Either of the two diagonal cells of -1/8 will do.

And the last theorem tells us that in a dual variable's column which contains non-zero entries in primal variables' rows (both of them), the diagonal cell is negative non-zero. They both are.

The top left-hand block is nevertheless negative semi-definite rather than negative definite - as may be illustrated by

$$\begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}-1/8 & 1/8\\ 1/8 & -1/8\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = 0$$

However, the weaker theorem:

"If a standard form tableau refers to a solution which is convex in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, the square block formed by the intersection of primal variables' rows and dual variables' columns in negative semidefinite."
is true. (n-dimentional generalization of (16.6.5))

Furthermore, since, as far as subspaces associated with invertable block-pivots is concerned, subspace convexity implies strict subspace convexity ($\Delta\underline{x}_1'\ D_{11}\ \Delta\underline{x}_1 \leq 0$ for all $\Delta\underline{x}_1$ satisfying $A_{11}\ \Delta\underline{x}_1 = 0$

implies $\Delta\underline{x}_1'\ D_{11}\ \Delta\underline{x}_1 < 0$ for all $\Delta\underline{x}_1' \neq 0$ satisfying $A_{11}\ \Delta\underline{x}_1 = 0$)

the negative semi-definiteness of the symmetric top left hand block of the pivot inverse is a sufficient condition to prove subspace convexity, and by implication strict convexity, and hence the existence of a negative definite block $T_{2,2}$ of the appropriate order.

Exercise

The following QP problem is given

Maximise $\tau = x_1 - x_2 - x_1^2 + x_1\ x_2 - 2x_2^2$

Subject to $x_1 + x_2 \geq 10$

Solve this problem

For all standard form tableaux developed verify the symmetry properties, i.e. check that they conform, if necessary after re-ordering), to the properties stated in section 16.4.

Also verify the convexity properties of the same tableaux. Extract the block, $T_{2,2}$ as well as the larger semi-definite block $\begin{bmatrix} T_{1,1} & T_{2,2} \\ T_{2,2} & T_{2,2} \end{bmatrix}$ from each tableau and indicate which of the other non-positive diagonal cells are non-positive by any property stated in this section, or only because this is a problem in which the objective function is convex throughout the coordinate space.

16.7 The non-standard form tableau and its standard form neighbours

Most of the theorems to be discussed in this section concerns a (2 by 2) 4 cell block of the non-standard form tableau consisting of the intersection of the badname variable row and the driving variable row, with the last leaving variable column and the incoming variable column.

We call this block the non-standard form block. In association with this block, we define 4 standard form tableaux, which may or may not exist.

The smaller subspace predecessor tableau is obtained by introducing the last leaving variable back into the basis, in exchange for the driving variable. The smaller subspace successor tableau is obtained by introducing the incoming variable into the basis in exchange for the driving variable. The larger subspace predecessor tableau is obtained by introducing the last leaving variable into the basis, in exchange for the badname variable. The larger subspace successor tableau is obtained by introducing the incoming variable into the basis in exchange for the badname variable.

Together these four standard form tableaux (as far as they exist) are indicated as the neighbouring standard form tableaux, or standard form neighbours. The terms "smaller" subspace and "larger" subspace refer to the situation where with an initially feasible basis, the badname variable is a dual variable and the driving variable is a primal variable. They will, however be used in the general situation where the badname could be a primal variable, i.e. a negative valued slack-variable and the connotation of "more space" and "less space" may not be applicable, even though the terms are used.

Theorem (non standard form block non-singularity theorem). The non-standard form block is invertable, its determinant being either +1 or -1.

Proof:

Suppose the theorem to be true for some non-standard form tableau.

We now consider the implication of the column-updating rule: the last leaving variable's column is obtained by division by minus the pivot (see section 8.8). If (as assumed) the theorem is true for the current non-standard form tableau, the last leaving variable column of the non-standard form block of a non-standard form successor tableau contains at least one non-zero element, and either the smaller subspace predecessor tableau of the successor tableau (= the smaller subspace successor tableau of the current tableau) exists, or else the larger subspace predecessor tableau of the non-standard form successor tableau (= the larger subspace successor tableau of the current non-standard form tableau exists.)

If the smaller subspace predecessor exists, we may summarize its relation with the actual tableau, as follows:

Non standard form tableau

	last leaving v	incoming v
badname v	?	?
driving v	$\neq 0$	?

Smaller subspace predecessor

	driving v	incoming v
badname v	?	$\neq 0$
last leaving v	$\neq 0$	?

By the symmetry property of the standard form tableau, the badname variable row/incoming variable column cell of the standard form tableau is of the same absolute value as the last leaving variable row/driving variable column cell, which is the reciprocal of the pivot used to create the standard form tableau.

The other non-zero diagonal cell now provides the other pivot needed to calculate the inverse of the non-standard form block.

If the smaller subspace predecessor does not exist, the larger subspace predecessor exists, and pivoting on the badname variable row/last leaving variable column cell allows us to invoke the same argument with respect to the symmetry property of the larger subspace predecessor tableau.

Hence the theorem assumed to be true for any current non-standard form tableau, implies the validity of the same theorem for its non-standard form successor tableau (if one exists). The smaller subspace predecessor of the first non-standard form successor tableau of a standard form tableau is that some standard form tableau itself.

Therefore the theorem is true for the first of any series of non-standard form tableaux, and by implications for all its non-standard form successor tableaux.
q.e.d.

The following corrollae - inferences of the fact that a non-singular matrix contains at least one non-zero entry in each row/column -, may now be stated: If the smallers subspace predecessor tableau does not exist, then the smaller subspace successor and the larger subspace predecessor tableaux exist.

The non-standard form block plays a crucial role in recognizing the transmission of convexity properties between its standard form neighbours. Before we proceed to stating the theorems and their proofs, it is useful to introduce one more term. The subspace described by standard form tableau Q is said to be contained in the subspace described by standard form tableau T, if T may be transformed into Q, by eliminating a primal variable from the basis of the solution described by T, bringing the corresponding dual variable into the basis instead.

There are two cases. If the eliminated variable is a primal slack-variable, D_{11} remains the same but A_{11} loses a row. If the eliminated variable is an element of $\underline{x}$, the order of D_{11} is reduced by one, and A_{11} loses a column.

Since the definition of subspace convexity ($\Delta\underline{x}_1' D_{11} \Delta\underline{x}_1 \leq 0$ for all $\Delta\underline{x}_1$ satisfying $A_{11}\Delta\underline{x}_1 = 0$) includes vectors $\Delta\underline{x}_1$ for which one element is zero the subspace convexity of Q is implied by the by the subspace convexity of T, if T obeys that property.

We now state the theorems

Theorem (smaller subspace immediate neighbour convexity transmission theorem)

If the smaller subspace predecessor tableau exists, and describes a solution which is strictly convex in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace then a sufficient condition for the smaller subspace successor tableau to exist and obey the same convexity property is that the entry in the driving variable row/last leaving variable column cell is positive (non-zero), while the driving variable row/incoming variable column cell is negative non-zero.

Proof

Existence:

Obvious: the negative non-zero number is the pivot needed to generate it

Convexity:

The condition implies the presence of a negative non-zero diagonal entry in the last leaving variable row/incoming variable column cell in the smaller subspace predecessor tableau, as may be illustrated by summarizing the transition from the non-standard form tableau to the smaller subspace predecessor tableau

Non standard form tableau

	last leaving variable	incoming variable
driving variable	+	-

Smaller subspace predecessor

	driving variable	incoming variable
last leaving variable	+	-

This negative diagonal cell provides the pivot needed to transform the smaller subspace predecessor tableau into the smaller subspace successor tableau. If the incoming variable is a primal variable

this cell provides a further negative pivot in $T^*_{2,2}$ as defined in the previous section. If the incoming variable is a dual variable, no further proof of convexity is needed the subspace defined by the successor tableau is contained in the one defined by the predecessor tableau.
q.e.d.

We might formulate the theorem as stated above somewhat more compact, by introducing the convex transition. A condition is said to indicate the convex transition from the standard form tableau T to the standard form tableau Q, if the condition is sufficient to prove that Q describes a solution which is convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, provided T does so. We will however reserve the use of the term convex transition for the discussion of examples, or where the meaning of the term is self-evident without reference to the definition, and formulate the theorems themselves explicitly.

The same argument concerning the implied existence of a negative diagonal pivot gives rise to the following parallel theorems.

If the smaller subspace predecessor tableau exists, and describes a solution which is strictly convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace then a sufficient condition for the smaller subspace successor tableau to exist and obey the same convexity property is that entry in the driving variable row/ last leaving variable column cell is negative (non-zero) while the driving variable row/ incoming variable column cell is positive non-zero. (Upon developing the smaller subspace predecessor, division by a negative pivot gives rise to a negative diagonal cell).

If the larger subspace predecessor tableau exists, and describes a solution which is strictly convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, then a sufficient condition for the larger subspace successor tableau to exist and obey the same convexity property is that the entry in the badname row/last leaving variable column cell is positive (non-zero), while the badname row/incoming variable column cell is negative non-zero.

Note however, that since column-updating is by division by minus the pivot, the similar theorem concerning the relation between smaller and larger subspace successors requires equal signs. If the smaller subspace successor tableau exists and describes a solution which is strictly convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, then a sufficient condition for the larger subspace successor tableau to exist and obey the same convexity property is that the entry in the driving variable row/incoming variable column cell is negative (non-zero) while the badname-variable row/incoming variable cell is also negative non-zero.

The following arrangement of the signs of the four elements of the non-standard form block allows all four neighbouring non-standard form tableaux to exist and they are either all convex, or none of them describe convex solutions:

	last leaving variable	incoming variable
badname variable	+	-
driving variable	+	-

with all four cells containing non-zero numbers of the signs as indicated. The complete reversal of all of these signs, i.e.

	last leaving variable	incoming variable
badname variable	-	+
driving variable	-	+

has the same implications.

The presence of one of these two arrangements (in practice only the first one is used) is also a necessary condition for the convexity of all four neighbouring standard form tableaux.

Theorem

If both entries in the driving variable's row of the non-standard form block are non-zero and of the same sign,
then both the smaller subspace predecessor tableau and the smaller subspace successor tableau exist , and at least one of the two describes a solution which does not obey the property of convexity in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace.

Proof

By the complementary rule, the last leaving variable and the incoming variable are each other's complements. Therefore at least one of them is a dual variable. Upon pivoting on the primal variable's entry in the driving variable's row, we develop a standard form tableau, in which a positive non-zero figure, the ration between the two, figures in the diagonal cell of a dual variable's column, contradicting the convexity property of the tableau as thus developed. (The existence of the other tableau is obvious).
q.e.d.

The parallel theorem concerning the badname-row and the larger subspaces will be obvious and is not formulated explicitly here.

The parallel theorem concerning two successor tableaux (or two predecessor tableaux) is again slightly different: if two cells in the same column are both non-zero, but of different signs at least one of the two standard form tableaux describes a solution which is not convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace.

We may refer to the group of theorems concerning the convex transitions between the immediate neighbours of a non-standard form tableau as the immediate neighbour convexity transmission theorems.

The non-standard form block non-singularity theorem does not require that all four elements of the non-standard form block are non-zero. On the contrary, if a primal badname is chosen in the set-up tableau (i.e. we wish to make a violated restriction binding), the first smaller subspace predecessor (the set-up tableau itself) contains a zero in the badname row/incoming variable cell, and one always develops a non-standard form block which contains at least one zero element.

Example

Maximise $\tau = -x_1 - x_2 - x_1^2 - x_2^2$

Subject to $x_1 + 2x_2 \geq 3$

$(x_1, x_2 \geq 0)$, see tableau 16.7a below

TABLEAU 16.7 A

SET-UP TABLEAU FOR ILLUSTRATING
THE NON-STANDARD FORM BLOCK.

NAME	X1	X2	P1	VALUE
D1	-2	-	1	1
D2	-	-2	②	1
S1	-1	-2	-	-3
2T	1	1	3	-

The first badname obviously is s_1, therefore p_1 is the driving variable and a non-standard form tableau is developed, which is tableau 16.7b. Note that this tableau has been re-ordered.

TABLEAU 16.7 B

NON-STANDARD FORM TABLEAU,
WITH NON-STANDARD FORM BLOCK.

NAME	!!	X1	D2	X2	!!	VALUE
D1	!!	-2	-0.50	1	!!	0.50
S1	!!	-1	-	-2	!!	-3
P1	!!	-	0.50	-1	!!	0.50
2T	!!	1	-1.50	4	!!	-1.50

From tableau 16.7b we extract the non-standard form block as given below.

	d_2 last leaving v.	x_2 incoming v.
s_1 (badname variable)	-	-2
p_1 (driving variable)	$\frac{1}{2}$	-1

The driving variable row confirms the convex transition from the trivial basis to the x_2 subspace - not surprising since this is a convex problem. The badname row does not provide any similar information at least not directly but the x_2 column tells us that the x_2 subspace is convex, therefore, so is the $x_2 = 1\frac{1}{2}$ subspace.

The following theorems relating to zeros in the non-standard form block are now stated.

If the driving variable row/last leaving variable column cell contains a zero entry,
then the smaller subspace predecessor of the current non-standard form tableau does not exist, but the smaller subspace predecessor of the immediately preceding non-standard form tableau does exist.

Proof

We first note that the zero contradicts the development of the current non-standard form tableau from a standard form tableau (otherwise that cell would contain the reciprocal of the pivot). Hence the smaller subspace predecessor does not exist and the current tableau was developed from a non-standard form tableau.

By the column updating rule that non-standard form predecessor tableau contains a zero in the driving variable row/incoming variable column. The non-standard form non-singularity theorem therefore requires the existence of a non-zero entry in the driving variable row/last leaving variable column cell of the non-standard form predecessor tableau, proving the existence of the previous tableau's smaller subspace predecessor. q.e.d.

Concerning the similar theorem for the badname-variable and the larger subspace we note that the case where a non-standard form tableau displays a zero in the badname row/last leaving variable column cell, but has no non-standard form predecessor does exist - it was illustrated above.

Otherwise the theorem is listed above for the driving variable is the same:

If the badname row/last leaving variable column cell contains a zero entry while a non-standard form predecessor tableau exists, then the larger subspace predecessor tableau of the non-standard form block the transition from a standard form predecessor tableau to the standard form successor tableau of the next non-standard form tableau may be proved by way of a pair of complementary steps.

A pair of complementary steps consists of two steps which involve pivots of the same absolute value, opposite complementary name-codes, and a zero in one of two diagonal cells which link the row of the one to the column of the other and vice versa.

The 2 by 2 block

	first incoming v.	second incoming v.
second leaving v		-p
first leaving v	p	d

is called the pair block

Theorem (complementary pair theorem)

If a standard form tableau (hereafter indicated as T), can be transformed into another standard form tableau (hereafter indicated a Q), by a complementary pair then:

either T and Q both describe solutions which are convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace,
Or neither of them does.

Proof:

An extract from T may be summarized as follows:

	$\underline{d}_2^*$	x_j	d_k
$\underline{x}_2^*$	T_{22}	$-\underline{v}$	$\underline{u}$
d_j	$\underline{v}'$		$-p$
x_k	$\underline{u}'$	p	d

or

	$\underline{d}_2^*$	d_j	x_k
$\underline{x}_2^*$	T_{22}	$\underline{v}$	$-\underline{u}$
x_j	$\underline{v}'$		$-p$
d_k	$\underline{u}'$	p	d

where the first incoming variable associated with $\underline{v}$ and $-\underline{v}$ is a primal variable, indicated as x_j (it could also be s_i), the second one associated with $\underline{u}'$ and $-\underline{u}$ is then a dual variable. The block $T_{2,2}$ is the negative-definite block $T_{2,2}$ as proved in section 16.6, provided T conforms to subspace convexity.

The opposite signs of p and -p imply that of the two incoming variables one is a primal variable and the other a dual variable; if they were both of the same class there would be full symmetry between the two pivots.

The alternating sign presentation of $\underline{v}$ and $\underline{u}$ follows from this. Alternatively, if the first incoming variable is the dual variable, we would write $\underline{v}$ and $-\underline{u}$ but the same result follows)

Concerning the convex transition from T to Q: We assume that T describes a convex solution. If $T_{2,2}$ is of zero order the theorem is trivial since both subspaces reduce to single points. Otherwise, the equivalent block of Q_{22} is expressed as:

$$Q_{22} = T_{2,2} + 2\,\underline{u}\,\underline{v}'\,d/p^2 + \underline{v}\,\underline{v}'\,d/p^2 \qquad (16.7.1)$$

if the first incoming variable is a primal variable and

$$Q_{2,2} = T_{2,2} - 2\,\underline{u}\,\underline{v}'\,d/p^2 + \underline{v}\,\underline{v}'\,d/p^2 \qquad (16.7.2)$$

if the first incoming variable is a dual variable.

For d = 0, convex transition follows in both orderings, because $Q_{2,2}$ is equal to $T_{2,2}$, hence negative definite if $T_{2,2}$ is negative definite. If we assume d < 0, convex transition follows by inversion of the pairblock along the main diagonal as may be summarized as follows:

$$\begin{bmatrix} & -p \\ p & (d) \end{bmatrix} \qquad \begin{bmatrix} (p^2/d) & p/d \\ p/d & 1/d \end{bmatrix} \qquad \begin{bmatrix} d/p^2 & 1/p \\ -1/p & - \end{bmatrix}$$

For d < 0 both pivots, d and p^2/d are negative, and we therefore have proved convex transition from T to Q for d < 0. For d > 0, convexity of T implies that the first incoming variable is the dual variable, the second one the primal one, as manifest non-convexity only occurs in primal variables' columns. But if so the first one must be a dual variable, and the zero on the diagonal implies $\underline{v} = 0$, and convex transition follows for the same reason as proved for d = 0, $Q_{2,2}$ being identical to $T_{2,2}$. We therefore prove convex transition from T to Q, for d = 0, for d < 0, and for d > 0.

Concerning the convex transition from Q to T: The above calculation of the inverse of the pairblock shows that this transition is also by a complementary pair.

Having proved convex transition in both directions it also follows that if T did not describe a convex solution in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, then neither does Q, otherwise the convex transition from Q to T would contradict the assumed non-convexity of T. The same argument applies to non-convexity of Q. q.e.d.

We may now use this theorem to prove convex transition from a smaller subspace predecessor of one tableau to the smaller subspace successor of the next tableau, without actually developing either tableau.

Example

Maximise $\tau = 3x_1 + x_1^2 + x_1 \cdot x_2 - x_2$

Subject to $x_1 + x_2 \leq 10 \qquad (x_1, x_2 \geq 0)$

The set-up tableau of this problem is given in tableau 16.7c, which is given below, together with its non-standard-form successor tableau.

TABLEAUX 16.7 C AND 16.7 D

SET-UP TABLEAU AND FIRST NON-STANDARD FORM TABLEAU, FOR ILLUSTRATION OF CONVEX TRANSITION.

NAME	!!	X1	X2	P1	!!	VALUE
D1	!!	2	1	-1	!!	-3
D2	!!	①	-	-1	!!	1
S1	!!	1	1	-	!!	10
2T	!!	-3	1	-10	!!	-

NAME	!!	D2	X2	P1	!!	VALUE
D1	!!	-2	1	1	!!	-5
X1	!!	1	-	-1	!!	1
S1	!!	-1	①	1	!!	9
2T	!!	3	1	-13	!!	3

In tableau 16.7c we choose d_1 as badname-variable, and we assume that d_2 is selected as leaving variable, this row indicating the smallest non-negative quotient.

In the resulting non-standard form tableau 16.7d, the non-standard form block has been marked; it is also given in a separate extract below.

	d_2 last leaving v.	x_2 incoming v.
d_1 badname v.	-2	1
x_1 driving v.	1	-

We note that this tableau has no smaller subspace successor tableau that is, the driving tableau x_1 cannot be eliminated by the incoming variable x_2. However, if we first develop a further non-standard form successor tableau - by eliminating s_1, then we know that <u>that</u> tableau has a smaller subspace successor tableau.

This non-standard form successor tableau is tableau 16.7e below.

TABLEAU 16.7 E

A FURTHER NON-STANDARD FORM SUCCESSOR TABLEAU, WITH NON-STANDARD FORM BLOCK.

NAME	!!	D2	S1	P1	!!	VALUE
D1	!!	-1	-1	-	!!	-14
X1	!!	1	-	-1	!!	1
X2	!!	-1	1	1	!!	9
2T	!!	4	-1	-14	!!	-6

The smaller subspace predecessor of the first non-standard form tableau 16.7d is the set-up tableau 16.7c, the smaller subspace successor tableau of the second non-standard form tableau 16.7e may be obtained by eliminating the driving variable x_1 against p_1 as incoming variable. This step, (which we would not make in actual calculation) leads to tableau 16.7f below, (re-ordered as 16.7g.)

TABLEAUX 16.7 F AND 16.7 G

SMALLER SUBSPACE SUCCESSOR TABLEAU OF TABLEAU 16.7 E, AS DEVELOPED AND RE-ORDERED TO STANDARD FORM.

NAME	!!	D2	S1	X1	!!	VALUE
D1	!!	-1	-1	-	!!	-14
P1	!!	-1	-	-1	!!	-1
X2	!!	-	1	1	!!	10
2T	!!	-10	-1	-14	!!	-20

NAME	!!	X1	D2	S1	!!	VALUE
D1	!!	-	-1	-1	!!	-14
X2	!!	1	-	1	!!	10
P1	!!	-1	-1	-	!!	-1
2T	!!	-14	-10	-1	!!	-20

Theorem (smaller subspace complementary pair transition theorem)

If two successive non-standard form tableaux, of which the first one was transformed into the second one (by a positive pivot), do not have a common smaller subspace successor/predecessor while the smaller subspace predecessor of the first non-standard form tableau describes a solution which is strictly convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace.

then

a sufficient condition for the smaller subspace successor tableau of the second non-standard form tableau to obey the similar convexity property is that the entry in the driving variable row/last leaving variable column cell in the first non-standard form tableau is positive non-zero, and the driving variable row/incoming variable column cell of the second tableau is negative non-zero. (the two other cells in the two driving variable rows being zero).

Proof:

The assumptions stated imply the following sign and figure arrangement in a 6 cell extract from the smaller subspace predecessor tableau of the first non-standard form tableau.

smaller subspace predecessor tableau (standard form)

	driving v.	1st incoming v.	2nd incoming v.
badname v	?	p or -p	?
1st leaving v	p	0	-q
2nd leaving v	?	q	d

where p is the reciprocal of the pivot used to transform the first non-standard form tableau into its smaller subspace predecessor - hence by assumption $p > 0$ -, q is the pivot used to transform the first non-standard form tableau into the second one - hence by assumption $q > 0$, and d is the leaving variable row/next incoming variable column cell of the first non-standard form tableau. This shows that the complementary pair theorem is applicable and the theorem follows.
q.e.d.

The similar parallel theorem with respect to the badname-variable row and the larger subspace follows by analogy, and will be referred to as the larger subspace complementary pair transition theorem.

Exercise:

The following QP problem is given

Maximise $\tau = x_1 - x_2 - x_1^2 + x_1 \cdot x_2 - 2x_2^2$

Subject to $x_1 + x_2 \geq 10$ $(x_1, x_2 \geq 0)$

Solve this problem starting with s_1 as badname (If you did not already do this at the end of section 16.6 this being the same problem). For each non-standard form tableau developed

a) Extract the non-standard form block
b) Develop the neighbouring standard form tableaux (as far as they exist). Indicate where possible, which one of them may be transformed into one of the others by
b1) a diagonal pivot, or
b2) by a complementary pair.

16.8 Dual variable elimination and the parameter theorems

Three criteria are relevant in deciding whether to drive a dual variable out by the usual pivoting elimination process or alternatively to possibly breach its non-negativity. We may outline these criteria as convergence, boundedness and optimality; this section is devoted to the first problem.

The issue of convergence is linked with that of maintaining subspace convexity, as may be illustrated by the following example, which relates to a non-convex problem.

Maximise $\tau = x_1^2 - x_1 + x_1 \cdot x_2 - x_2$

Subject to $x_1 + x_2 \geq 5$ $(x_1, x_2 \geq 0)$

We must obviously start with s_1 as badname but if we apply the rule of the smallest quotient as known from LP without modification and combine it with the complementarity rule, we get into an endless cycle, as may be illustrated by actually going through the cycle once.

This has been illustrated in the tableau-series 16.8 below.

TABLEAU-SERIES 16.8

ILLUSTRATION OF CYCLING IN A NON-CONVEX PROBLEM.

16.8 A (SET-UP)

NA.	X1	X2	P1	VAL.
D1	2	1	②	1
D2	1	-	1	1
S1	-2	-1	-	-5
2T	1	1	5	-

16.8 B (NON-STAND. F.)

NA.	X1	X2	D1	VAL.
P1	①	0.5	0.5	0.5
D2	-	-0.5	-0.5	0.5
S1	-2	-1	-	-5
2T	-4	-1.5	-2.5	-2.5

16.8 C (STANDARD FORM)

NA.	P1	X2	D1	VAL.
X1	①	0.5	0.5	0.5
D2	-	-0.5	-0.5	0.5
S1	2	-	1	-4
2T	4	0.5	-0.5	-0.5

16.8 D (= 16.8 B)

NA.	X1	X2	D1	VAL.
P1	1	0.5	(0.5)	0.5
D2	-	-0.5	-0.5	0.5
S1	-2	-1	-	-5
2T	-4	-1.5	-2.5	-2.5

16.8 E (= SET-UP)

NA.	X1	X2	P1	VAL.
D1	2	1	2	1
D2	1	-	1	1
S1	-2	-1	-	-5
2T	1	1	5	-

In the set up tableau 16.8a, the driving variable p_1 is the incoming variable, and the smallest quotient is found in the d_1 row. Tableau 16.8b is a non-standard form tableau, with x_1 as incoming variable, by the complementarity rule; the smallest non-negative quotient is found in the p_1 row.

This elimination of the driving variable p_1 leads to a standard form tableau, except that tableau 16.8c has not been re-ordered to the standard form presentation.

We again choose s_1 as badname, and p_1 as driving variable. The smallest non-negative quotient is now found in the x_1 row, and we make the previous step in opposite direction. Tableau 16.8d is identical to tableau 16.8b, but - coming from a different standard form tableau, the complementarity rule now indicates d_1 as incoming variable, and we also make the first step in the opposite direction.

The following exception to the rule of the smallest quotient, as far as dual leaving variables is concerned is sufficient to ensure convergence.

A dual variable may be refused as a leaving variable on account of failing to satisfy the driving variable increment qualifier. If advance calculation of the driving variable row/incoming variable column cell indicates that accepting a particular dual variable, would activate a complementary primal incoming variable, which displays a positive non-zero entry in the driving variable's row, then the dual variable in question is refused as leaving variable. It will be noted that this qualifier was breached in the example given above; d_2 should have been eliminated despite the fact that this would imply that d_1 would become negative. As a result, the non-convex variable x_1 was activated without guarantee of developing a subspace in which this variable would be contained, and the driving variable p_1 was again eliminated.

Theorem (driving variable parameter theorem)

If the driving variable increment qualifier is obeyed - and otherwise the rule of the smallest quotient applied -, then all non-standard form tableaux which are developed from a standard form tableau describing a solution which is convex in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace, display the following properties:

a) The entry in the driving variable row/incoming variable column cell is non-positive, and negative non-zero, if the similar entry in an immediately preceding non-standard form tableau was zero.

b) The entry in the driving variable row/last leaving variable column cell is non-negative, and positive non-zero if the similar entry in an immediately preceding non-standard form tableau was zero.

c) The smaller subspace predecessor and successor tableaux (as far as existing) describe solutions that are convex in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace.

Proof:

Concerning the successor tableau of an actual standard-form tableau:

ad a) For a primal leaving variable a positive entry in the diagonal cell would contradict the convexity property of the initial standard form tableau, hence a)

For a dual leaving variable a) is required by programme provision (driving variable increment qualifier), hence a)

ad b) The cell in question is the reciprocal of the pivot; hence b)

ad c) The smaller subspace predecessor is the actual standard form tableau, which is assumed to obey the convexity property referred to.

If the smaller subspace successor exists, a) and b) ensure convex transition (see the previous section). Hence c).

Concerning the successor tableau of a non-standard form tableau: assume the theorem to hold for the preceding non-standard form tableau. The same theorem, as applicable to the current non-standard form tableau, may then be shown as follows:

ad b) The non-negativity applies, because by assumption a) applies to the previous non-standard form tableau and column-updating by division by minus a positive pivot changes a non-positive entry into a non-negative one.

If the entry in the previous tableau's driving variable row/incoming variable column cell was zero, then, by the non-standard form block non-singularity theorem, the previous tableau's entry in the driving variable row/last leaving variable column cell was non-zero, hence positive non-zero. Hence b).

ad a) and c) If the smaller subspace predecessor exists, b) as proved above implies that the transformation between the current tableau and the smaller subspace predecessor is by way of a positive pivot.

The proof supplied for the successor tableau of an actual standard form tableau is therefore applicable, proving that the driving variable row/incoming variable column cell is non-positive. Furthermore, the (positive) non-zero entry in the last leaving variable column implies a negative non-zero entry in the previous tableau's driving variable row/incoming variable column cell and no property of non-zeroness of the driving variable row/variable column cell of the current tableau is stated by the theorem. Hence a), if the smaller subspace predecessor tableau exists.

Concerning c), we first note that if the smaller subspace predecessor tableau exists, its convexity property is already assumed, this tableau being identical to the smaller subspace successor tableau of the previous non-standard form tableau.

If the smaller subspace successor tableau does not exist, no further proof of c) is needed. If the smaller subspace successor tableau exists, a) and b) imply c), by the smaller subspace immediate neighbour convexity transmission theorem. Hence c), if the smaller subspace predecessor tableau exists.

If the smaller subspace predecessor tableau does not exist, the driving variable row/last leaving variable column cell of the current tableau, and the driving variable row/incoming variable column cell of the previous tableau are both zero. Therefore the driving variable row/incoming variable column of the current tableau is non-zero (otherwise the non-standard form block would be singular), and the same is true for the driving variable row/last leaving variable column cell of the previous tableau. The latter cell therefore is positive non-zero by property b) assumed to be valid for the previous tableau.

To show that the smaller subspace complementary pair theorem is applicable, we therefore still need to prove the non-positivity of the driving variable row/incoming variable column cell.

If the incoming variable is a primal variable this condition applies by programme requirement - the driving variable increment qualifier. If the incoming variable is a dual variable this follows, because the opposite would contradict the symmetry property of the previous tableau's smaller subspace predecessor tableau (see also the proof of the smaller subspace complementary pair transition theorem). Hence a) and c).

The proof now follows by recursive induction q.e.d.

Note that no programme restriction applies to the driving variable itself, which may, in case of a dual badname (and therefore a primal driving variable), and a non-convex programming problem, display manifest non-convexity.

The corresponding theorem for the larger subspace and the badname-variable is therefore weaker.

Theorem (weak badname variable parameter theorem)

If the driving variable increment qualifier is obeyed - and otherwise the rule of the smallest quotient applied - non-standard form tableaux which are developed from a standard form tableau describing a solution that is convex in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace, and of which one (indicated as the current one), obeys the following properties:

a) the entry in the badname variable row/incoming variable column cell is non-positive (and negative non-zero if the similar cell in any immediately preceding non-standard form tableau was zero)

b) the entry in the badname variable row/last leaving column cell is non-negative (and positive non-zero if the similar entry in any immediately preceding non-standard form tableau was zero)

Then

All non-standard form successor tableaux of the current non-standard form tableau (if any) obey the same properties, and furthermore, the current and its successor tableaux also obey

c) the larger subspace predecessor and successor tableaux (as far as existing), describe solutions that are conved in the $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace.

Summary of the proof

ad c) If all four standard form neighbours of the current tableau exist, convex transition from the smaller subspace tableaux to the corresponding larger subspace tableaux follows from the appropriate immediate neighbour convex transition theorem. If there are zeros in the non-standard form block, the applicability of the complementary pair theorem may be proved, showing convex transition from the smaller subspace predecessor to the larger subspace successor tableau, or from the smaller subspace successor to the larger subspace predecessor tableau, as the case may be. If there is a zero in the driving variable's row, and two non-zero elements in the badname row, convex transition to one of the two larger subspaces follows by the complementary pair theorem, to the other by one of the immediate neighbour convex transition theorems.

Hence c) in all cases for the current tableau. From this point onwards, the proof is analogous to the one supplied for the driving variable parameter theorem.

End of proof summary.

The difference between the (strong) driving variable parameter theorem and the weak badname variable parameter theorem arises on account of two points:

1) there is no programme requirement with respect to the badname-row/incoming variable cell,

and

2) in the case of a dual badname in a non-convex programming problem there may initially be manifest non-convexity in the diagonal badname-row/driving variable column cell, i.e. this cell may contain a positive non-zero entry.

However, in the case of a primal badname, i.e. the negative valued slack of a violated restriction, we have the following:

Theorem (primal badname convexity theorem)

If a series of non-standard form tableaux is developed by selecting a primal badname-variable in a standard form tableau which satisfies the property of subspace convexity (and in the case of a dual leaving variable the driving variable increment qualifier is obeyed),
then
the first non-standard form tableau satisfies properties a) and b) listed above for the badname-row.

Proof

If the leaving variable is a dual variable the transition of the initial standard form tableau to the non-standard form block is summarized as follows:

Standard form tableau

	driving v.	next incoming v.
badname v.	-/0	-
leaving v.	⊕	-/0

Non-standard form block

	last leaving v.	incoming v.
badname v.	+/0	-
driving v.	+	-/0

In the standard-form tableau the leaving variable row/driving variable column cell (marked " + "), is positive non-zero because it is the pivot, the badname-row next incoming variable

cell then is negative non-zero by the symmetry rules. The badname row/driving variable column cell (marked "-/0") is non-positive because non-convexity, if present, is not manifest in dual variables' columns. The driving variable row/next incoming variable column cell is required to be non-positive by programme provision, on account of the driving variable increment qualifier.

The sign properties of the non-standard form block as marked, now follow from the logic of the pivoting operation. If the leaving variable is a primal variable the similar transition is summarized as follows:

Standard form tableau

	driving v.	next incoming v.
badname v.	d_1	p
leaving v.	p	d_2

Non-standard form block

	last leaving v.	incoming v.
badname v.	$-d_1/p$	$p-d_1 \cdot d_2/p$
driving v.	$1/p$	d_2/p

The assumed convexity properties of the initial standard-form tableau imply that the block

$$\begin{bmatrix} d_1 & p \\ p & d_2 \end{bmatrix}$$ is negative semi-definite

Also, since both columns are dual variables columns with a non-zero entry p in a primal variable's row, not only must $d_1 \leq 0$, $d_2 \leq 0$ be assumed, but $d_1 < 0$, $d_2 < 0$ as well.

The requirement of negative semidefiniteness now implies (after pivoting on d_1)

$$d_2 - p^2/d_1 \leq 0,$$

hence, after multiplication by d_1/p $(d_1/p < 0)$, $p - d_1 \cdot d_2/p \leq 0$, proving the assumed sign properties in all cases.
q.e.d.

In the case of a dual badname-variable, we may not necessarily assume that conditions a) and b) are initially satisfied for the badname-variable row. However, the only accepted way of again developing an actual standard form tableau, is by eliminating the badname-variable.

Since that is a negative-valued variable, it must be done by way of a negative pivot, it obviously follows that for the last non-standard form tableau, of which the new actual standard form tableau is the larger subspace successor tableau, property a) applies by way of a negative non-zero pivot in the badname-row/incoming variable cell.

Convex transition to the new standard form tableau now follows in one of the following two ways: if the corresponding smaller subspace successor tableau also exists, convex transition from the smaller subspace successor to the larger successor tableau follows by the immediate neighbour theorem.

If the corresponding smaller subspace successor tableau does not exist, convex transition from the smaller subspace predecessor to the larger subspace successor tableau follows by showing applicability of the complementary pair theorem.

This proves the property of subspace convexity for the next standard form tableau in all cases, provided the previous one obeyed that property.
q.e.d.

Exercise

The following QP problem is given

Maximise $\tau = x_1^2 + x_1 \cdot x_2 - x_1 - 2x_2$

Subject to $x_1 + x_2 \geq 5$

$3x_1 + x_2 \leq 30$

$(x_1, x_2 \geq 0)$

Solve that problem.

(Answer-sheet at the end of the chapter)

16.9 Step-length and objective function value

So far, the QP problem has been discussed in terms of finding a solution to both the primal requirements $A\underline{x} \leq \underline{b}$ and the optimality conditions $D\underline{x} - A'\underline{x} \leq -\underline{w}$, and it has been assumed that the rule of the smallest quotient is used as a method to select the leaving variable, whenever that is practicable.

If the badname variable is a primal variable, that is an efficient procedure although there are - in a non-convex problem - certain complications to be considered.

If the badname variable is a dual variable, there is no convergence or boundedness problem, but the procedure is not very efficient. This assymetry arises, on account of the direction in which the objective function changes: it increases in response to a dual badname variable and a primal driving variable, but its value is reduced in the presence of a primal badname-variable and a dual driving variable.

From (16.2.7), the Lagrangean expression

$$L = \underline{w}' \underline{x} + \tfrac{1}{2} \underline{x}' D \underline{x} + \underline{p}'(\underline{b} - A \underline{x})$$

we obtain the following differential expression

$$\Delta L = \underline{w}'\Delta \underline{x} + \underline{x}'D\Delta\underline{x} - \underline{p}'A'\Delta\underline{x} + (\underline{b} - A \underline{x})' \Delta \underline{p}$$

$$= -\underline{d}'\Delta \underline{x} + \underline{s}' \Delta \underline{p} \qquad (16.9.1)$$

The second-order term $\frac{1}{2} \Delta \underline{x}' D \Delta \underline{x}$ has been omitted from the right-hand side of (16.9.1) and the symbol Δ should be read as indicating differentiation, not a vector of finite differences. (The use of the symbol d might give rise to confusion with the vector $\underline{d}$).

The complementary slackness condition, upheld for all variables except the basic pair, causes (16.9.1) to collapse into only one term.

If the badname-variable is a negative shadowprice d_j, we have:

$$\Delta L = - d_j \Delta x_j \qquad (16.9.2)$$

Since no other term of the Lagrangean expression is present in (16.9.2), we immediate infer

$$\Delta\tau = - d_j \Delta x_j \qquad (16.9.3)$$

if d_j or x_j is the badname variable. The case of an element of $\underline{x}$ becoming the badname variable does not arise when applying the algorithm when starting at the trivial solution but could arise when re-entering a modified problem.

In that case (16.9.2) and (16.9.3) are also applicable.

The i^{th} slack variable may be re-specified as an explicit variable, bringing a new additional slack-variable into the problem. Thus, for example, the problem

Maximise $\tau = x_1 + x_2 - x_1^2$

Subject to $x_1 + x_2 \leq 10$

$(x_1, x_2 \geq 0)$

may be re-specified as

Maximise $\tau = x_1 + x_2 - x_1^2 + x_3$

Subject to $x_1 + x_2 + x_3 \leq 10$

$(x_1, x_2, x_3 \geq 0)$

Obviously the shadowprice d_3 is equal to p_1, as may be seen by writing out the dual requirement on x_3.

Therefore, by generalization of (16.9.3), we also have

$$\Delta\tau = - p_1 \Delta s_i \qquad (16.9.4)$$

if p_i or s_i is the badname variable.

If the badname-variable is a dual variable (d_j or p_i), we obviously have $d_j < 0$ or $p_i < 0$ and the value of the objective function increases with every increase in the driving variable.

If the badname variable is a primal variable, this is not the case, on the contrary, generally $d_j > 0$ or $p_i > 0$ and the value of the objective function drops with every increase (towards zero) of the badname-variable.

A similar conclusion is also apparent directly from (16.10.1), which becomes in the case of a primal variable s_i being badname-variable.

$$\Delta L = s_i \Delta p_i \qquad (16.9.5)$$

For $s_i < 0$, ΔL is obviously non-positive, and negative non-zero for $\Delta^1 p_i \neq 0$. The Lagrangean is in that case not identical with the objective function, but the two functions are equal to each other in a standard form tableau, and both are continuously falling (or not increasing) if the badname-variable is a primal variable.

Therefore, having postulated boundedness of the primal feasible space area in all directions, we accept a dual variable (other than a dual badname-variable, as leaving variable, only if the badname-variable is a primal variable.

The relative advantage of respecting dual restrictions when eliminating a primal badname variable is most apparent with a convex problem which is already in optimal form: the righthand-side stays non-negative in all dual variables.

16.10 Boundedness and artificial feasibility

Earlier in this chapter we postulated that each incoming variable always find a leaving variable. As far as the version of the algorithm which has been outlined so far is concerned, it does not in fact meet that obvious requirement of being always effective.

Example

Maximise $\tau = x_1^2 - x_1$

Subject to $x_1 \geq 2 \quad x_1 \geq 1$

$(x_1 \geq 0)$

The set-up tableau of this problem is given in tableau 16.10a

TABLEAU 16.10 A

ILLUSTRATION OF SPURIOUS UNBOUNDEDNESS

NAME	!!	X 1	P 1	P 2	!!	VALUE
D 1	!!	2	-1	1	!!	1
S 1	!!	1	-	-	!!	2
S 2	!!	-1	-	-	!!	-1
2T	!!	1	-2	1	!!	-

The negative-valued slack variable s_2 is the obvious badname variable, but the p_2 column can only be brought into the basis if d_1 is accepted as leaving variable, in breach of the driving variable increment qualifier. (If this step is made in breach of the qualifier, subsequent application of the complementarity rule leads to cycling.)

An incoming variable column is said to be unbounded if no leaving variable may be found against which the corresponding variable can be exchanged. If, on the other hand, the rules permit the elimination of some variable (possibly among others), we say that the incoming variable is bounded by that variable as leaving variable (or blocked, according to Cottle's terminology).

If an incoming variable is not bounded by any other variable as leaving variable, despite the existence of an optimal and feasible solution, we speak of a spuriously unbounded column.

In the example given, $x_1 = 2$ is the optimal and feasible solution, therefore the p_2-column is spuriously unbounded. There are two obvious approaches towards dealing with this problem, i.e. avoiding activation of spuriously unbounded columns, even where they exist, and the imposition of upper limits in all variables.

In a convex problem, the first approach is normally effective if preference is given to badname-variables which are associated with negative non-zero elements on the main diagonal.

Example

Maximise $\tau = x_1 - x_1^2$

Subject to $x_1 \geq 2$

$(x_1 \geq 0)$

The set-up tableau of this problem is given in tableau 16.10b below.

TABLEAU 16.10 B

P1 IS UNBOUNDED, X1 IS BOUNDED

NAME	!!	X 1	P 1	!!	VALUE
D 1	!!	-2	1	!!	-1
S 2	!!	-1	-	!!	-2
2T	!!	-1	2	!!	-

If s_1 is selected as badname-variable, we activate the unbounded p_1-column, but if d_1 is selected as the first badname-variable we activate the x_2-column as incoming variable column and the problem is readily solved in two steps.

In a non-convex problem, such avoidance of spuriously unbounded columns in this way is not always possible, and we even lack a proof in the convex case.

However, if the current solution is primal feasible, we have the following

Theorem

If the current basis represents a feasible solution - vector, not finding a leaving variable, after activating the incoming variable column, implies that the QP problem which one attempts to solve, permits an infinite increase in the value of the objective function, in a feasible direction (i.e. is unbounded).

Proof

In a tableau which represents a feasible solution the badname-variable is a dual variable, and the driving variable is a primal variable. It was shown in the previous section that an increase of a primal driving variable is linked - by (16.9.3) or (16.9.4), depending on whether the badname-variable is a dual slack-variable d_j or a specified dual variable p_i -, to a corresponding increase in the value of the objective function. Therefore, if the incoming variable is the driving variable, or if a non-zero (and therefore negative non-zero) entry occurs in the driving variable row/incoming variable column cell, no further proof is needed.

Hence, if there is spurious unboundedness, this assumes that the incoming variable is some other variable than the driving variable while (in a non-standard form tableau), the driving variable row/incoming variable column cell contains a zero entry.

The non-standard form block non-singularity theorem then tells us that there is a non-zero entry in the badname-row/incoming variable column cell.

Now consider the transition of the non-standard form block, to the corresponding extract of the smaller subspace predecessor tableau, which is summarized below, assuming that the determinant of the non-standard form block is -1, as follows:

Non-standard form tableau

	last leaving v.	incoming v.
badname v.	?	p
driving v.	(1/p)	0

Smaller subspace predecessor

	driving v.	incoming v.
badname v.	?	p
last leaving v.	p	0

If the entry in the badname-row/incoming variable column cell (marked "p") is in fact negative non-zero no further proof is needed either, as the incoming variable is bounded by the badname-variable (the marking assumes a positive non-zero entry).

Since no dual variables are permitted as leaving variables in the presence of a dual badname-variable, the incoming variable (in a non-standard form tableau activated by the complementarity rule) is a primal variable.

It is now readily verified that the two equal elements in the off-diagonal cells of the transformed non-standard-form block (marked "p") cannot, in fact, be equal to each other; they should be equal in absolute value but opposite in sign.

In that case the summary of the non-standard form block is in fact the following:

	last leaving v.	incoming v.
badname v.	?	-p
driving v.	1/p	0

On account of the driving variable parameter theorem we must then assume $p > 0$, and the incoming variable is found to be bounded by the badname variable as leaving variable. Therefore only the case of genuine unboundedness with a negative non-zero entry in the driving variable row/incoming variable column cell actually exists.
q.e.d.

To accommodate the situation where (most of the) elements of the initial vector $\underline{d}$ are non-negative and we wish to maintain this initial non-negativity of the dual solution in the presence of a starting solution which is not primal feasible, we postulate that each element of $\underline{x}$, and each element of $\underline{p}$ has an upper limit.

This does not require any significant additional tableau-space as we may generalize the device of the double value column (introduced in section 10.3 for the L.P. problem), to quadratic programming.

The interpretation of upper limits on the elements of $\underline{x}$ is straightforward, but the interpretation of upper limits on dual variables needs further discussion.

It is in fact readily verified that they correspond to artificial primal variables. At the cost of a huge penalty it is permitted to breach the primal restrictions, and the penalty coefficients are the upper limits on the dual variables.

For example if we suppress the non-active upper limit in x_1, the last example may now be written as

Maximise $\quad x_1 - x_1^2 - 100\, a_1$

Subject to $\quad -x_1 - a_1 \leq -2$

$\quad (x_1, \; a_1 \geq 0)$

and the dual restriction on a_1 is in fact $p_1 \leq 100$.

In the interest of being able to elucidate the significance this problem is written here in the form of an explicit set-up tableau (tableau 16.10c).

Here z_1 is the slack-variable associated with the dual restriction on the artificial variable.

TABLEAU 16.10 C

EXPLICIT FORMULATION OF
AN ARTIFICIAL VARIABLE.

NAME	!!	X1	A2	P1	!!	VALUE
D1	!!	-2	-	1	!!	-1
Z2	!!	-	-	1	!!	100
S1	!!	-1	-1	-	!!	-2
2T	!!	-1	100	2	!!	-

In the rest of this section the following conventions with respect to upper limit distances and their associated dual variables will be used. Artificial variables are indicated as a_i, the index i being also associated with the i^{th} primal restriction.

The dual slack associated with the optimality condition on the artificial variable a_i is then indicated as z_i (z_1 in the example above).

Distances between x_j (elements of $\underline{x}$) and their upper limits are indicated as b_j, conform section 10.3, for the LP problem. Bound-distances b_j should not be confused with the elements of $\underline{b}$ in $A\underline{x} \leq \underline{b}$, but this should be clear from the context.

Dual variables, associated with binding upper limit restrictions are indicated as u_j. (For the sake of completeness the associated numerical codes are also given here:

j for x_j, $1000 + i$ for s_i, $2000 + j$ for b_j,

$3000 + i$ for a_i,

$-j$ for d_j, $-1000-i$ for p_i, $-2000 -j$ for u_j,

$-3000 - i$ for z_i; these codes are the computational means to keep track of the complementary slackness condition and the complementarity rule.)

Just as in the LP case, one may economize on tableau-space, by suppressing the storage of unit vectors. This, however, gives rise to additional complications in the manipulation of tableaux. An example of the application of the upper limit facility (together with the full tableaux) is given here. It concerns the example used in the beginning of this section.

TABLEAU 16.10 D (SERIES)

ILLUSTRATION OF THE COMPACT FORM OF THE QP-TABLEAU.

EXPLICIT TABLEAUX

NA.	X1	A2	P1	P2	VAL
D1	2	-	-1	1	1
D2	-	-	-	(1)	100
S1	1	-	-	-	2
S2	-1	-1	-	-	-1
2T	1	100	-2	1	-

BRING IN THE DRIVING VARIABLE

NA.	X1	A2	P1	D2	VAL
D1	2	-	-1	-1	-99
P2	-	-	-	1	100
S1	1	-	-	-	2
S2	-1	(-1)	-	-	-1
2T	1	100	-2	-1	-100

NOW ELEMINATE S2 BY A2

NA.	X1	S2	P1	D2	VAL
D1	2	-	-1	-1	-99
P2	-	-	-	1	100
S1	1	-	-	-	2
A2	1	-1	-	-	1
2T	-99	100	-2	-1	-200

RE-ORDER TO STANDARD FORM

NA.	X1	D2	P1	S2	VAL
D1	2	-1	-1	-	-99
A2	(1)	-	-	-1	1
S1	1	-	-	-	2
P2	-	1	-	-	100
2T	-99	-1	-2	100	-200

D1 IS BADNAME, ELEM. A2 BY X1

NA.	A2	D2	P1	S2	VAL
D1	-2	(-1)	-1	2	-101
X1	1	-	-	-1	1
S1	-1	-	-	1	1
P2	-	1	-	-	100
2T	99	-1	-2	1	-101

CONDENSED TABLEAUX

NA.	X1	P1	P2	VAL	DIS
D1	2	-1	1	1	X
S1	1	-	-	2	X
S2	-1	-	-	-1	X
2T	1	-2	1	-	X
UB	100	100	(100)	X	X

. NO EQUIVALENT .
. OF THESE .
. TABLEAUX .
. ON ACCOUNT .
. OF THE .
. STANDARD FORM .
. DOUBLE STEP .

VECTOR-UPDATES ONLY

NA.	X1	P1	Z2	VAL	DIS
D1	2	-1	-1	-99	X
S1	1	-	-	2	X
A2	(1)	-	-	1	X
2T	-99	-2	-1	-200	X
UB	100	100	X	X	X

DITTO

NA.	A2	P1	Z2	VAL	DIS
D1	-2	-1	(-1)	-101	X
S1	-1	-	-	1	X
X1	1	-	-	1	99
2T	99	-2	-1	-101	X
UB	X	100	X	X	X

NOW ELEMINATE D1 BY D2

NA.	A2	D1	P1	S2	VAL
D2	2	-1	1	-2	101
X1	1	-	-	-1	1
S1	-1	-	-	1	1
P2	-2	1	-1	2	-1
2T	101	-1	-1	-1	-

D1 BY Z2, RE-FORM P2 AND S2

NA.	S2	P1	D1	VAL	DIS
P2	2	-1	1	-1	101
S1	1	-	-	1	X
X1	-1	-	-	1	99
2T	-1	-1	-1	-	X
UB	X	100	X	X	X

RE-ORDER AGAIN TO STANDARD F.

NA.	D1	A2	P1	S2	VAL
X1	-	1	-	-1	1
D2	-1	2	1	-2	101
S1	-	-1	-	(1)	1
P2	1	-2	-1	2	-1
2T	-1	101	-1	-1	-

RE-ORDER ALSO (NOT THE SAME)

NA.	D1	P1	S2	VAL	DIS
X1	-	-	-1	1	99
S1	-	-	(1)	1	X
P2	1	-1	2	-1	101
2T	-1	-1	-1	-	X
UB	X	100	X	X	X

P2 IS BADNAME, ELEM. S1 BY S2

NA.	D1	A2	P1	S1	VAL
X1	-	-	-	1	2
D2	-1	-	1	2	103
S2	-	-1	-	1	1
P2	1	-	(-1)	-2	-3
2T	-1	100	-1	1	1

DITTO, ELEMINATE S1 BY S2

NA.	D1	P1	S1	VAL	DIS
X1	-	-	1	2	98
S2	-	-	1	1	X
P2	1	(-1)	-2	-3	103
2T	-1	-1	1	1	X
UB	X	100	X	X	X

NOW ELEMINATE P2 BY P1

NA.	D1	A2	P2	S1	VAL
X1	-	-	-	1	2
D2	-	-	1	-	100
S2	-	-1	-	1	1
P1	-1	-	-1	2	3
2T	-2	100	-1	3	4

DITTO, ELEMINATE P2 BY P1

NA.	D1	P2	S1	VAL	DIS
X1	-	-	1	2	98
S2	-	-	1	1	X
P1	-1	-1	2	3	97
2T	-2	-1	3	4	X
UB	X	100	X	X	X

RE-ORDER TO STANDARD FORM

NA.	D1	A2	S1	P2	VAL
X1	-	-	1	-	2
D2	-	-	-	1	100
P1	-1	-	2	-1	3
S2	-	-1	1	-	1
2T	-2	100	3	-1	4

RE-ORDER ALSO (NOT THE SAME)

NA.	D1	S1	P2	VAL	DIS
X1	-	1	-	2	98
P1	-1	2	-1	3	97
S2	-	1	-	1	X
2T	-2	3	-1	4	X
UB	X	X	100	X	X

WHICH IS THE OPTIMUM

To contain the size of the explicit tableau to some extent, only the active upper limit on p_2 is included in the tableaux.

The following notes arise in connection with the tabulation 16.10d.

First of all, not all explicit tableaux have obvious condensed equivalents. If a row and and column are "flicked over", this is computationally most easily handled in one operation, but in terms of solution-interpretation, there are separate steps. If two steps of this type form a complementary pair linking two standard form tableaux, we speak of a standard form double step.

One further computational detail is useful to mention at this point.

Upper limits and upper limit distances have been marked with a cross X indicating "not applicable" where they are ignored by the search operations. In fact figures are entered in these cells and they are used when rebuilding the corresponding "normal" variable. For example, z_2 is the distance between p_2 and its upper limit. As z_2 enters the basis under circumstances where the non-negativity of p_2 is not protected at that stage of the algorithm, z_2 has no upper limit, but the value of p_2 is in fact stored in the upper limit cell of the z_2 vector. The upper limit facilities solve the problem of boundedness, so to say "by brute force".

With such strong restrictions in the specification of the problem, we have little difficulty in proving that all incoming variable columns are bounded. This proof is as follows:

If the incoming variable is a primal variable: obvious. Even if the incoming variable itself is a slack-variable which has no upper limit, a change in a primal slack-variable requires a change in at least one specified primal variable x_j, which is driven either towards its upper limit, or towards its lower limit of zero.

If the badname-variable is a primal variable, there is a non-positive entry in the driving variable row/incoming variable column cell, as well as in the badname-variable row/incoming variable column cell, at least one of them being negative non-zero. Therefore, either the badname-variable is driven towards zero and can be eliminated at that point, or the driving variable is driven against its upper limit.

Having proved boundedness of primal variables as well as of all variables in the case of a primal badname-variable, we still

need to prove boundedness of a dual incoming variable in the presence of a dual badname-variable. For this we refer to the proof supplied for the case of a feasible solution-vector, as the same argument (contradiction of the symmetry property is also applicable in the general case of a dual badname-variable and a dual incoming variable.

End of proof summary

Note that the above proof implies that the algorithm will always converge to the development of an end-of-algorithm standard-form tableau with a non-negative (primal and dual) solution vector.

If the meaningful problem is empty, this will therefore become apparent because artificial variables are left in the end-of-algorithm basis. If the meaningful problem is unbounded, this will become apparent because artificial fancy-high upper limits on primal variables are binding on the optimal solution.

That a binding artificial upper limit on a primal variable implies unboundedness of the meaningful problem is obvious. The relation between an artificially feasible end-of-algorithm solution and emptiness of the meaningful problem needs, however, further investigation.

We partition A in two block-rows. One block-row refers to the satisfied restrictions, the other to the ones in which artificial variables remained in the basis; the vectors $\underline{b}$ and $\underline{p}$ are partitioned accordingly.

If an artificially feasible solution is developed the optimality conditions as satisfied by the artificially feasible solution are

$$\underline{d} = A_1' \underline{p}_1 + A_2' \underline{p}_2 - D\underline{x}_1 - \underline{w}_1 \geq 0 \qquad (16.10.1)$$

the optimality-conditions on the meaningful variables (with only those elements of $\underline{d}$ which refer to non-basic variables being in fact non-zero),

and

$$0 = -\underline{p}_2 + \underline{f}_2 \qquad (16.10.2)$$

the optimality conditions on the active artificial variables, $\underline{f}_2$ being the fancy-high upper limits on p_2.

If the meaningful problem was indeed empty, the primal solution would be equal to the primal solution of the following problem:

Maximise $-\underline{f}_2' A_2 \underline{x}$

Subject to $A_1 \underline{x} \leq \underline{b}_1$

implying inter alia a solution in the corner of the feasible space area.

The dual requirements of this problem are that for some $\underline{p}^{**} \geq 0$,

$$-A_2' \underline{f}_2 - A_1' \underline{p}^{**} \leq 0 \qquad (16.10.3)$$

We would like to assume that (16.10.1) could be obtained as the summation of

$$\underline{d}^* = A_1' \underline{p}_1^* - D\underline{x}_1 - \underline{w}_1 \qquad (16.10.4)$$

and

$$\underline{d}^{**} = A_1' \underline{p}_1^{**} + A_2' \underline{p}_2 \qquad (16.10.5)$$

where $\underline{d}^*$, $\underline{p}_1^*$ is the dual solution associated with the (block triangular) block-pivot which emerges if the violated restrictions are removed from the problem (not normally an optimal solution, i.e. not $\underline{d}^* \geq 0$, $p_1^* \geq 0$) and $\underline{d}^{**}$ are the slack-variables in (16.10.3), $\underline{d}^{**} \geq 0$.

If the dual variables are of a different (greater) order of magnitude than the primal variables, it is safe to assume that (16.10.1) is in practice equivalent to (16.10.5).

To allow this condition of dominance of (16.10.1) by the dual variables' terms $\underline{d}$, $A_1' \underline{p}$, and $A_2' \underline{p}$ to be met, we must ensure that incoming variables which are activated before all dual badnames have been dealt with are not only technically bounded, but also bounded even without the artificial upper limits on primal variables. Otherwise we may after all develop an artificially feasible "optimum" for a non-empty problem.

Example

Maximise $x_1 + x_2 - x_1^2 + x_2^2$

$$x_1 \geq x_2 + 2, \quad x_1, x_2 \geq 0$$

We put the artificial limits all at 100

$$(x_1, x_2 \leq 100, \quad p_1 \leq 100).$$

If this problem is solved by selecting badname-variables in the (inappropriate) order, d_2, d_1, s_1, we develop the series of tableaux as summarised in the tabulation 16.10e.

TABLEAU 16.10 E

IN A NON-CONVEX PROBLEM WITHOUT A FEASIBLE STARTING SOLUTION, INAPPROPRIATE BADNAME-SELECTION LEADS TO AN ARTIFICIALLY FEASIBLE SOLUTION WITH A HIGH POSITIVE SOLUTION VALUE.

NAME	!!	X 1	X 2	P 1	!!	VALUE	DIST
D 1	!!	-2	-	1	!!	-1	X
D 2	!!	-	2	-1	!!	-1	X
S 1	!!	-1	1	-	!!	-2	X
2T	!!	-1	-1	2	!!	-	X
BOUND	!!	100	(100)	100	!!	X	X

NAME	!!	X 1	B 2	P 1	!!	VALUE	DIST
D 1	!!	(-2)	-	1	!!	-1	X
U 2	!!	-	2	1	!!	201	X
S 1	!!	-1	-1	-	!!	-102	X
2T	!!	-1	201	102	!!	20200	X
BOUND	!!	100	100	100	!!	X	X

NAME	!!	D 1	B 2	P 1	!!	VALUE	DIST
X 1	!!	-0.50	-	-0.50	!!	0.50	99.50
U 2	!!	-	2	1	!!	201	X
S 1	!!	-0.50	-1	-0.50	!!	-101.50	X
2T	!!	-0.50	201	101.50	!!	20200.50	X
BOUND	!!	100	100	(100)	!!	X	X

NAME	!!	D 1	B 2	Z 1	!!	VALUE	DIST
X 1	!!	-0.50	-	0.50	!!	50.50	49.50
U 2	!!	-	2	-1	!!	101	X
A 1	!!	0.50	1	-0.50	!!	51.50	X
2T	!!	50.50	-101	51.50	!!	4900.50	X
BOUND	!!	100	100	100	!!	X	X

N.B.
Readers who might experience difficulty in following the tableau-manipulation in this example, may wish to work through the corresponding explicitly written problem, for which the set-up tableau is given here, with suppression of the never-active upper limit on x_1.

TABLEAU 16.10 F

THE PROBLEM GIVEN IN TABLEAU 16.10 E, WITH AN ARTIFICIAL VARIABLE FOR RESTRICTION ONE, AND AN EXPLICIT UPPER LIMIT ON X2.

NAME	X1	X2	A1	P1	U2	VALUE
D1	-2	-	-	1	-	-1
D2	-	2	-	-1	-1	-1
Z1	-	-	-	1	-	100
S1	-1	1	-1	-	-	-2
B2	-	1	-	-	-	100
2T	-1	-1	100	2	-100	-

(Note that the example contains two standard form doublt steps; one for x_2 reaching its upper limit, and one for p_1 reaching its upper limit. When the tableaux are written in full, these double steps obviously become two separate steps. Furthermore, in the absence of the actual unity-pivots, step-markings for upper limits have been made on the upperbounds row.)

The true optimal and feasible solution of this problem is, however, $x_1 = 2$, $x_2 = 0$, and if either d_1 or s_1 is selected as the first bad name-variable, that solution is found as the end-of-algorithm solution.

To ensure "meaningful boundedness", i.e. boundedness even in the absence of (artificial) upper limits of all incoming variables activated by a dual badname-variable, it is sufficient that the standard-form tableau in which the badname-variable is selected contains a negative non-zero cell in the diagonal badname-variable row/incoming variable column cell.

Theorem (Convex primal boundedness theorem)

If the badname-variable is a dual variable (and therefore the driving variable is a primal variable) and the driving variable displays (the strict convexity property of) a negative non-zero diagonal cell in the standard-form tableau (in which the badname-variable is selected)
then all incoming variables are bounded by the badname-variable

as leaving variable.

Proof

The condition is equivalent to strict convexity in an $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace, which may be developed by direct exchange of the driving variable against the badname-variable. The tableau which is developed by that step, is either the next actual standard form tableau (in which case no further proof is needed), or it is the larger subspace predecessor tableau of the first non-standard-form tableau developed.

The existence of, and the convexity properties of some other standard form neighbours of the current tableau is now shown by the following summary of the relationship between the current tableau and its larger subspace predecessor.

Non-standard form tableau

	last leaving v.	incoming v.
driving v.	a	1/b + ac
badname v.	(b)	bc

Larger subspace predecessor

	badname v.	income v.
driving v.	-a/b	1/b
last leaving v.	1/b	c

We begin with denoting the figures in the last leaving variables column of the current non-standard form tableau as a and b, as marked.

By assumption, the larger subspace predecessor exists, and describes a solution which is (strictly) convex in the $A_{11}\,\underline{x}_1 = \underline{b}_1$ subspace.

For b=0 the theorem follows immediately from the properties of the non-standard form block as discussed in sections 16.7 and 16.8, we now assume b > 0.

Upon developing the larger subspace successor tableau we calculate the cells marked -a/b and 1/b in the badname variable column. The badname-variable is a dual variable containing a non-zero entry in a primal variable's row (1/b in the last leaving variable's row), therefore the diagonal cell is negative non-zero.

$$-a/b < 0,$$

therefore, since b > 0, a > 0.

Similarly, the current incoming variable is a dual variable containing a non-zero entry in a primal variable's row (1/b in the driving variable's row). Therefore (denoting the diagonal cell as c)

$$c < 0.$$

The other cells in the current tableau are now calculated backwards. The results obtained so far, a > 0 and c < 0 imply bc < 0, showing that the initial assumptions (convexity in the $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, b > 0) imply:

(1) the existence and convexity of the larger subspace successor tableau - thus permitting recursive application of the same argument - and

(2) boundedness of the incoming variable's column by the badname-variable as leaving variable.

q.e.d.

Furthermore, since the similar property a > 0 is valid for the successor tableau, we must also assume

$$1/b + ac < 0$$

i.e.

all four standard form neighbours exist.

In other words, a negative non-zero cell on the main diagonal in the initially selected badname-row/driving variable column cell ensures non-standard form blocks which contain four non-zero elements.

Exercise 16.10

Solve the LP problem given in the example in section 9.5 (artificial variables in LP), as a quadratic programming problem. Do this first with two explicitly written artificial variables, and subsequently with artificial upper limits on the dual variables, using the tableau-presentation discussed in this section.

16.11 The treatment of infeasible starting solutions

In this section, we outline, in effect, three alternative versions of the QP algorithm. We call them the convex mode of operation, the non-convex mode of operation, and the amended convex mode of operation.

The convex mode of operation relies on the selection of badname variables, which can be either primal or dual variables and uses the complementarily rule to drive them out. The non-convex mode of operation is characterized by two separate stages of calculation, the initial Phase I being primarily devoted to finding a feasible solution, by means of complementary pairs of steps. The third version of the algorithm is to some extent a compromise between these two, and will therefore be discussed last. We first outline the convex and the non-convex modes of operation.

In the convex mode of operation, we proceed as follows

1) Select as badname-variable a basic variable which is associated with a negative entry in the value column and preferably also a negative entry in the diagonal cell (thereby avoiding the activation of artificial upper limits as far as possible). The criterion function suggested here is the product of the entry in the valule column (if negative) multiplied by the sum of the diagonal cell and a small negative number. The function chosen was in fact:

   $$T[I, N+1] * (T[I,I] - 0.000001),$$

   For a violated primal restriction this function is of necessity positive, a manifestly non-convex driving variable is therefore always a primal variable (associated with a dual badname-variable), and such a choice is made only if the solution is primal feasible.

2) Bring in the driving variable as incoming variable.

3) In standard form tableaux and non-standard form tableaux alike select the leaving variable as follows:
If the badname-variable is a dual variable apply the rule of the smallest quotient over the badname variable row, and primal variable's rows. (Accept a negative pivot only in the badname variable row, fly through other violated restrictions.)

If the badname variable is a primal variable then dual variables also qualify as leaving variables and are included in the search for the smallest quotient, subject to the driving variable increment protection qualifier.

4) If no negative entries in the value column are left, the problem is solved, otherwise return to 1.

This is a convergent procedure. For each selected primal badname-variable at least one violated primal restriction becomes satisfied in a finite number of steps, (either artificially or also meaningfully); once only dual badname variables are left the value of the objective function increases by a finite amount in every two steps made. Its main drawback is, however, that an artificially feasible end of algorithm standard form tableau, is not necessarily evidence of emptiness.

In the non-convex mode of operation, there are two distinct phases, "Phase I" and "Phase II", where Phase I concerns finding a feasible solution, just as in LP, Phase II the search for the optimum, a feasible solution having been found already. An essential characteristic of Phase I is that, - just as in LP - only the corners of the feasible space area are investigated. This means that standard form tableaux maintain the characteristic block triangular structure with zero entries in all primal variable row/dual variable column cells throughout Phase I.

We proceed as follows:

Phase I

1) Select as badname-variable a primal variable which is associated with a (the most) negative entry in the value column. If no negative-valued slack-variable is found, proceed to Phase II.

2) Select an incoming variable column associated with a negative entry in badname row/income variable column cell (if none is found this proves the problem to be an empty one).

The recommended selection criterion within this category of incoming variables is as follows:

First, preferred columns are those which are also associated with a negative valued dual slack-variable, and among those the one with the largest absolute value of the sum of the two negative coefficients, the one in the s_i row/x_j column cell and the one in the d_j row/value column cell. In the absence of preferred incoming variable columns choose the incoming variable by the criterion of the dual ratio. In a QP tableau in standard form, the search operation for the smallest dual ratio may be performed as a search for the smallest critical ratio with the driving variable column as tentative incoming variable column.

The primal variable associated with the dual variable tentatively indicated as leaving variable, is then the incoming variable.

Disregard the driving variable increment qualifier (illustration example to be given below). Do not however, bring in the driving variable as incoming variable at this stage, we need a primal incoming variable.

3) Select a primal variable as leaving variable, by applying the rule of the smallest quotient. The eligible non-negative quotients relate to two categories of primal variables viz:

   a) Non negative valued primal variables, whose non-negativity needs to be protected.

   b) The (negative valued) badname variable, but only if one of the two following conditions is satisfied. If no pivot of satisfactory absolute value has been found in category a), or if a higher solution value is attained by selecting the badname variable instead. (See also the discussion on "flying through" violated restrictions in an LP problem in Chapter IX.

   Dual variables do not at this point qualify as leaving variables. Since the badname row/incoming variable column cell contains a negative non-zero cell, the issue of unboundedness does not arise here.

4) Make the step

5) In the resulting non-standard form tableau select (disregarding the value column), as incoming variable the complement of the incoming variable selected under 3, as leaving variable the complement of the leaving variable

selected under 3, thus completing the step made under 4 as a complementary pair.

6) Make the step selected under 5
This step does not cause any change in any primal variable's entry in the value column all primal variable's row/incoming variable column cells containing zero entries.

On account of the block triangular structure of the block-pivot, the leaving variable row/next incoming variable column cell of the standard form tableau in which the search operations under 2 and 3 were performed, contains a zero entry. The figures in the next incoming variable column are therefore unchanged between the standard form and the non-standard form tableau.

7) If the leaving variable selected under 3 was the badname-variable, or if the badname-variable has ceased to be negative-valued (by flying through" its non-negativity), return to 1, otherwise to 2.

Phase II:

8) Proceed as in the convex mode.

Note that in this version of the algorithm, dual variables are only accepted as leaving variables, if they are either: the complement of a previously introduced primal variable (in the second step of a pair made under 6 in Phase I), or: the badname-variable (in Phase II).

It follows that the only way in which we may develop a solution-vector in which the number of basic variables x_j exceeds the number of eliminated slack variables s_i is by direct exchange of a primal driving variable x_j or s_i against the badname variable d_j or p_i, a situation which can only arise in Phase II.

The convex mode of operation as outlined above is basically a formalization of an algorithm which has been discussed and illustrated extensively in the previous section, but we now give an illustration of the non-convex mode of operation.

Example

Maximise $\tau = x_1 + x_2 - x_1^2 + x_2^2$

Subject to $x_1 \geq x_2 + 2;\ 2x_2 \geq x_1 \qquad (x_1, x_2 \geq 0)$

The set-up tableau of this problem is given below as tableau 16.11a.

TABLEAU 16.11 A

ILLUSTRATION-PROBLEM CONCERNING
THE NON-CONVEX MODE OF OPERATION.

NAME	!!	X1	X2	P1	P2	!!	VALUE
D1	!!	-2	-	1	-1	!!	-1
D2	!!	-	2	-1	2	!!	-1
S1	!!	-1	1	-	-	!!	-2
S2	!!	(1)	-2	-	-	!!	-
2T	!!	-1	-1	2	-	!!	-

We now proceed as follows:

Phase I

1 Choose s_1 as badname-variable. (in the convex mode: d_1)

2 Choose x_1 as incoming variable (preferred)

3 Choose s_2 as leaving variable (the pivot is actually the same as in the convex mode with d_1 as badname-variable.

4 Make the step, to develop tableau 16.11b

(The tableau is at this point also re-ordered. This re-ordering is organized in such a way as to permit return to a correctly ordered standard form tableau, by the second step of the pair, without further re-ordering.)

TABLEAU 16.11 B

NON-STANDARD FORM TABLEAU, FOLLOWING
THE FIRST (PRIMAL) STEP OF A COMPLEMEN-
TARY PAIR.

NAME	!!	P2	X2	P1	S2	!!	VALUE
X1	!!	-	-2	-	1	!!	-
D2	!!	2	2	-1	-	!!	-1
S1	!!	-	-1	-	1	!!	-2
D1	!!	(-1)	-4	1	2	!!	-1
2T	!!	-	-3	2	1	!!	-

5 To get back to standard form, we echange p_2 against d_1 as leaving variable (step still the same as in the convex mode).

6 Develop the next standard form tableau, which is tableau 16.11c.

TABLEAU 16.11 C

STANDARD FORM TABLEAU, S1 STILL IS THE BADNAME-VARIABLE.

NAME	!!	D1	X2	P1	S2	!!	VALUE
X1	!!	-	-2	-	1	!!	-
D2	!!	2	-6	1	4	!!	-3
S1	!!	-	(-1)	-	1	!!	-2
P2	!!	-1	4	-1	-2	!!	1
2T	!!	-	-3	2	1	!!	-

7 s_1 is still badname variable and has not so far been eliminated therefore return to 2 (in the convex mode d_2 would now become the badname-variable)

2 Select x_2 as incoming variable (preferred)(in the convex mode x_2 would be the driving variable, hence also the incoming variable).

3 Select s_1, the badname variable, as leaving variable, this being the only primal variable which gives rise to a quotient of the appropriate sign. (in the convex mode d_2 would become the leaving variable).

4 Develop the non-standard form tableau 16.11d

TABLEAU 16.11 D

NON-STANDARD FORM TABLEAU, FOLLOWING THE FIRST (PRIMAL) STEP OF THE SECOND COMPLEMENTARY PAIR.

NAME	!!	D1	P1	S 1	S2	!!	VALUE
X1	!!	-	-	-2	-1	!!	4
X2	!!	-	-	-1	-1	!!	2
D2	!!	2	(1)	-6	-2	!!	9
P2	!!	-1	-1	4	2	!!	-7
2T	!!	-	2	-3	-2	!!	6

5 Select as incoming variable p_1 and as leaving variable d_2

6 Develop the tableau 16.11e

TABLEAU 16.11 E

A FEASIBLE (AND OPTIMAL) TABLEAU

NAME	!!	D1	D2	S 1	S2	!!	VALUE
X1	!!	-	-	-2	-1	!!	4
X2	!!	-	-	-1	-1	!!	2
P1	!!	2	1	-6	-2	!!	9
P2	!!	1	1	-2	-	!!	2
2T	!!	-4	-2	9	2	!!	-12

which is the optimum tableau.

At this point it is useful to summarize the relative(dis)-advantages of the two methods, in relation to certain groups of problems. The various cases are first summarized as follows:

	convex problem	non-convex problem
The trivial basis is primal feasible (not dual feasible)	makes no difference the two methods become identical in that case	
The trivial basis is dual feasible (not primal feasible)	Use the convex mode; optimal form is maintained throughout	? ?? ? ? ? The convex mode may get there quicker, but the non-convex mode is safer
The trivial basis is neither primal feasible nor dual feasible	If many dual restrictions are violated, use the non-convex mode: it can seek out preferred columns	Use the non-convex mode - no problem of spurious emptiness -

In a non-convex problem with a dual feasible trivial basis (a "minimization" problem), there may, or there may not be a trade off between computational efficiency in the well-behaved case, and proof supported effectiveness.

The problem of "running away" in the artificially feasible area may arise despite the initial optimal form, because satisfied dual restrictions may be flown through in the wrong direction first, on account of the driving variable increment qualifier and then activate directions in which the quadratic component of the objective function comes to dominate over the penalty coefficient which are supposed to drive the artificial variables out.

In addition to this the non-convex mode can also be more effective as an optimizing algorithm than the convex mode. In the convex mode, the value of the objective function consistently drops when a violated primal restriction figures as badname. The non-convex mode, with its L.P. style Phase I on the other hand, can in fact gain solution value during Phase I.

Even if a drop in the value of the objective function is unavoidable, the non-convex mode has greater freedom in selecting columns and the criterion of the dual ratio does to some extent contain the drop in the solution value.

The amended convex mode is an attempt to improve on the convex mode as an efficient procedure to solve problems in which the trivial basis violates primal as well as dual requirements. It does so, by preferably selecting as badname variables shadow-prices of variables which would qualify as "preferred" incoming variables in Phase I of the non-convex mode, in preference to other dual variables.

Example

Maximise $\tau = x_1 - x_2 + 2x_3 - x_1^2 - 2x_2^2 - x_3^2$

Subject to $-x_1 - x_2 + x_3 \leq -2$

$x_1 - 2x_3 \leq 1$

$(x_1, x_2, x_3 \geq 0)$

For this example we only give the set up tableau, and arguments for choosing a particular first pivot.

If s_1 were selected as badname variable in the convex mode, we would be forced to introduce x_2 into the basis as the first primal incoming variable, despite the fact that x_2 leads to a reduction in the value of the objective function.

In fact s_1 is selected as badname variable only in the non-convex mode. In the convex mode shadow prices of the strictly convex variables x_1 and x_3 both have preferences over s_1.

TABLEAU 16.11 F

PIVOT-SELECTION UNDER THE AMENDED CONVEX MODE.

NAME	!!	X1	X2	X3	P1	P2	!!	VALUE
D1	!!	(-2)	-	-	1	-1	!!	-1
D2	!!	-	-4	-	1	-	!!	1
D3	!!	-	-	-2	-1	2	!!	-2
S1	!!	-1	-1	1	-	-	!!	-2
S2	!!	1	-	-2	-	-	!!	1
2T	!!	-1	1	-2	2	-1	!!	-

In the convex mode as it was outlined above, d_3 would be chosen. This is a pity because x_1 leads to an increase in the objective function and to a less negative value of s_1 at the same time. The non-convex mode will seek it out for that reason. The convex mode as amended does the same by selecting d_1 as badname variable, in preference to the more negative valued d_3 variable. This is done by multiplying the criterion-function for selecting the badname variable with another load-factor, which is in fact 0.01 plus the sum of the absolute values of all the negative entries in the intersections of violated restriction rows and the driving variable column i.e. in the example at hand 1.01 for d_1 and 0.01 for d_3 (and 0.01 for directly tackling s_1 as badname variable).

The subject of this section, the treatment of infeasible starting solutions is a major point of distinction between the QP algorithms offered in this book, and their closest relatives as previously developed by other authors. Having covered this point, it is therefore appropriate to briefly review these related QP algorithms, in particular Cottle's principal pivoting method and the a-symmetric version of the Van de Panne and Whinston method.

The symmetric variant of Van de Panne's method will be discussed in the next chapter, whilst other QP algorithms, although existing*, differ so substantially from the algorithms offered here that a more extensive discussion would be needed, and we shall refrain from such a more extensive summary.

*This refers in particular to Beale's [3] method.

Cottle's algorithm is most akin to what has been described here as the convex mode of operation. Its convergence rests on regularly reducing the number of violated restrictions whether they are primal or dual restrictions. For this reason it has been suggested (by Van de Panne, Linear and Quadratic Programming p 323) that this method should be more appropriately called the index-method, rather than the principal pivoting method, as Cottle names this method. It is designed for convex problems only, and it applies the rule of the smallest quotient in the more strict sense of protecting the non-negativity of both primal and dual variables at all stages of calculation. Basically it is the algorithm which we began to outline in section 16.3.

The driving variable increment qualifier - and hence the possibility to apply the convex mode of operation to non-convex problems is, as far as I know, novel to this book. If the original version of Cottle's method is applied to a problem where the objective function is not properly convex, it may break down in cycling, the convex mode of operation can be shown to converge in all cases even where the end of algorithm tableau may represent (in the non-convex case) a local rather than the global maximum and even possibly a local maximum in an artificially feasible region giving rise to a spurious indication of emptiness.

The two-phase method on the other hand is directly due to Van de Panne. Indeed the non-convex mode of operation comes near to being an implementation of the Van de Panne and Whinston method.

The one difference is that, for similar reasons as advocated in Chapter IX for the L.P. case, there is no complete separation between the two phases i.e. optimality is considered already during Phase I, giving rise to the notion of "preferred" columns, to the possibility to "fly through" violated primal restrictions, and to the use of the dual ratio as a selection-criterion for choosing between non-preferred columns. Van de Panne (Linear and Quadratic Programming p 279) recommends the minimization of the sum of all artificial variables by an LP algorithm as Phase I for the QP problem, a procedure which has the same limitations as using artificial variables in the LP case.

16.12 Ordering of the quadratic programming tableau

Recall the symmetry properties of the quadratic programming tableau, especially when in standard form (see section 16.4). For a number of reasons, it is desirable to keep these symmetry properties apparent in the tableaux developed by the algorithm.

Firstly, a symmetrically ordered tableau is the most suitable way of presenting the end result. Writing a programme-code is also simplified, if it is known where the rows and columns of the tableau may be found. Unfortunately, simplex steps tend to bring the tableau in a random non-ordering, in quadratic as well as in linear programming.

This problem was solved, for linear programming (see sections 12.2 and 12.3), by re-ordering the tableau afterwards.

With quadratic programming it is more practical to maintain ordering as far as possible at the time of making the steps.

This may be done, by the following two devices, to be applied at each step, except when the tableau is coming back to standard form.

a) Interchange the pivot column with the "alternate" of the pivot row. This alternate is the next pivot column.

b) Interchange the pivot row with the "badname" row.

Application of these re-ordering rules may be illustrated by writing an example, as follows:

Maximise $\tau = x_1 - 5x_2 + x_1^2 - x_2^2$

Subject to $x_1 + x_2 \geq 2$

$x_1 + x_2 \leq 10$

$(x_1, x_2 \geq 0)$

TABLEAU 16.12 A

SET-UP TABLEAU OF THE EXAMPLE ON ORDERING.

NAME				X1	X2	P1	P2		VALUE
	CODE			1	2	-1001	-1002		
D1	-1		2	-	1	-1			-1
D2	-2		-	-2	①	-1			5
S1	1001		-1	-1	-	-			-2
S2	1002		1	1	-	-			10
2T			-1	5	2	-10			-

The set-up tableau of this problem is given in tableau 16.12a and we now proceed as follows:

Re-order, to maintain ordering in the next tableau

a) Interchange the p_1 column (pivot column) with the x_2 column, x_2 being the complementary variable to the pivot-row variable d_2.

b) Interchange the d_2-row (pivot) with the s_1 (badname) row.

The resulting tableau has been named 16.12b.

TABLEAU 16.12 B

INITIAL TABLEAU, RE-ORDERERED FOR THE FIRST STEP.

NAME			X1	P1	X 2	P2		VALUE
	CODE		1	-1001	2	-1002		
D1	-1		2	1	-	-1		-1
S1	1001		-1	-	-1	-		-2
D2	-2		-	(1)	-2	-1		5
S2	1002		1	-	1	-		10
2T			-1	2	5	-10		-

This re-ordering ensures that the new basic variable (the pivot column) and the just eliminated pivot-row variable (which may stay a non-basic variable) are stored in their "natural" positions.

TABLEAU 16.12 C

NON-STANDARD FORM TABLEAU, WITH ONLY THE 'WRONG' NAMES OUT OF ORDERING.

NAME			X1	D2	X2	P2		VALUE
	CODE		1	-2	2	-1002		
D1	-1		2	-1	2	-		-6
S1	1001		-1	-	(-1)	-		-2
P1	-1001		-	1	-2	-1		5
S2	1002		1	-	1	-		10
2T			-1	-2	9	-8		-10

The d_2 column in the new tableau comes on place 2, and only the x_2 column, which is to become pivot column immediately afterwards is in a "wrong" place. the p_1 row in the new tableau comes in place (minus) 1001, and only the badname-row (to be eliminated in due course) is in the "wrong" place.

The next tableau, numbered 16.12d, is in standard-form and no re-ordering is needed. Here we select d_1 as the badname-variable.

TABLEAU 16.12 D

STANDARD FORM TABLEAU, ORDERED ACCORDING TO CODES.

NAME	!		!!	X1	D2	S1	P2	!!	VALUE
	!	CODE	!!	1	-2	1001	-1002	!!	
D1	!	-1	!!	-	-1	2	-	!!	-10
X2	!	2	!!	(1)	-	-1	-	!!	2
P1	!	-1001	!!	2	1	-2	-1	!!	9
S2	!	1002	!!	-	-	1	-	!!	8
2T	!		!!	-10	-2	9	-8	!!	-28

Before actually making the next step, we again re-order, to maintain ordering in the next tableau, which is once more in non-standard form. To develop tableau 16.2e, we proceed as follows:

a) Interchange the pivotal x_1 - column with the (next pivotal) d_2 - column.

b) Interchange the pivotal x_2 - row with the d_1 badname - row.

TABLEAU 16.12 E

THE SAME TABLEAU, RE-ORDERED FOR THE NEXT STEP

NAME	!		!!	D2	X1	S1	P2	!!	VALUE
	!	CODE	!!	-2	-2	1001	-1002	!!	
X2	!	2	!!	-	(1)	-1	-	!!	2
D1	!	-1	!!	-1	-	2	-	!!	-10
P1	!	-1001	!!	1	2	-2	-1	!!	9
S2	!	1002	!!	-	-	1	-	!!	8
2T	!		!!	-2	-10	9	-8	!!	-28

As a result of the ordering of tableau 16.12e, the successor tableau only has the badname-row i.e. d_1 and the complement of the current pivot row i.e. d_2 out of ordering. After making the step, we develop tableau 16.12f.

TABLEAU 16.12 F

AGAIN, THE 'WRONG' NAMES ARE IN THE WRONG SLOTS.

NAME		!!	D2	X2	S1	P2	!!	VALUE
	CODE	!!	-2	2	1001	-1002	!!	
X1	1	!!	-	1	-1	-	!!	2
D1	-1	!!	-1	-	2	-	!!	-10
P1	-1001	!!	1	-2	-	-1	!!	5
S2	1002	!!	-	-	1	-	!!	8
2T		!!	-2	10	-1	-8	!!	-8

The next tableau will be in standard form, therefore no reordering before making the next step is needed. We develop tableau 16.12g.

TABLEAU 16.12 G

STANDARD FORM TABLEAU, ALREADY CORRECTLY ORDERED.

NAME		!!	D1	X2	S1	P2	!!	VALUE
	CODE	!!	-1	2	1001	-1002	!!	
X1	1	!!	-	1	-1	-	!!	2
D2	-2	!!	-1	-	-2	-	!!	10
P1	-1001	!!	1	-2	2	-1	!!	-5
S2	1002	!!	-	-	1	-	!!	8
2T		!!	-2	10	-5	-8	!!	12

This particular problem of manipulating the QP tableau may now be considered as having adequately illustrated and we will not pursue the example any further.

16.13 Equations and variables without non-negativity-restriction

There is a strong analogy between an equation and a variable without non-negativity requirement. The one is the dual problem of the other. A variable which is not subject to a non-negativity restriction, but is allowed to attain negative values, is brought into the list of basic variables with priority. It then stays a basic variable irrespective of its sign. Its associated dual

restriction therefore is an equation. Likewise the dual variable associated with an equation has to be brought into the list of basic variables, and then stays there. It does not become badname and is exempt from the search operation for a pivot row and is allowed to become negative instead.

In the simplex algorithm for quadratic programming which is the subject of this chapter, both problems are handled along the lines followed in section 10.2 for equations in linear programming.

The equation is presented with a negative constant, i.e. a violated restriction. The "conservative" rule for choosing the pivot-row is applied insofar as equation-slacks are concerned, i.e. they are not allowed to become positive. Once the equation is binding, the associated dual variable is or becomes a basic variable. The dual variable of an equation is initially brought into the vector of basic variables with a positive sign but is allowed to become negative later on.

Example

Maximise $1.4\ x_1 + 2\ x_2$

Subject to $x_1 + x_2 = 3$

$(x_1, x_2 \geq 0)$

We write the equation as $-x_1 - x_2 = -3$, rather than as $x_1 + x_2 = 3$, and, just as in the LP-case, we start with a set-up tableau which contains in fact a legative-valued slack-variable, i.e. $s_1 = -3$.

This set-up tableau is given as tableau 16.13a below.

TABLEAU 16.13 A

SET-UP TABLEAU WITH AN EQUATION, PRESENTED AS A VIOLATED INEQUALITY.

NAME			X 1	X 2	P 1			VALUE
	CODE		1	2	-1001			
D 1	-1		-	-	1		≤	-1.40
D 2	-2		-	-	1		≤	-2
S 1	1001		-1	-1	-		=	-3
2T			-1.40	-2	3			-

To facilitate following the ordering
tableau 16.13a and its successor tableaux are presented with numerical name-codes as well as alphanumerical names.

Note, that this is really an LP problem, which is solved by a QP algorithm.

The solution of this problem, by the non-convex mode of operation, is given below. We first exchange the (preferred) x_2-variable against the undesired s_1-slack, and develop tableau 16.13b.

TABLEAU 16.13 B

THE EQUATION IS NOW SATISFIED.

NAME			X 1	P 1	S 1			VALUE
	CODE		1	-1001	1001			
D 1	-1		-	1	-		≤	-1.40
X 2	2		1	-	-1		≤	3
D 2	-2		-	1	-		≤	-2
2T			0.60	3	-2			6

To get back to standard form, we complete the pair of steps, exchangingly p_1 against d_2. We develop tableau 16.13c.

TABLEAU 16.13 C

THE DUAL VARIABLE OF AN EQUATION MAY ATTAIN A NEGATIVE VALUE.

NAME			X 1	D 2	S 1			VALUE
	CODE		1	-2	1001			
D 1	-1		-	-1	-		≤	0.60
X 2	2		1	-	-1		≤	3
P 1	-1001		-	1	-		=	-2
2T			0.60	-3	-2			12

The exchange of one dual variable against another, irrespective of the non-negativity of either the incoming variable, or of any other dual variable, is a normal feature of the non-convex mode of operation.

However, since p_1 is the only negative-valued variable, tableau 16.13c represents the optimum. Dual variables of equations are not restricted in sign.

Note that this fact affects the possibility to interpret the corresponding row of the tableau as an inequality. For example, the d_1-row of tableau 16.13c can be read as

$$- d_2 \leq 0.60$$

The slack of this restriction is the d_1-variable. The similar reading of the p_1-row,

$$d_2 \leq -2$$

does not apply, as the "slack-variable" of this restriction, p_1, does not have to be non-negative, and no restriction on d_2 is given by the p_1-row.

The treatment of variables without sign restriction is on the same lines. "Free" variables, are initially presented with a violated dual restriction, i.e. as "desirable" variables. Once a free variable is in the list of basic variables it is exempt from search operations.

Example

Maximise $\tau = x_1 + x_2$

Subject to $2x_1 + x_2 \leq 10$

$x_1 \geq -2$

$x_1 \leq 1$

($x_2 \geq 0$, but x_1 not restricted in sign)

This too, is a linear programming problem, which we solve by the quadratic programming algorithm, in tableau 16.13d to 16.13h , as follows:

As in the case of equation-requirements on primal variables, writing the "-" sign in the d_1 and d_2 rows of tableau 16.13d initially, and indeed as long as d_1 and d_2 are basic variables, indicates an intention, not a met requirement, as is in this example the case with the s_1, s_2 and s_3-rows where the "$\leq$" sign figures.

TABLEAU 16.13 D

SET-UP TABLEAU WITH FREE VARIABLES,
PRESENTED AS 'DESIRABLE' VARIABLES.

NAME	!	X1	X2	P1	P2	P3	!		!	VALUE
D1	!	-	-	-2	1	-1	!	=	!	-1
D2	!	-	-	-1	-	-	!	=	!	-1
S1	!	2	1	-	-	-	!	≤	!	10
S2	!	-1	-	-	-	-	!	≤	!	2
S3	!	①	-	-	-	-	!	≤	!	1
2T	!	-1	-1	-10	-2	-1	!		!	-

In order to drive out the negative-valued d_1-variable, d_1 is chosen as the first badname-variable and x_1 becomes the driving variable. The rule of the smallest quotient then indicates s_1 as leaving variable.

The resulting non-standard form tableau is not reported explicitly here, instead we give the next successor tableau, which is again a standard form tableau, tableau 16.13e.

TABLEAU 16.13 E

X1 IS NOW A BASIC VARIABLE,
AND WILL STAY IN THE BASIS.

NAME	!	D1	X2	P1	P2	S3	!		!	VALUE
X1	!	-	-	-	-	1	!	≤	!	1
D2	!	-	-	-1	-	-	!	=	!	-1
S1	!	-	①	-	-	-2	!	≤	!	8
S2	!	-	-	-	-	1	!	≤	!	3
P3	!	-1	-	2	-1	-	!	≤	!	1
2T	!	-1	-1	-8	-3	1	!		!	2

In tableau 16.13e we repeat the same procedure, d_2 is chosen as badname, and x_2 is the next pivot column. The rule of the smallest (only) quotient indicates s_1 as pivot row.

The immediate successor tableau of tableau 16.13e again is a non-standard form tableau, which is not printed. The complementarity rule activates p_1 as incoming variable, and the

badname-variable d_2 can at that stage be eliminated. The then resulting standard form tableau is given as tableau 16.13f below.

TABLEAU 16.13 F

X2 IS NOW ALSO A BASIC VARIABLE,
AND WILL STAY IN THE BASIS AS WELL.

NAME	!	D1	D2	S1	P2	S3	!		!	VALUE
X1	!	-	-	-	-	1	!	≤	!	1
X2	!	-	-	1	-	-2	!	≤	!	8
P1	!	-	-1	-	-	-	!	≤	!	1
S2	!	-	-	-	-	①	!	≤	!	3
P3	!	-1	2	-	-1	-	!	≤	!	-1
2T	!	-1	-8	1	-3	-1	!		!	18

In tableau 16.13f we select p_3 as badname-variable, s_3 becomes the driving variable. The rule of the smallest quotient is now applied to the s_3-column, with exclusion of the quotient in the x_1-row, although that quotient is actually the smallest non-negative quotient.

The result of the choice of s_2 with a quotient of 3 and the rejection of x_1 as leaving variable is that x_1 now becomes negative in tableau 16.13g.

TABLEAU 16.13 G

X1 HAS BEEN ALLOWED TO BECOME NEGATIVE.

NAME	!	D1	D2	S1	S2	P2	!		!	VALUE
X1	!	-	-	-	-1	-	!	≤	!	-2
X2	!	-	-	1	2	-	!	≤	!	14
P1	!	-	-1	-	-	-	!	≤	!	1
P3	!	-1	2	-	-	(-1)	!	≤	!	-1
S3	!	-	-	-	1	-	!	≤	!	3
2T	!	-1	-8	1	1	-3	!		!	21

The negative entry for x_1 in the value column does not constitute a violation of any restriction, as the non-negativity requirement does not apply to x_1.

We continue with the next step. The alternative of the just eliminated s_2 - variable (p_2) becomes a pivot column. There is only one positive quotient and the badname - variable p_3 is eleminated.

TABLEAU 16.13 H

AN OPTIMAL AND FEASSIBLE SOLUTION,
DESPITE THE NEGATIVE VALUE OF X1.

NAME	!	D1	D2	S1	S2	P3	!		!	VALUE
X1	!	-	-	-	-1	-	!	≤	!	-2
X2	!	-	-	1	2	-	!	≤	!	14
P1	!	-	-1	-	-	-	!	≤	!	1
P2	!	1	-2	-	-	-1	!	≤	!	1
S3	!	-	-	-	1	-	!	≤	!	3
2T	!	2	-14	1	1	-3	!		!	24

The solution given by tableau 16.13h is the optimal and feasible solution, i.e. no badname is found among those variables to which the non-negativity requirement applies.

The one other problem to be noted in connection with mixed systems is that it may be necessary to "turn round" an equation (or to re-define the sign of a variable). This problem is illustrated here in relation to the elimination of an equation-slack in the convex mode - it does not arise in the same way in the non-convex mode.

Example: Maximise $4x_1 - x_1^2 + 4x_2 - x_2^2$

subject to $x_1 + x_2 = 1$

$(x_1, x_2 \geq 0)$

(see tableau 16.13i)

We select the shadowprice of the strictly convex variable x_1 as badname-variable. Under the rules of the convex mode of operation, the positive quotient -1/-1 in the s_1-row does not qualify as being included in the search for the smallest quotient. Column-updating by a negative pivot would generate a non-standard form tableau which does not obey the recognizeable

convexity properties discussed in Sections 16.6-16.8. (The d_1/p_1 cell is positive and does not permit the elimination of d_1 by p_1 while assigning a non-negative value to p_1.). There are two possible ways of handling this problem.

TABLEAU 16.13 I

SET-UP TABLEAU WITH AN EQUATION PRESENTED AS A SINGLE INEQUALITY.

NAME	!!	X1	X2	P1	!!	VALUE
D1	!!	-2	-	1	!!	-4
D2	!!	-	-2	1	!!	-4
S1	!!	-1	-1	-	!!	-1
2T	!!	-4	-4	1	!!	-

We may read the tableau as if the equation was written in both directions, e.g. although we actually write only one s_1-row, our tableau interpretation is as given in tableau 16.13j below.

TABLEAU 16.13 J

SET-UP TABLEAU WITH AN EQUATION PRESENTED AS A DOUBLE INEQUALITY.

NAME	!!	X1	X2	P11	P12	!!	VALUE
D1	!!	-2	-	1	-1	!!	-4
D2	!!	-	-2	1	-1	!!	-4
S11	!!	-1	-1	-	-	!!	-1
S12	!!	(1)	1	-	-	!!	1
2T	!!	-4	-4	1	-1	!!	-

The quotient for both presentations of the equation is obviously the same, as we put the equation in the $x_1 + x_2 \leq 1$ form, once the smallest quotient is found for that equation.

Upon selecting s_1 as leaving variable, the tableau is written as tableau 16.13k below.

TABLEAU 16.13 K

RE-INTERPTETED TABLEAU WITH AN EQUATION PRESENTED AS A SATISFIED RESTRICTION.

NAME	!!	X1	X2	P12	!!	VALUE
D1	!!	-2	-	-1	!!	-4
D2	!!	-	-2	-1	!!	-4
S12	!!	①	1	-	!!	1
2T	!!	-4	-4	-1	!!	-

and from that point onwards the algorithm works in the usual way.

The alternative possibility is to fly through an equation and turn it round afterwards. After exchanging x_1 against d_1, we initially develop tableau 16.13l, in which the equation appears as an amply fulfilled restrictions, but before selecting a badname in the next standard-form tableau, (in this case immediately), we change the presentation of the tableau by

TABLEAUX 16.13 L AND 16.13 M

EQUATION-PRESENTATION AFTER FLYING THROUGH

NA.	D1	X2	P11	VAL
X1	-0.5	-	-0.5	2
D2	-	-2	1	-4
S11	-0.5	-1	-0.5	1
2T	-2	-4	-1	8

EQUATION-PRESENTATION AFTER TURNING ROUND

NA.	D1	X2	P12	VAL
X1	-0.5	-	0.5	2
D2	-	-2	-1	-4
S12	0.5	1	-0.5	-1
2T	-2	-4	1	8

turning it round as in tableau 16.13m.

If the latter device is applied, s_1 will of necessity be selected at some stage as badname-variable.

The same choice between "flicking round" at the stage of pivot selection or at a later stage is also present for variables

without non-negativity restriction, at least when the convex mode of operation is applied only to convex problems. When the convex mode of operation is used on a non-convex problem the qualifier may cause dual restrictions to be flown through. There is also the complication that with a dual badname-variable the normal procedure is to fly through dual restrictions anyhow.

The code offered in Section 16.15 therefore treats the two problems in a different way, flicking round equations at the stage of pivot-selection and changing the presentation of variables without non-negativity restriction just before selecting a badname-variable. This is actually done for equations as well but that loop is activated only if an equation initially written in a form in which its constant is positive.

16.14 Lower bounds in quadratic programming

The device of preliminary adjustment of the set-up tableau, as discussed in Section 10.4 for the LP problem may also be applied to a quadratic programming problem. The one point where simply copying of the tableau-manipulation from Section 10.4 produces a result which is actually incorrect relates to the value of the objective function, a point to be discussed in more detail later in this section.

Example: Maximise $x_1 - x_1^2 + 2x_2 - x_2^2$

subject to $x_1 + x_2 \geq 3$

$5 \leq x_1 \leq 100,\ 2 \leq x_2 \leq 100$

Tableau 16.14a below gives the set-up tableaux for this problem, without the lower limits, and with the LP-style adjustment

The adjusted tableau is already in optimal form, confirming the fact that $x_1 = 5$, $x_2 = 2$, is the optimal and feasible solution of the stated problem. We need, however, to analyse the interpretation of the dual restrictions and the solution value in more detail.

We apply the same device as used in Section 10.3 and put

$$x_1 = y_1 + 5, \quad x_2 = y_2 + 2$$

TABLEAU 16.14 A

LP-TYPE LOWER-BOUNDS ADJUSTMENT IN QP

SET-UP TABLEAU WITHOUT THE LIMITS

NAME	X1	X2	P1	VALUE
D1	-2	-	1	-1
D2	-	-2	1	-2
S1	-1	-1	-	-3
2T	-1	-2	3	-
UB	100	100	100	X

ADJUSTED ACCORDING TO SECTION 10.4

NAME	Y1	Y2	P1	VALUE
D1	-2	-	1	9
D2	-	-2	1	2
S1	-1	-1	-	4
2T	-1	-2	3	9
UB	95	98	100	X
LB	5	2	0	X

Re-numbering the one initially specified explicit restriction as s_3, the set-up tableau now becomes the following:

TABLEAU 16.14 B

QP PROBLEM WITH EXPLICIT LOWER LIMITS.

NAME	X1	X2	Y1	Y2	P1	P2	P3	VALUE
D1	-2	-	-	-	1	-	1	-1
D2	-	-2	-	-	-	1	1	-2
D3	-	-	-	-	-1	-	-	-
D4	-	-	-	-	-	-1	-	-
S1	-1	-	1	-	-	-	-	-5
S2	-	-1	-	1	-	-	-	-2
S3	-1	-1	-	-	-	-	-	-3
2T	-1	-2	-	-	5	2	3	-

Note that the d_3 and d_4 restrictions simply say $-p_1 + d_3 = 0$ and $-p_2 + d_4 = 0$, i.e. d_3 (= the shadowprice of the non-negativity of y_1) is the same variable as p_1 (= the dual variable associated with the equation $x_1 = \bar{y}_1 + 5$). As a shadowprice of an equation p_1 is as such allowed to become negative but this will not occur on account of the non-negativity of d_3; the same argument applies to p_2 and d_4. Substitution of d_3 for p_1 and d_4 for p_2 into the d_1 and d_2 equations results in:

$$-2x_1 + d_3 + p_3 = -1 \text{ for } d_3 = p_1$$

and

$$-2x_2 + d_4 + p_3 = -2 \text{ for } d_4 = p_2$$

Furthermore, x_1 and x_2 themselves may be treated as variables which are free of non-negativity restrictions for the same reasons as already mentioned in Section 10.3 for the LP case. This turns the dual requirements associated with x_1 and x_2 into equations but we can, in practice, interpret d_3 and d_4 as slack-variables.

Which gets us, as far as the restrictions is concerned, to a direct generalisation of Section 10.4 to the QP case.

For the value of the objective function this simple procedure gives rise to an erroneous result. The calculated result so far is actually (once) the linear component of the objective function, $\underline{w}'\underline{x}$, i.e. $x_1 + 2x_2 = 5 + 2 \times 2 = 9$. Reference to (16.5.5) shows that, to actually obtain 2τ as the 2τ entry for a tableau in standard form, this figure needs to be doubled and twice the value of the quadratic component of the objective function needs to be added, i.e. the current initial value for $x_1 = 5$ and $x_2 = 2$ is: $2\tau = 2 \times 9 - 2 \times 5^2 - 2 \times 2^2 = -40$. Denoting the lower bounds vector (at which the solution is initialised as $\underline{x}^*$, we obtain by application of (16.5.5)

$$2\tau^* = 2\underline{w}'\underline{x}^* + \underline{x}^{*\prime}D\underline{x}^* \qquad (16.14.1)$$

The correct set-up tableau for the re-formulated problem is given below as tableaux 16.14c. This tableau also contains adjustments to the rest of the 2τ-row, conform the symmetry-rules.

TABLEAU 16.14 C

CORRECT LOWER-BOUNDS ADJUSTMENT IN QP.

NAME	!!	Y1	Y2	P1	!!	VALUE
D1	!!	-2	-	1	!!	9
D2	!!	-	-2	1	!!	2
S1	!!	-1	-1	-	!!	4
2T	!!	9	2	-4	!!	-40
UB	!!	95	98	100	!!	X
LB	!!	5	2	0	!!	X

16.15 Commented text of a quadratic programming code

Two bits of code are offered in this section. They are: The procedure QUAD which contains the actual quadratic programming algorithm, a main programme which reads the data and converts the tableau on the lines explained in section 16.14. A solution reporting procedure which prints the solution of the specified problem, is not listed: The procedure REPQ is simply an adaptation of the LP reporting procedure REPO from section 10.4, the adaptation consisting in considering the more complicated coding of a QP tableau.

The effect of the adjustment device is to make bounded variables to be in effect restricted to the interval

$$lb_j \leq x_j \leq ub_j \qquad (16.15.1)$$

As the QUAD procedure could malfunction if negative upper limit distances were generated, the main programme contains a check on the non-negativity of the interval spanned by (16.15.1).

For the same reasons as given in section 10.4 for the L.P. case it is inadvisable to declare variables with a "$-\infty$" (e.g.-1000 000) lower limit. Variables without meaningful bounds should be declared as variables without non-negativity restriction.

The algorithm as offered in the QUAD procedure is a combination of the amended convex mode of operation and the non-convex mode of operation, the "mix" of the combination being controlled by the parameter-variable NNEGD.

This is the acceptable number of negative dual variables.

If this number is supplied in the interval $-2 \leq$ NNEGD $\leq$ N+M, the choice between the two modes of separation is determined as follows:

At the moment of calling the convex mode of operation is presumed but a switch to the non-convex mode of operation takes place if either of the following two conditions is encountered:

a) the actual number of negative-valued dual variables (other than equations-shadow prices) is found to exceed the specified permitted number NNEGD, or
b) any sign of non-convexity is observed.

In the interest of being able to apply the non-convex mode of operation also on re-entry of the algorithm, the code also contains a provision for verifying the condition of its

applicability, i.e. the zero diagonal entry in the badname row.

If NNEGD is supplied negative the obvious implication is that the non-convex mode is followed in any case. This would be done, for example if the problem were known to be non-convex despite the lack of manifest non-convexity. If NNEGD is supplied as NNEDG N+M, the convex mode is followed regardless any symptoms of the problem being in fact non-convex. If NNEGD<-2 is supplied, the compulsory use of the non-convex mode is assumed.

Use of these 'non-choice' facilities carries a restriction which is not strictly necessary but which was found useful when using a modified of this algorithm in the context of a more general kind of quadratic programming in Chapter XVIII.

If either mode is employed "perforce" in this way and a shadowprice of a binding inequality is found to display manifest non-convexity by way of a positive non-zero diagonal entry, the inequality stays binding as an equation. i.e. its shadowprice is not selected as badname-variable.

Technically this means that the problem is not actually solved. Note, however, that supplying NNEGD as NNEGD = -1 also effectively makes the use of the non-convex mode obligatory: there are no tableaux in which -1 negative-valued dual variables occur, but the restriction on not permitting a non-convex direction does not apply in that event.

The intertwining of the two modes of operation as well as the complications associated with the upper limit facilities do unavoidably give use to a somewhat complicated code and although some major elements of the QP algorithm may still clearly be recongized in the code-listing it becomes in practice difficult to read and follow the code in its entirety.

The alphanumerical labels and the comment may however go some way towards relieving this problem.

The listings are now given, as follows:

TEXT-LISTING OF THE QUADRATIC PORGRAMMING PORCEDURE.

```
'PROCEDURE' QUAD(T,M,N,NEQ,NAV,NNEGD,ROWLST,COLLST,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,NNEGD,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'BEGIN' 'INTEGER' NAME,BADN,COLN,ROWN,B,BB,D,I,II,J,K,KK,R,
N OF ACT NEG D, N OF P BAS V,RR, NON SQUARE;
'REAL' ASC,QUO,PIV,COP,NUM,FANCYHIGH,PENALTY,CRIT;
'BOOLEAN' CONVEX, CONVEX MODE, FLICKED, LOOKED FOR PAIR;
```

```
'COMMENT'
QUADRATIC PROGRAMMING ALGORITHM.

THE SHORTENED TABLEAU WITHOUT UNIT VECTORS IS USED.

A MIXED SYSTEM OF EQUATIONS AND INEQUALITIES IS EXPECTED.

THE NON-NEGATIVITY RESTRICTION APPLIES TO SOME, BUT NOT TO
ALL SPECIFIED VARIABLES.

ALL SPECIFIED VARIABLES, PRIMAL AS WELL AS DUAL VARIABLES
HAVE UPPER BOUNDS.
THE UPPER BOUNDS ON THE DUAL VARIABLES CORRESPOND TO UNPUNCHED
UNIT VECTORS, REPRESENTING ARTIFICIAL FEASABILITY VARIABLES.

THE DUAL UPPER LIMITS ARE, HOWEVER, ONLY ACTIVATED
WHEN THEY RELATE TO A DUAL DRIVING VARIABLE, I.E. IF
THE DUAL VARIABLE IN QUESTION IS THE SHADOWPRICE OF A
VIOLATED PRIMAL RESTRICTION.
OTHER DUAL UPPER LIMITS ARE CALCULATED, BUT IGNORED BY
THE SEARCH OPERATIONS.

THE VALUES OF THE DUAL UPPERBOUNDS, I.E. THE PENALTY-
COEFFICIENTS FOR THE INTRODUCTION OF ARTIFICIAL VARIABLES
IN THE VECTOR OF BASIC VARIABLES, ARE SET BY THE PROGRAMME.
THE FIGURE AT WHICH THE DUAL UPPERBOUNDS ARE SET
IS ASSIGNED TO A VARIABLE, CALLED PENALTY.
PRIMAL UPPERBOUNDS FOR WHICH A ZERO IS SUPPLIED ARE
REPLACED BY A SUITABLE HIGH NUMBER, CALLED FANCYHIGH.

USER TO DECLARE PREVIOUSLY THE TABLEAU AND THE TWO NAMELISTS.
THE EXTENDED TABLEAU-MATRIX IS OF ORDER M+N+3 BY M+N+2,
AND THE NAMELISTS ARE BOTH OF ORDER M+N.
UNLESS THE RE-ENTRY MODE IS USED,
THE TABLEAU SHOULD BE PART-FILLED BEFORE ENTRY, AS FOLLOWS:

THE SYMMETRIC MATRIX D, OF ORDER N BY N IN THE TOP-LEFTHAND
CORNER, TO INDICATE THE QUADRATIC COMPONENT OF THE MAXIMAND
W TRANSPOSE X + 1/2 X TRANSPOSE D X.
THE BOTTOM PART OF THE LEFTHAND BLOCK-COLUMN SHOULD CONTAIN
THE COEFFICIENTS MATRIX A, REPRESENTING THE LINEAR RESTRIC-
TIONS   A X  LESS THAN OR EQUAL TO B.
THE INTERSECTION OF THE TOP BLOCK-ROW WITH
THE RIGHT-HAND N+M+1 TH COLUMN SHOULD CONTAIN MINUS THE
VECTOR W, THE LINEAR COMPONENT OF THE PREFERENCE FUNCTION,
THE BOTTOM BLOCK-ROW PART OF THAT SAME RIGHT-HAND COLUMN
SHOULD CONTAIN THE VECTOR B.

THE LEFT-HAND SIDE SUB-VECTOR OF THE BOTTOM M+N+2 ND ROW
SHOULD CONTAIN THE SPECIFIED UPPER BOUNDS.
THE SIMILAR RESERVATION FOR LOWER BOUNDS IN ROW M+N+3
IS NOT ACTUALLY USED BY THE QUAD-PROCEDURE, BUT IT IS
SUGGESTED THAT THIS ROW BE FILLED BEFORE ENTRY AS WELL.

THE FIRST NEQ ROWS OF A ARE RESERVED FOR EQUATIONS, AND
THE FIRST NAV COLUMNS ARE RESERVED FOR VARIABLES
WITHOUT SIGN-RESTRICTION.
```

```
THE NAMELISTS ARE FILLED BY THE PROGRAMME,
EXCEPT IF THE RE-ENTRY MODE IS USED.

NAME-CODES ALLOCATED BY THE PROGRAMME ARE AS FOLLOWS:
PRIMAL NAMES HAVE POSITIVE CODES, DUAL NAMES HAVE NEGATIVE
CODES.
THE SPECIFIED ACTIVITIES HAVE CODES EQUAL TO THEIR
INDICES, I.E. FROM 1 TO N.
THE SLACKS OF THE RESTRICTIONS, HAVE NAME-CODES EQUAL TO
THEIR INDICES PLUS 1000, I.E. FROM 1001 TO 1000+M.

DUAL VARIABLES HAVE NEGATIVE NAME-CODES, EQUAL IN ABSOLUTE
VALUE TO THE CODES OF THE ASSOCIATED PRIMAL VARIABLES.
HENCE THE SHADOWPRICES OF THE SPECIFIED ECONOMIC VARIABLES
HAVE NAME-CODES FROM -1 TO -N, AND THE SHADOWPRICES OF THE
RESTRICTIONS HAVE NAMECODES FROM -1001 TO -1000 - M.

DURING CALCULATIONS, THE PROGRAMME MAY GENERATE NAMECODES
WITH ENLARGEMENTS, I.E. WITH AN INCREASED ABSOLUTE VALUE.

UPPER BOUNDS HAVE NAME-CODES EQUAL TO THE NAMES OF THE
CORRESPONDING VARIABLES, INCREASED BY AN ENLARGEMENT OF 2000.
HENCE 2033 IS A PRIMAL VARIABLE, INDICATING THE DISTANCE
OF VARIABLE 33 FROM ITS SPECIFIED UPPER BOUND.
THE DUAL VARIABLE ASSOCIATED WITH THIS UPPER BOUND
(WHEN BINDING), IS THEN CODED AS -2033.

DUAL UPPERBOUNDS AND ARTIFICIAL FEASABILITY VARIABLES HAVE
NAMECODES WHICH ARE 2000 HIGHER IN ABSOLUTE VALUE THEN THE
CORRESPONDING ORDINARY VARIABLES.
FOR EXAMPLE, 1002 IS THE SLACK OF THE 2ND RESTRICTION, -1002
IS THE ASSOCIATED DUAL VARIABLE.
THEN 3002 IS THE ARTIFICIAL VARIABLE WHICH MAKES THE SECOND
RESTRICTION FULFILLED AT THE COST OF A PENALTY LOSS IN VALUE
OF THE OBJECTION FUNCTION, AND -3002 IS THE CORRESPONDING
UPPER BOUND ON THE DUAL VARIABE ASSOCIATED WITH THE
SECOND RESTRICTION.

ANOTHER TYPE OF ENLARGEMENT RELATES TO A MIXED SYSTEM.
FOR EQUATIONS, AND/OR VARIABLES WITHOUT NON-NEGATIVITY
RESTRICTION THERE MAY BE ENLARGEMENTS OF 500.
THESE ENLARGEMENTS OF 500 RECORD THE FACT THAT A PARTICULAR
VARIABLE (OR EQUATION), IS PRESENTED WITH THE OPPOSITE SIGN
COMPARED TO WHAT THE USER PRESENTED AS INPUT-INFORMATION.

HENCE 504 IS THE 4 TH VARIABLE, WHICH IS OF TYPE ABSOLUTE,
WITH THE SIGN INVERTED, I.E. MINUS X(4).
SIMILARLY, -1507 IS THE DUAL VARIABLE ASSOCIATED WITH THE
SEVENTH RESTRICTION, WHICH IS AN EQUATION, AND IT HAS BEEN
NECESSARY TO INVERT THE SIGN OF EVERY COEFFICIENT IN THAT
EQUATION, MAKING THE DUAL VARIABLE INTO MINUS THE
ORIGINALLY SPECIFIED ONE.

ONCE A VARIABLE OF TYPE ABSOLYTE, OR A DUAL VARIABLE OF AN
EQUATION IS A BASIC VARIABLE IN A STANDARD FORM TABLEAU,
THE PROGRAMME RECONVERTS SUCH A VARIABLE (IF IT CARRIES THE
ENLARGEMENT OF 500), TO THE PRESENTATION WHICH
CORRESPONDS TO THE INPUT-DATA.
ENLARGEMENTS OF AN ABSOLUTE VALUE OF 500 THEREFORE DO NOT
OCCUR IN THE OPTIMAL TABLEAU.
;
```

```
FANCYHIGH:=10000; PENALTY:=1000;
CONVEX := 'TRUE'; CONVEX MODE := 'TRUE';
N OF ACT NEG D := 0;


'IF' REENTRY=1 'THEN' 'BEGIN'
  'COMMENT'
  THE PARAMETER REENTRY SHOULD BE SUPPLIED AS ZERO FOR NORMAL
  QUADRATIC PROGRAMMING, AND AS ONE, IF RE-ENTRY AFTER
  ADJUSTMENT OF THE RIGHTHAND-SIDE IS ASKED FOR.

  ON EXIT, THE NORMAL VALUE OF THE REENTRY-PARAMETER IS ZERO,
  REENTRY=-1 INDICADES THAT THE PROBLEM HAS BEEN FOUND
  EMPTY IN THE NON-CONVEX MODE OP OPERATION.
  (THE CONVEX MODE OF OPERATION REACTS TO EMPTYNESS BY
  GENERATING AN ARTIFICIALLY FEASIBLE SOLUTION.)
  OTHERWISE, WHEN A FEASIBLE SOLUTION HAS BEEN FOUND,
  REENTRY=100 INDICATES THAT THE PROBLEM IS NOT CONVEX.
  THIS MEANS IN GENERAL THAT THE REPORTED SOLUTION IS
  A LOCAL RATHER THAN THE GLOBAL MAXIMUM.
  ;
  'GOTO' START; 'END';


COMPLETE TABLEAU:

TRANSPOSE A TO MINUS A TRANSPOSE:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
T[J,I+N] := - T[I+N,J];

TRANSPOSE MINUS W TO MINUS W TRANSPOSE:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' T[M+N+1,J]:=T[J,M+N+1];

TRANSPOSE B TO MINUS B TRANSPOSE:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
T[M+N+1,N+I] := -T[N+I,M+N+1];

FILL BOTTOM RIGHT ZERO BLOCK:
'FOR' I:= N+1 'STEP' 1 'UNTIL' N+M 'DO'
'FOR' J:= N+1 'STEP' 1 'UNTIL' N+M 'DO' T[I,J]:=0;

SET FANCY UPPER BOUNDS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[M+N+2,J]=0
'THEN' T[M+N+2,J]:=FANCYHIGH;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' T[I,N+M+2]:=PENALTY
-T[I,N+M+1];
'FOR' J:=N+1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' T[N+M+2,J]=0
'THEN' T[M+N+2,J]:=PENALTY;

FILL DUMMY UPPER BOUNDS WITH ZEROS:
'FOR' I:=N+1 'STEP' 1 'UNTIL' M+N+1 'DO' T[I,M+N+2]:=0;

ATTEND DEGENERACY:
'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' T[I,M+N+1]=0
'THEN' 'BEGIN'
  NUM:=1;
  'FOR' J:=1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' T[I,J] < 0
  'THEN' NUM:=NUM-T[I,J];
  'IF' I < NAV+1 'THEN' NUM:=-NUM;
  'IF' I>N 'AND' I<N+NEQ+1 'THEN' NUM:=-NUM;
  T[I,M+N+1] := 0.0000000001 * NUM;  'END';
```

```
FILL NAMELISTS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
   COLLST[J]:=J; ROWLST[J]:=-J;  'END';
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
   COLLST[N+I]:=-1000-I; ROWLST[N+I]:=1000+I;  'END';

START:
STANDARD FORM:
BADN:=0;

ATTEND MIXED SYSTEM:

ATTEND ABS V:
'FOR' I:=1 'STEP' 1 'UNTIL' NAV 'DO'
'IF' ((ROWLST[I] < 0 'AND' -ROWLST[I] < NAV+1)
'OR' (-ROWLST[I] > 500 'AND' -ROWLST[I] < 501+NAV))
'AND' 'NOT' T[I,N+M+1] < 0 'THEN' 'BEGIN'
  'IF' T[I,N+M+1] = 0 'THEN' T[I,N+M+1] := 0.00000001;

  TURN ABS VAR ROUND:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' 'BEGIN'
    T[I,J] := -T[I,J];  T[J,I] := -T[J,I]; 'END';
  'IF' COLLST[I] < 500 'THEN' COLLST[I]:=COLLST[I]+500
  'ELSE' COLLST[I]:=COLLST[I]-500;
  ROWLST[I] := -COLLST[I]; 'END';

PUT ABS VAR RIGHT AGAIN:
'FOR' I:=1 'STEP' 1 'UNTIL' NAV 'DO'
'IF' ROWLST[I]=500+I 'THEN' 'BEGIN'
  ROWLST[I]:=ROWLST[I]-500;
  COLLST[I]:=-ROWLST[I];
  T[I,N+M+2] := FANCYHIGH+T[I,N+M+1];
 'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' 'BEGIN'
    T[I,J]:=-T[I,J]; T[J,I]:=-T[J,I]; 'END'; 'END';

ATTEND EQUATION:
'FOR' I:=N+1 'STEP' 1 'UNTIL' N+NEQ 'DO'
'IF' ((ROWLST[I] > 1000 'AND' ROWLST[I] < 1001 + NEQ)
'OR' (ROWLST[I] > 1500 'AND' ROWLST[I] < 1501+NEQ))
'AND' 'NOT' T[I,N+M+1] < 0 'THEN' 'BEGIN'
  'IF' T[I,N+M+1] = 0 'THEN' T[I,N+M+1] := 0.00000001;

  TURN EQUATION ROUND:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' 'BEGIN'
    T[I,J] := -T[I,J];  T[J,I] := -T[J,I]; 'END';
  'IF' ROWLST[I] < 1500 'THEN' ROWLST[I]:= ROWLST[I]+500
  'ELSE' ROWLST[I] := ROWLST[I]-500;
  COLLST[I]:=-ROWLST[I]; 'END';

PUT EQUATION RIGHT AGAIN:
'FOR' I:=N+1 'STEP' 1 'UNTIL' N+NEQ 'DO'
'IF' COLLST[I]=1500+I-N 'THEN' 'BEGIN'
  COLLST[I]:=COLLST[I]-500;
  ROWLST[I]:=-COLLST[I];
  T[I,N+M+2] := PENALTY+T[I,N+M+1];
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' 'BEGIN'
    T[I,J]:=-T[I,J]; T[J,I]:=-T[J,I]; 'END'; 'END';
```

```
SELECT BADNAME:
PREPARE INV OF MODE:
LOOKED FOR PAIR := 'FALSE';
N OF ACT NEG D := 0; N OF P BAS V := 0;

INITIATE ASCENT LOW:
ASC:=-10000000000000000;
'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO' 'BEGIN'

  'IF' T[I,I] > 0 'THEN' 'BEGIN'
    'IF' T[I,I] < 0.0000001 'THEN' T[I,I] := 0;
    'IF' T[I,I] > 0 'THEN' CONVEX MODE := 'FALSE';
    'IF' NNEGD > N+M 'THEN' CONVEX MODE := 'TRUE'; 'END';

  COUNT:
  'IF' I > NAV 'AND' 'NOT'
  (I>N 'AND' I<N+NEQ+1) 'THEN' 'BEGIN'
    'IF' ROWLST[I] < 0 'AND' T[I,N+M+1] < 0
    'THEN' N OF ACT NEG D := N OF ACT NEG D + 1; 'END';

  'IF' ROWLST[I] > 0
  'THEN' N OF P BAS V := N OF P BAS V + 1;

  ABS VAR STAYS:
  'IF' I < NAV+1 'AND'  (ROWLST[I] = I 'OR' ROWLST[I]=500+I)
  'THEN' 'GOTO' END OF BADNAME SELECTION LOOP;

  DUAL VAR EQUATION STAYS:
  'IF' I > N 'AND' I < N+NEQ+1
  'AND' (-ROWLST[I] = 1000+I-N 'OR' -ROWLST[I] = 1500+I-N)
  'THEN' 'GOTO' END OF BADNAME SELECTION LOOP;

  CRIT := 1;
  'IF' ROWLST[I] < 0 'THEN' 'BEGIN'

    REFUSE NON CONVEX SLACKS WHEN MODE FORCED:
    'IF' -ROWLST[I] > 1000+NEQ 'AND' -ROWLST[I]<1001+M
    'AND' (NNEGD>N+M 'OR' NNEGD<-1) 'THEN' 'BEGIN'
      'IF' T[I,I] > 0
      'THEN' 'GOTO' END OF BADNAME SELECTION LOOP; 'END';

    'FOR' II:=NAV+1 'STEP' 1 'UNTIL' N+M 'DO'
    'IF' ROWLST[II] > 0 'AND' T[II,N+M+1] < 0
    'AND' T[II,I] < 0 'THEN' CRIT:=CRIT-T[II,I]; 'END';

  CRIT := CRIT*T[I,N+M+1]*(T[I,I]-0.0001);

  'IF' T[I,N+M+1] < 0 'AND' CRIT > ASC 'THEN' 'BEGIN'
    BADN:=ROWLST[I]; B:=I; ASC:=CRIT;
    'END';

  END OF BADNAME SELECTION LOOP:  'END';

'IF' T[B,B] > 0 'THEN' CONVEX := 'FALSE';
'IF' NNEGD > N+M 'THEN' CONVEX MODE := 'TRUE';
```

```
'IF' BADN=0 'THEN' 'GOTO' END OF QUAD;

'IF' 'NOT' CONVEX 'OR' N OF ACT NEG D > NNEGD
'THEN' CONVEX MODE := 'FALSE';
'IF' NNEGD > N+M 'THEN' CONVEX MODE := 'TRUE';

'IF' CONVEX MODE 'OR' LOOKED FOR PAIR 'THEN' 'GOTO' PHASE II;
'IF' BADN>0 'AND' T[B,B]=0 'THEN' 'GOTO' PHASE I;

'FOR' I:=NAV+1 'STEP' 1 'UNTIL' N,
N+NEQ 'STEP' 1 'UNTIL' N+M 'DO'
'IF' T[I,M+N+1] < 0 'AND' ROWLST[I] > 0 'AND' T[I,I]=0
'THEN' 'GOTO' PHASE I SEARCH;
'GOTO' PHASE II;

PHASE I SEARCH:
B:=0; BADN := 0; ASC := 0;
'FOR' I := NAV+1 'STEP' 1 'UNTIL' N,
N+1 'STEP' 1 'UNTIL' N+M 'DO'
'IF' ROWLST[I] > 0 'AND' T[I,N+M+1] < 0 'THEN' 'BEGIN'
  'IF' -T[I,N+M+1] > ASC 'THEN' 'BEGIN'
    B:=I; ASC := -T[I,N+M+1]; BADN:=ROWLST[I]; 'END'; 'END';

PHASE I:
COLN :=0; K:=0; ASC:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N+M 'DO'
'IF' COLLST[J] > 0
'AND' (T[B,J] < 0 'OR' J < NAV+1) 'THEN' 'BEGIN'

  'IF' J<NAV+1 'THEN' 'BEGIN'
    'IF' 1000*ABS(T[B,J]) > ASC 'THEN' 'BEGIN'
      K:=J; COLN := COLLST[K]; ASC:=1000*ABS(T[B,K]);
      'END'; 'END';

  'IF' -T[B,J]>0 'THEN' 'BEGIN'

    'IF' T[J,N+M+1] < 0 'THEN' 'BEGIN'
      'IF' 10*(-T[B,J]-T[J,N+M+1]) > ASC 'THEN' 'BEGIN'
        ASC := -10*(T[B,J]+T[J,N+M+1]);
        K:=J; COLN:=COLLST[J]; 'END'; 'END'

    'ELSE' 'BEGIN'
      'IF' T[J,N+M+1]<0.0000001
      'THEN' T[J,N+M+1]:=0.0000001;
      'IF' -T[B,J]/T[J,N+M+1] > ASC 'THEN' 'BEGIN'
        K:=J; COLN:=COLLST[K];
        ASC:=-T[B,J]/T[J,N+M+1]; 'END';
      'END'; 'END';
  'END';


'IF' COLN = 0 'THEN' 'BEGIN'
  'COMMENT'
  SIGNALLING OF AN EMPTY PROBLEM IN THE NON-CONVEX MODE;
  REENTRY := -1;
  NEWLINE(1);
  WRITETEXT('('ROW%')');
  WRITE(30,FORMAT('('S-NDDDD')'),BADN);
  WRITETEXT('('%%FOUND%IMPOSSIBLE%TO%SATISFY')');
  'GOTO' FINAL END OF QUAD; 'END';
```

```
ATTEND FREE VAR IN NON CONVEX MODE:
'IF' (COLN>0 'AND' K<NAV+1 'AND' T[B,K]>0)
'THEN' 'BEGIN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M 'DO' T[K,J]:=-T[K,J];
  NUM := T[K,N+M+1]; T[K,N+M+1]:=T[K,N+M+2];
  T[K,N+M+2] := NUM;
  'FOR' I:=1 'STEP' 1 'UNTIL' N+M+1 'DO' T[I,K]:=-T[I,K];
  'IF' COLN < 500 'THEN' COLN:=COLN+500
  'ELSE' COLN:=COLN-500;
  COLLST[K]:=COLN; ROWLST[K]:=-COLN; 'END';

'GOTO' INITIATE SEARCH FOR PIVOT ROW;

PHASE II:
D:=K:=B; COLN:=-BADN;

NON STANDARD FORM:
'IF' ABS(ROWN) < 2000 'THEN' FLICKED := 'FALSE';

INITIATE SEARCH FOR PIVOT ROW:
QUO:=1000000000000; ROWN:=0;

CHECK ON ROUNDING NON CONVEXITY:
'IF' T[B,K] > 0 'THEN' 'BEGIN'
  'IF' T[B,K] < 0.00000001 'THEN' T[B,K] := 0; 'END';

'IF' 'NOT' CONVEX 'OR' N OF ACT NEG D > NNEGD
'THEN' CONVEX MODE := 'FALSE';
'IF' NNEGD > N+M 'THEN' CONVEX MODE := 'TRUE';
'IF' N OF P BAS V = M 'AND' 'NOT' CONVEX MODE
'AND' BADN > 0 'THEN' 'GOTO' SEEK SMALLEST QUOTIENT;

TRY THE BADNAME ROW:
BUT NOT IN NON CONVEX MODE:
'IF' T[B,K] < -0.0000000001 'THEN' 'BEGIN'
  'COMMENT'
  THE BADNAME-ROW HAS PREFERENCE, ACCEPT A NEGATIVE PIVOT;
  R:=B; ROWN:=BADN; QUO:=T[B,M+N+1]/T[B,K]; 'END';

SEEK SMALLEST QUOTIENT:
'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO'
'IF' ABS(T[I,K]) > 0.000000001 'THEN' 'BEGIN'

  CONSIDER UPPERB ON DUAL VAR:
  'IF' ROWLST[I] < 0 'AND' ROWLST[I] = -BADN
  'AND' T[I,K] < 0 'THEN' 'GOTO' TRY UPPER BOUND;

  KEEP SHADOWPRICE EQUATION BASIC VAR:
  'IF' (ROWLST[I] < -1000 'AND' ROWLST[I] > -1000-NEQ-1)
  'OR' (ROWLST[I] < -1500 'AND' ROWLST[I] > -1500-NEQ-1)
  'THEN' 'GOTO' DONE;

  CONSIDER EQUATION SLACK EVEN IF NEG:
  'IF' ((ROWLST[I] > 1000 'AND' ROWLST[I]<1001+NEQ)
  'OR' (ROWLST[I] > 1500 'AND' ROWLST[I] < 1501+NEQ))
  'AND' T[I,K]<0 'AND' T[I,N+M+1]<0
  'THEN' 'GOTO' TRY VALUE COL;
```

```
  INVESTIGATE UPPERB PRIMAL VAR:
  'IF' ROWLST[I] > 0 'AND' ROWLST[I] < N+1
  'AND' T[I,K] < 0 'THEN' 'GOTO' TRY UPPER BOUND;

  KEEP ABS VARIABLE BASIC:
  'IF' (ROWLST[I] > 0 'AND' ROWLST[I] < NAV+1)
  'OR' (ROWLST[I] > 500 'AND' ROWLST[I] < NAV+501)
  'THEN' 'GOTO' DONE;

  ATTEND SPECIAL FEATURES OF NON CONVEX MODE:
  'IF' N OF P BAS V = M 'AND' 'NOT' CONVEX MODE
  'AND' BADN>0'THEN' 'BEGIN'
    'IF' ROWLST[I]<0 'THEN' 'GOTO' DONE;
    'IF' ROWLST[I]=BADN 'AND' T[I,K]<0
    'AND' 'NOT' T[I,N+M+1]>0 'THEN' 'GOTO' TRY VALUE COL;
    'END';

  REFUSE QUOTIENT WITH THE WRONG SIGN:
  'IF' T[I,K] < 0 'OR' T[I,M+N+1] < 0 'THEN' 'GOTO' DONE;

  TRY VALUE COL:
  'IF' T[I,M+N+1]/T[I,K] < QUO 'THEN' 'BEGIN'

    'IF' ROWLST[I]<0 'THEN' 'BEGIN'

      DO I NEED TO ELEMINATE:
      'IF' BADN < 0 'THEN' 'GOTO' DONE;


      CHECK SUBSPACE CONVEXITY:
      WHEN INCOMING VAR IS DRIVING VAR:
      'IF' COLN=-BADN 'AND' T[I,I] > 0 'THEN' 'GOTO' DONE;

      FOR DRIVING VAR IN NON STAND F:
      'IF' 'NOT' COLN=-BADN
      'AND' T[D,I]-T[D,K]*T[I,I]/T[I,K]>0
      'THEN' 'GOTO' DONE; 'END';

    R:=I; ROWN:=ROWLST[I]; QUO:=T[I,M+N+1]/T[I,K]; 'END';
  'GOTO' DONE;

  TRY UPPER BOUND:
  'IF' ABS(-T[I,M+N+2]/T[I,K]) < ABS(QUO) 'THEN' 'BEGIN'
    R:=I; QUO:=-T[I,M+N+2]/T[I,K];
    NAME:=ABS(ROWLST[I]);
    'IF' NAME < 2000 'THEN' ROWN:=NAME+2000
    'ELSE' ROWN:=NAME-2000;
    'IF' ROWLST[I] < 0 'THEN' ROWN:=-ROWN; 'END';

  DONE:  'END';

'IF' T[B,B]=0 'AND' 'NOT' CONVEX MODE
'AND' BADN > 0 'AND' COLLST[B]=-BADN 'THEN' 'BEGIN'
  'IF' T[B,K] > -0.0000000001 'THEN' 'GOTO' FLY THROUGH;
  'IF' R=0 'THEN' 'GOTO' ACCEPT BV;
  'IF' 'NOT' T[B,N+M+1]/T[B,K] < QUO 'THEN' 'BEGIN'
    'IF' N OF P BAS V = M 'THEN' 'GOTO' NOT SMALLER;
    'GOTO' ACCEPT BV; 'END';
  'IF' ABS(T[R,K]) < 0.01 'THEN' 'GOTO' ACCEPT BV;
  'IF' -QUO*T[N+M+1,K] > -(T[B,N+M+1]/T[B,K])*T[N+M+1,B]
  'THEN' 'GOTO' FLY THROUGH;
  ACCEPT BV:
  ROWN:=BADN; R:=B; QUO:=T[B,N+M+1]/T[B,K];
```

```
  NOT SMALLER:
  FLY THROUGH:
  'IF' N OF P BAS V > M 'THEN' 'BEGIN'
    'IF' 'NOT' ROWN=BADN 'THEN' 'BEGIN'
      LOOKED FOR PAIR := 'TRUE'; 'GOTO' PHASE II; 'END';
    NON SQUARE := N OF P BAS V - M;
    N OF P BAS V := M; 'END'; 'END';


ATTEND UPPER BOUND COLUMN VAR:

BIND SPECIFIED VAR:
'IF' COLN > 0 'AND' COLN < N+1 'THEN' 'GOTO' UPPERB ALLOWED;

BIND UPPER LIMIT DISTANCE:
'IF' COLN > 2000 'AND' COLN < 2001+N
'THEN' 'GOTO' UPPERB ALLOWED;

BIND DUAL DRIVING VARIABLE:
'IF' (COLN<0 'AND' COLN=-BADN) 'THEN' 'BEGIN'
  'IF' ABS(COLN)<N+1 'THEN' T[N+M+2,K]:=PENALTY;
  'GOTO' UPPERB ALLOWED; 'END';

UPPERBOUND NOT ALLOWED:
'GOTO' CHECK FOR BOUNDEDNESS;

UPPERB ALLOWED:

'IF' ROWN=0 'OR' QUO > T[M+N+2,K] 'THEN' 'BEGIN'

  START OF VECTORS ONLY UPDATING LOOP:

  MARK NO PIVOT ROW:
  R := 0;

  QUO := T[M+N+2,K];

  ASSEMBLE ROWN:
  'IF' COLN < 0 'THEN' NAME := -COLN 'ELSE' NAME := COLN;
  'IF' NAME < 2000 'THEN' ROWN := NAME+2000
  'ELSE' ROWN:=NAME-2000;
  'IF' COLN < 0 'THEN' ROWN := -ROWN;

  COLLST[K]:=ROWN;

  UPDATE VECTORS:
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO' 'BEGIN'
    T[I,M+N+1]:=T[I,M+N+1]-T[I,K]*QUO;
    T[I,M+N+2]:=T[I,M+N+2]+T[I,K]*QUO;
    T[I,K] := -T[I,K]; 'END';

  'IF' (ABS(NAME) = ABS(BADN))
  'OR' ('NOT' CONVEX MODE 'AND' N OF P BAS V = M
  'AND' BADN>0 'AND' ROWN>0) 'THEN' 'GOTO'
  STANDARD FORM DOUBLE STEP;
```

```
ATTEND COMPLEMENT:
'IF' ROWLST[K]=-COLN 'THEN' 'BEGIN'

  ATTEND COMPLEMENTARY ROW:
  ROWLST[K]:=-ROWN;
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M 'DO' T[K,J]:=-T[K,J];
  COP:=T[K,M+N+1]; T[K,M+N+1]:=T[K,M+N+2];
  T[K,M+N+2]:=COP; 'END' 'ELSE' 'BEGIN'

  ATTEND COMPLEMENTARY COLUMN:
  COLLST[RR]:=-ROWN;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO'
  T[I,RR]:=-T[I,RR];

  ATTEND 2T ROW OF THE DUAL OF THE UPPERB:
  'IF' COLN < 0 'THEN' T[M+N+1,RR]:=T[M+N+1,RR]+T[M+N+2,K]
  'ELSE' T[M+N+1,RR]:=T[M+N+1,RR]-T[M+N+2,K]; 'END';


STANDARD FORM DOUBLE STEP:
'IF' (ABS(NAME) = ABS(BADN))
'OR' ('NOT' CONVEX MODE 'AND' N OF P BAS V = M
'AND' BADN>0 'AND' ROWN>0)
'THEN' 'BEGIN'

  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' T[K,J]:=-T[K,J];

  'IF' ROWN < 0 'THEN'
  T[M+N+1,M+N+1]:=T[M+N+1,M+N+1]+T[M+N+1,K]*QUO
  'ELSE' T[M+N+1,M+N+1]:=T[M+N+1,M+N+1]-
  T[M+N+1,K]*QUO;

  'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO'
  'IF' ROWLST[I] < 0 'THEN' T[M+N+1,I]:=T[I,M+N+1]
  'ELSE' T[M+N+1,I]:=-T[I,M+N+1];
  ROWLST[K] := -ROWN;

  'IF' 'NOT' ABS(ABS(BADN)-ABS(ROWN))=2000 'THEN' 'BEGIN'
    'COMMENT'
    PAIR IN PHASE I IN NON-CONVEX MODE;
    CHECK ON FLYING THROUGH:
    'IF' 'NOT' T[K,N+M+1] < 0 'THEN' 'GOTO' STANDARD FORM;
    'GOTO' PHASE I; 'END';

  'GOTO' STANDARD FORM; 'END';

ADJ COLLST AND PERM UPPERBOUND COLUMNS:
COLLST[B] := ROWN;  COLLST[K] := -ROWN;
'FOR' I:=1 'STEP' 1 'UNTIL' M+N+2 'DO' 'BEGIN'
  COP:=T[I,B]; T[I,B]:=T[I,K]; T[I,K]:=COP; 'END';

COLN:=-ROWN;
'GOTO' CHECK FOR STATUS;
END OF VECTORS ONLY UPDATING LOOP: 'END';
```

```
CHECK FOR BOUNDEDNESS:
'IF' ROWN=0 'THEN' 'GOTO' UNBOUNDED;

SAVE INDICES:
KK := K; RR := R;  BB := B;
'IF' 'NOT' CONVEX MODE 'AND' N OF P BAS V = M
'AND' BADN>0 'THEN' B:=R;

ATTEND NON NEGATIVITY EQ SL:
'IF' 'NOT' ROWN=BADN 'AND'
((ROWN>1000 'AND' ROWN<1001+NEQ)
'OR' (ROWN>1500 'AND' ROWN<1501+NEQ))
'THEN' 'BEGIN'
  'IF' T[R,N+M+1]<0 'AND' T[R,K]<0 'THEN' 'BEGIN'

    ADJUST CODE:
    'IF' ROWN<1500 'THEN' ROWLST[R]:=ROWN:=ROWN+500
    'ELSE' ROWLST[R]:=ROWN:=ROWN-500;
    COLLST[K]:=-ROWN;

    FLICK ROUND:
    'FOR' I:=1 'STEP' 1 'UNTIL' N+M+1 'DO' 'BEGIN'
      T[I,R]:=-T[I,R]; T[R,I]:=-T[R,I]; 'END'; 'END'; 'END';

PERMUTE COLUMNS:
'IF' ('NOT' CONVEX MODE 'AND' N OF P BAS V = M
'AND' BADN > 0)
'OR' (ROWN#BADN 'AND' ABS(ROWN)#ABS(BADN)+2000)
'THEN' 'BEGIN'
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+2 'DO' 'BEGIN'
    COP:=T[I,K]; T[I,K]:=T[I,R]; T[I,R]:=COP; 'END';
  COLLST[K]:=COLLST[R]; K:=R; 'END';
ADJUST COLLST:
COLLST[K]:=ROWN;

ATTEND UPPERBOUNDS OF PIVOTAL PAIR:
'IF' ROWN#ROWLST[R]
'THEN' 'GOTO' PRESENT NEWLY BINDING UPPERBOUND AS ROW;
NUM:=T[R,M+N+1]+T[R,M+N+2];
'IF' ROWN>2000 'THEN' NUM:=T[R,N+M+2]-T[R,N+M+1];
T[R,M+N+2]:=T[M+N+2,K]-QUO;
T[M+N+2,K]:=NUM;

PRESENT NEWLY BINDING UPPER BOUND AS ROW:
'IF' ROWN # ROWLST[R] 'THEN' 'BEGIN'
  FLICKED := 'TRUE';
  ROWLST[R]:=ROWN;
  'FOR' J:=1 'STEP' 1 'UNTIL' M+N 'DO' T[R,J]:=-T[R,J];
  NUM:=T[R,M+N+1]; T[R,M+N+1]:=T[R,M+N+2]; T[R,M+N+2]:=NUM;

  GENERATE COMP UL COLUMN:
  'IF' ABS(ROWN)#ABS(BADN)+2000 'THEN' 'BEGIN'
    'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO' T[I,KK]:=-T[I,KK];
    COLLST[KK]:=-ROWN;
    'IF' ROWN<0 'THEN' T[N+M+1,KK]:=T[N+M+1,KK]+
    T[R,N+M+1]+T[R,N+M+2]
    'ELSE' T[N+M+1,KK]:=T[N+M+1,KK]-
    T[R,N+M+1]-T[R,N+M+2]; 'END';
```

```
  STILL ATTEND UPPERBOUNDS OF PIVOTAL PAIR:
  NUM:=T[R,M+N+1]+T[R,M+N+2];
  T[R,M+N+2]:=T[M+N+2,K]-QUO;
  T[M+N+2,K]:=NUM;
  'END';

PERMUTE ROWS AND ADJUST ROWLST:
'IF' 'NOT' CONVEX MODE 'AND' N OF P BAS V = M
'AND' BADN>0 'AND' COLN>0 'THEN' B:=KK;
'FOR' J:=1 'STEP' 1 'UNTIL' M+N+2 'DO' 'BEGIN'
  COP:=T[R,J]; T[R,J]:=T[B,J]; T[B,J]:=COP; 'END';
ROWLST[R]:=ROWLST[B]; ROWLST[B]:=COLN;


UPDATE:
PIV:=T[B,K];

CLEAN AND UPDATE PIVOT ROW:
'FOR' J:=1 'STEP' 1 'UNTIL' M+N+1 'DO' 'BEGIN'
  'IF' J < M+N+1 'AND' ABS(T[B,J]) < 0.0000000001
  'THEN' T[B,J]:=0;
  'IF' T[B,J] # 0 'THEN' T[B,J]:=T[B,J]/PIV; 'END';

ADJUST COEFFICIENTS MATRIX:
'FOR' J:=1 'STEP' 1 'UNTIL' K-1,
K+1 'STEP' 1 'UNTIL' M+N+1 'DO'
'IF' T[B,J] # 0 'THEN'
'FOR' I:=1 'STEP' 1 'UNTIL' B-1,
B+1 'STEP' 1 'UNTIL' M+N+1 'DO'
'IF' T[I,K] # 0 'THEN' T[I,J]:=T[I,J]-T[B,J]*T[I,K];

'FOR' I:=1 'STEP' 1 'UNTIL' B-1,B+1 'STEP' 1 'UNTIL' M+N+1
'DO' 'BEGIN'
  T[I,M+N+2]:=T[I,M+N+2]+T[B,M+N+1]*T[I,K];
  T[I,K]:=-T[I,K]/PIV; 'END';
T[B,K]:=1/PIV;


REPRESENT NEW UPPERBO BASIC VAR NORMALLY:
'IF' ABS(COLN) < 1000
'THEN' 'GOTO' EXTRA STEP BACK TO STANDARD F;

'IF' COLN > 2000 'OR' COLN < -3000
'OR' (COLN < -2000 'AND' 'NOT' FLICKED) 'THEN' 'BEGIN'
  'COMMENT'
  SPECIFIED VARIABLES, I.E. PRIMAL NAMECODES ENTERING AS
  2000 + J, AND DUAL UPPERBOUNDS ENTERING AS -3000 -I,
  AND UPPER LIMITS ON DUAL SLACKS ENTERING AS -2000-J,
  FOLLOWING THE ELEMINATION OF ARTIFICIAL NEGATIVES OF
  ORDINARY VARIABLES (PERMITTED ONLY IN THE REENTRY MODE),
  ARE TO BE RE-DEFINED.
  THEY THEN BECOME ORDINARY VARIABLES, CODED AS J FOR THE
  J TH PRIMAL VARIABLE AND -1000 -I FOR THE I TH DUAL
  VARIABLE, OR -J FOR THE J TH DUAL SLACK.  ;

  'IF' COLN > 0 'THEN' NAME:=COLN-2000
  'ELSE' NAME:=COLN+2000;
  ROWLST[B]:=NAME;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO' T[B,I]:=-T[B,I];
  NUM:=T[B,M+N+1]; T[B,M+N+1]:=T[B,M+N+2]; T[B,M+N+2]:=NUM;
```

```
  'IF' COLN = -COLLST[B] 'THEN' 'BEGIN'
    'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO' T[I,B]:=-T[I,B];
    COLLST[B]:=-NAME;
    'IF' NAME > 0 'THEN' T[M+N+1,B] := T[M+N+1,B]
    -T[B,M+N+1]-T[B,M+N+2] 'ELSE'
    T[M+N+1,B]:=T[M+N+1,B]+T[B,M+N+1]+T[B,M+N+2]; 'END';
  END OF INCOMING VARIABLE LIMIT RENORMALISATION: 'END';

EXTRA STEP BACK TO STANDARD F:
UPPER LIMIT ELEMINATION TO ST F:
'IF' ABS(ROWN) = ABS(BADN)+2000 'THEN' 'BEGIN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+1 'DO' T[R,J]:=-T[R,J];
  'IF' BADN > 0 'THEN'
  T[M+N+1,M+N+1]:=T[M+N+1,M+N+1]-PENALTY*T[R,M+N+1]
  'ELSE' T[M+N+1,M+N+1]:=T[M+N+1,M+N+1] +
  T[M+N+2,K]*T[R,M+N+1];

  'FOR' J:=1 'STEP' 1 'UNTIL' M+N 'DO'
  'IF' ROWLST[J] < 0 'THEN' T[M+N+1,J]:=T[J,M+N+1] 'ELSE'
  T[M+N+1,J]:=-T[J,M+N+1];
  'IF' BADN > 0 'THEN' ROWLST[R] := BADN+2000
  'ELSE' ROWLST[R] := BADN-2000;
  'GOTO' STANDARD FORM; 'END';

CHECK FOR STATUS:

ATTEND PAIR IN NON CONVEX MODE:
'IF' 'NOT' CONVEX MODE 'AND' N OF P BAS V = M 'THEN' 'BEGIN'
  'IF' BADN>0 'AND' COLN<0 'THEN' 'BEGIN'
    'IF' COLN=-BADN 'THEN' 'GOTO' START;
    'IF' T[BB,N+M+1] > 0 'THEN' 'GOTO' STANDARD FORM;
    N OF P BAS V := N OF P BAS V + NON SQUARE;
    B:=BB; 'GOTO' PHASE I; 'END';
  'IF' BADN > 0 'AND' COLN>0 'THEN' 'BEGIN'
    'IF' R=0 'THEN' RR:=K;
    'IF' R=0 'THEN' KK:=K;
    B:=R:=RR; K:=KK;
    NAME:=ROWN; ROWN:=-COLN; COLN:=-NAME;
    'GOTO' ADJUST COLLST; 'END'; 'END';

'IF' BADN=ROWN 'THEN' 'GOTO' START;

PREPARE NON STANDARD FORM STEP:
'IF' R=0 'THEN' KK:=K;
'IF' R=0 'THEN' R:=B;
K:=KK; B:=R; COLN:=-ROWN;
'GOTO' NON STANDARD FORM;

UNBOUNDED:
NEWLINE(1);
WRITETEXT('('UNBOUNDED%COLUMN')');
WRITE(30,FORMAT('('S-NDDDDDDDDD')'),COLN);
NEWLINE(1);

END OF QUAD:

'IF' 'NOT' CONVEX 'THEN' REENTRY := 100;

FINAL END OF QUAD: 'END';
```

```
TEXT-LISTING OF THE CONTROLLING MAIN PROGRAMME.

'BEGIN' 'INTEGER' M,N,NAV,NEQ,REENTRY,I,J,NNEGD;

'PROCEDURE' QUAD(T,M,N,NEQ,NAV,NNEGD,ROWL,COLL,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,NNEGD,REENTRY;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'PROCEDURE' MATI(MATR,M,N,SR,SC);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC; 'ALGOL';
'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWL,COLL);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';
'PROCEDURE' REPQ(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'COMMENT'
SIMPLEX ALGORITHM FOR QUADRATIC PROGRAMMING.
FOR DETAILS OF THE ALGORITHM SEE THE TEXT OF
THE PROCEDURE QUAD.

PRESENTATION OF DATA:
FIRST THE NUMBER OF RESTRICTIONS I.E. M,
THEN THE NUMBER OF VARIABLES, I.E. N,
FOLLOWED BY THE NUMBER OF EQUATIONS,  NEQ,
THEREAFTER NAV, THE NUMBER OF ('FREE') VARIABLES
TO WHICH THE NON-NEGATIVITY RESTRICTION DOES  NOT  APPLY.
THEREAFTER NNEGD,
THE NUMBER OF NEGATIVE-VALUED DUAL VARIABLES WHICH IS PER-
MITTED UNDER THE CONVEX MODE OF OPERATION.

THEREAFTER PUNCH EACH ROW OF THE COMPOSITE MATRIX
   D    -W
   A     B
   U'    0 OR 1
   L'    0
TO REPRESENT:
MAXIM  W' X + 1/2 X' D X
A.X  < OR = B
AND  X  < OR = U
AND  X  > OR = L
;

READ ORDER PARAMETERS ETC:
M:=READ;  N:=READ;  NEQ:=READ;  NAV:=READ;  NNEGD:=READ;

NEWLINE(1);
'IF' NEQ > M 'THEN'
WRITETEXT('('YOU%HAVE%ERRONEOUSLY%SPECIFIED%MORE%
EQUATIONS%THAN%RESTRICTIONS')');

NEWLINE(1);
'IF' NAV > N 'THEN' WRITETEXT('('YOU%HAVE%ERRONEOUSLY%
SPECIFIED%MORE%FREE%VARIABLES%THAN%VARIABLES')');

REENTRY:=0;
```

```
'BEGIN'
  'ARRAY' TA[1:M+N+3,1:M+N+2];
  'INTEGER' 'ARRAY' ROWL,COLL[1:M+N];

  READ MAIN TABLEAU MATRIX:
  MATI(TA,N+M+2,N+1,0,0);

  REORDER TO APPROPRIATE BLOCKS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
  'FOR' I:=2,1 'DO'
  TA[M+N+I+1,J]:=TA[M+N+I,J];
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO' 'BEGIN'
    TA[I,M+N+1]:=TA[I,N+1]; TA[I,N+1]:=0; 'END';

  RE INTERPRET:

  INITIAL VALUE:

  FOR LINEAR PART OF INITIAL VALUE:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  TA[N+M+1,N+M+1] :=
  TA[N+M+1,N+M+1] -2*TA[J,N+M+1]*TA[N+M+3,J];

  FOR QUADRATIC PART OF INITIAL VALUE:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  TA[N+M+1,N+M+1]:=
  TA[N+M+1,N+M+1] + TA[N+M+3,I]*TA[I,J]*TA[N+M+3,J];

  REST OF VALUE COL:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' N+M 'DO'
  TA[I,N+M+1]:=TA[I,N+M+1]-TA[I,J]*TA[N+M+3,J];

  UPPER LIMITS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'IF' TA[N+M+2,J]=0 'THEN' TA[N+M+2,J]:=100000;
    TA[N+M+2,J] := TA[N+M+2,J]-TA[N+M+3,J];
    'IF' TA[N+M+2,J] < 0 'THEN' 'BEGIN'
      NEWLINE(1);
      WRITETEXT('('YOU%HAVE%SUPPLIED%A%LOWER%LIMIT%IN%
      EXCESS%OF%THE%CORRESPONDING%UPPER%LIMIT')');
      NEWLINE(1);
      WRITETEXT('('THE%QP%PROBLEM%IS%THEREFORE%EMPTY.')');
      'END';
    'END';

  NOW SOLVE:
  QUAD(TA,M,N,NEQ,NAV,NNEGD,ROWL,COLL,REENTRY);

  REPORT SOLUTION:
  REPQ(TA,M,N,NEQ,NAV,ROWL,COLL);
  'IF'  M+N < 13
  'THEN' TABO(TA,M+N,M+N,0,0,2,2,ROWL,COLL)
  'ELSE' TABO(TA,M+N,0,0,M+N,1,0,ROWL,COLL);
  NEWLINE(1);
  WRITETEXT('('SOLUTION%VALUE%IF%IN%STANDARD%FORM')');
  WRITE(30,FORMAT('('S-NDDDD.DDD')'),TA[M+N+1,M+N+1]/2);
  NEWLINE(1);
'END'; 'END'
```

ANSWER-SHEET 16.8 EX

NAME	X 1	X 2	P 1	P 2	VALUE	
D 1	2	1	1	-3	1	SET-UP TABLEAU.
D 2	1	-	(1)	-1	2	S1 IS BADNAME, P1 DRIVING VAR.
S 1	-1	-1	-	-	-5	REFUSE D1 AS LEAVING VARIABLE,
S 2	3	1	-	-	30	ON ACCOUNT OF THE QUALIFIER.
2T	1	2	5	-30	-	D2 BECOMES LEAVING VARIABLE.

NAME	X 1	D 2	X 2	P 2	VALUE	
D 1	1	-1	1	-2	-1	IN THE RESULTING NON-
S 1	-1	-	(-1)	-	-5	STANDARD FORM TABLEAU,
P 1	1	1	-	-1	2	X2 BECOMES INCOMING VAR.,
S 2	3	-	1	-	30	BY THE COMPLEMENTARITY RULE.
2T	-4	-5	2	-25	-10	S1 IS NOW LEAVING VARIABLE.

NAME	X 1	D 2	S 1	P 2	VALUE	
D 1	-	-1	1	-2	-6	WE NOW HAVE A STANDARD FORM
X 2	1	-	-1	-	5	TABLEAU.
P 1	(1)	1	-	-1	2	D1 BECOMES BADNAME,
S 2	2	-	1	-	25	X1 IS DRIVING VARIABLE,
2T	-6	-5	2	-25	-20	P1 IS SELECTED AS
						LEAVING VARIABLE.

NAME	S 1	D 2	P 1	P 2	VALUE	
X 1	-	1	1	-1	2	IN THIS NON-STANDARD FORM
X 2	-1	-1	-1	1	3	S1 IS INCOMING VARIABLE
D 1	1	-1	-	-2	-6	BY THE COMPLEMENTARITY RULE.
S 2	(1)	-2	-2	2	21	S2 IS LEAVING VARIABLE, BY
2T	2	1	6	-31	-8	THE RULE OF THE SMALLEST
						(ONLY) QUOTIENT.

NAME	P 2	D 2	P 1	S 2	VALUE	
X 1	-1	1	1	-	2	ANOTHER NON-STANDARD FORM
X 2	3	-3	-3	1	24	TABLEAU, IN WHICH THE
S 1	2	-2	-2	1	21	COMPLEMENTARITY RULE
D 1	(-4)	1	2	-1	-27	ACTIVATES P2 AS INCOMING
2T	-35	5	10	-2	-50	VARIABLE, D1 IS THE
						LEAVING VARIABLE.

NAME	D 1	D 2	P 1	S 2	VALUE	
X 1	-0.25	0.75	0.50	0.25	8.75	WE HAVE NOW REACHED
X 2	0.75	-2.25	-1.50	0.25	3.75	THE OPTIMUM.
S 1	0.50	-1.50	-1	0.50	7.50	
P 2	-0.25	-0.25	-0.50	0.25	6.75	
2T	-8.75	-3.75	-7.50	6.75	186.25	

CHAPTER XVII

PARAMETRIC METHODS IN QUADRATIC PROGRAMMING

17.1 Postoptimal variation of the right-hand side of a quadratic programming problem

We now discuss the following problem:

Maximise

$$\tau = \tau_o + \underline{w}'\underline{x} + \tfrac{1}{2}\,\underline{x}'D\underline{x} \qquad (17.1.1)$$

Subject to

$$A\underline{x} \leq \underline{b} + \lambda\,\underline{v} \qquad (13.2.1)$$

($\underline{x} \geq 0$ unless otherwise stated)

$$\tau \geq 0 \qquad (17.1.2)$$

The main block of linear side-restrictions (and the parametric variation of the righthand-side of these restrictions), (13.2.1), are the same as those arising in the case of a linear programming problem, and we will make use of the same notation as developed in Chapter XIII.

The non-negativity restriction on the objective function, i.e. (17.1.2) is a novelty which will cause some complications.

These complications are not particularly serious and the restriction of a quadratic objective function to non-negative values falls within the general notion of sensitivity analysis.

The addition of a constant τ_o to the objective function, i.e. (17.1.1), rather than (16.1.1) is related to this non-negativity requirement on the objective function. The non-negativity requirement on $\tau(\underline{x})$ can be made into a redundant restriction, by specifying a large initial value for τ. Our approach to solving the parametric quadratic programming problem is via extending the algorithm which we developed for parametric variation of (the right-hand side of) a linear programming problem to the quadratic case.

Recall section 16.3.

The optimal and feasible vertex of a quadratic programming problem is also a feasible solution to the linear programming problem specified in section 16.3, and modified as follows:

Maximise

$$\mu = \underline{w}'\underline{x} + \underline{b}'\underline{p} \qquad (16.3.1)$$

Subject to

$$D\underline{x} - A'\underline{p} \leq -\underline{w} \qquad (16.2.3)$$

$$A\underline{x} \leq \underline{b} + \lambda\underline{v} \qquad (13.2.1)$$

If we followed the analogy of section 16.3 completely, the optimal and feasible vertex of the adjusted quadratic programming problem would have as objective function of the linear equivalent

$$\mu^{**} = \underline{w}'\underline{x} + (\underline{b}' + \lambda\underline{v}')\underline{p} \qquad (17.1.3)$$

We shall, however, initially be content with the objective function indicated by (16.3.1), as a feasible vertex which also satisfies the complementary slackness condition (i.e. the requirement of standard form), will be sufficient, without regard to optimal form.

As in the LP case, parametric steps may be made either by exogenous adjustment of the righthand-side - a QP generalization of the approach outlined in section 13.1, or the parameter may be treated as a variable and re-defined after having entered the basis - a QP generalization of the procedure outlined in section 13.5 for the LP-case -.

To illustrate the procedure, we slightly modify an example of a quadratic programming problem, which we already presented in section 16.3.

We put the initial value of the objective function at 10, and define the parametric activity as moving the one restriction outwards.

The parametric problem therefore is:

Maximise

$$\tau = 10 - x_1 - x_2 - x_1^2 - x_2^2$$

Subject to

$$-x_1 - 2x_2 \leq -3 + \lambda$$

$$\tau = 10 - x_1 - x_2 - x_1^2 - x_2^2 \geq 0$$

$$0 \leq x_1, x_2 \leq 100$$

As in the linear programming case, the computational implementation is facilitated, if the parametric activity is brought over to the lefthand side, and grouped with the other variables for purposes of updating. This is so, irrespective of the method of actually making the parametric step. The resulting set-up tableau 17.1a is given below, with deletion of the fancyhigh upper limits on x_1 and x_2.

TABLEAU 17.1 A

SET-UP TABLEAU FOR PRIMAL PARA-
METRICVARIATION OF A QP-PROBLEM.

NAME	!		!!	X 1	X 2	L	P 1	!!	VALUE
	!	CODE	!!	1	2	3	-1001	!!	
D 1	!	-1	!!	-2	-	-	1	!!	1
D 2	!	-2	!!	-	-2	-	2	!!	1
D 3	!	-3	!!	-	-	-	1	!!	0
S 1	!	1001	!!	-1	-2	-1	-	!!	-3
2T	!		!!	1	1	-	3	!!	20

The corresponding initial optimum is given below in tableau 17.1b.

TABLEAU 17.1 B

INITIAL OPTIMUM OF THE PRIMAL PARAMETRIC
VARIATION QP PROLEM GIVEN IN TABLEAU 17.1 A

NAME	!		!!	D 1	D 2	L	S 1	!!	VALUE
	!	CODE	!!	-1	-2	3	1001	!!	
X 1	!	1	!!	-0.40	0.20	0.20	-0.20	!!	0.40
X 2	!	2	!!	0.20	-0.10	0.40	-0.40	!!	1.30
D 3	!	-3	!!	-0.20	-0.40	-0.40	0.40	!!	-1.80
P 1	!	-1001	!!	0.20	0.40	0.40	-0.40	!!	1.80
2T	!		!!	-0.40	-1.30	-1.80	1.80	!!	12.90

The dual variable d_3 is negative, indicating that the value of the objective function will actually increase, if the parametric variable is brought into the basis.

However, in a non-linear problem, the value of the objective function is a non-linear function of the value of the parameter.

How non-linear this relationship is, is indicated by the diagonal d_3/λ entry of -2/5.

The dual variable is the first-order derivative of the objective function with respect to λ, and this first-order derivative is itself a function of λ.

If we follow the convention that the parameter is always stored as the last specified variable we may indicate this dual variable as d_n, and the diagonal entry (-2/5 in the d_3/λ cell in this case) as $t_{n,n}$.

In the context of the k^{th} parametric step, these two numbers are related to each other by the following equation

$$d_n(\lambda_k) = d_n - t_{n,n} \cdot \lambda_k \qquad (17.1.4)$$

In (17.1.4), $d_n(\lambda_k)$ is the value of the n^{th} dual slack-variable (shadowprice), expressed as a function of the k^{th} increment of the parameter. The term d_n is the current value of this shadowprice, i.e. -1.80 in the example; $t_{n,n}$ is the n^{th} diagonal cell of the tableau -0.40 in the example.

The shadowprice of the λ-column remains minus the first-order derivative of the objective-function when λ is increased exogenously. Assuming that we are not concerned with the value of the objective, function, the usual search operation indicates λ=2 as the critical value at which x_1 will be eliminated. If the right-hand side is adjusted exogenously, the parametric step results in tableau 17.1c.

TABLEAU 17.1 C

DISPLACED OPTIMUM OF THE PRIMAL PARAMETRIC VARIATION QP PROBLEM GIVEN IN TABLEAU 17.1 A

NAME	CODE	D 1	D 2	L1	S 1	(L=2) VALUE
	CODE	-1	-2	3	1001	
X 1	1	-0.40	0.20	0.20	-0.20	-0.00
X 2	2	0.20	-0.10	0.40	-0.40	0.50
D 3	-3	-0.20	-0.40	-0.40	0.40	-1
P 1	-1001	0.20	0.40	0.40	-0.40	1
2T		0	-0.50	-1	1	???

The usual updating procedure is not applicable to the 2τ-row, if we only changed the 2τ/value cell, the old pseudo-objective function μ (see 16.3.1) would stand. The entries in this row have been obtained by copying from the value column, with the sign-adjustments indicated in section 16.4. The 2τ/value cell, marked ??? in tableau 17.1c is so far unknown.

The fact that the d_3 variable is the shadow-price of the redefined λ-variable throughout the step, may be expressed algebraically by the following equation:

$$\frac{\partial \tau}{\partial \lambda}(\lambda_k) = -\, d_n(\lambda_k) \qquad (17.1.5)$$

Substitution of the right-hand side of (17.1.4) for $d_n(\lambda_k)$ in (17.6.5) gives the following result:

$$\frac{\partial \tau}{\partial \lambda}(\lambda_k) = -d_n + t_{n,n} \cdot \lambda_k \qquad (17.1.6)$$

The use of a partial derivative for $\frac{\partial \tau}{\partial \lambda}$ in both (17.1.5) and (17.1.6) relates to the assumption that other non-basic variables stay at their pre-assigned values of zero, i.e. these relations apply for a particular vertex. Integration of (17.1.6) with respect to λ_k, from zero to a specific value of λ_k, to be indicated as $\bar{\lambda}_k$ yields:

$$\int_o^{\bar{\lambda}_k} \frac{\partial \tau}{\partial \lambda}(\lambda_k)\, d\lambda_k = -d_n \int_o^{\bar{\lambda}_k} d\lambda_k + t_{n,n} \int_o^{\bar{\lambda}_k} \lambda_k d\lambda_k$$

Evaluation of the integrals yields:

$$\tau(\bar{\lambda}_k) - \tau = -d_n\, \bar{\lambda}_k + \tfrac{1}{2}\, t_{n,n}\, (\bar{\lambda}_k)^2 \qquad (17.1.7)$$

Since λ_k is normally used to indicate a specific parametric increment, we can now dispense with the distinction between λ_k (any value) and the $\bar{\lambda}_k$, the position at the next vertex and re-write (17.1.7) as

$$\tau(\lambda_k) = \tau - d_n \cdot \lambda_k + \tfrac{1}{2}\, t_{n,n} \cdot \lambda_k^2 \qquad (17.1.8)$$

We may use (17.6.8), to impose (if so desired) a non-negativity requirement on τ.

We find a zero value of $\tau(\lambda_k)$ for

$$\lambda_k = \frac{d_n \pm \sqrt{d_n^2 - 2\tau \cdot t_{n,n}}}{t_{n,n}} \qquad (17.1.9)$$

The parameter is increased from zero to a positive value, and the parametric step is only made if the initial value of the objective function is non-negative.
(If no non-negativity requirement on τ is intended, (17.1.9) may be made redundant, by specifying a large initial value of τ.)

For $t_{n,n} = 0$, τ is after all a linear function of λ, otherwise the larger of the two roots is positive and applicable, the smaller is negative. Thus λ_k is restricted to the interval between zero and the value indicated by the positive root indicated by (17.6.9).

Since $t_{n,n}$ is non positive (negative), the upper limit on λ_k is, for $t_{n,n} \leq 0$

$$\lambda_k \quad \frac{d_n - \sqrt{d_n^2 - 2\tau \cdot t_{n,n}}}{t_{n,n}} \qquad (17.1.10)$$

The quadratic term in (17.1.8) may in fact be zero, in which case (17.1.9) and (17.1.10) are replaced by simpler linear formulae.

For $t_{n,n} \geq 0$, including the linear case $t_{n,n} = 0$, (17.1.10) is not applicable. For $t_{n,n} \geq 0$, $d_n \geq 0$, there is no upper limit at all, and the parametric search operation may (or may not) become unbounded. For the example at hand, we find:

$$\lambda \leq \frac{-1.80 - \sqrt{(1.80)^2 + 12.90 \times 0.40}}{-0.40} = 11.75$$

We now adjust the initial tableau with respect to two things. We mark the upper limit on λ. Also, we implement the calculation of the first terms of (17.1.8) with respect to 2τ, i.e. two terms of:

$$2\tau(\lambda_k) = 2\tau - 2 \cdot d_n \cdot \lambda_k + t_{n,n} \lambda_k^2 \qquad (17.1.11)$$

We double the $2\tau/\lambda$ entry, from -1.80 to -3.60 and "normal" updating will amend the 2τ entry to become $2\tau - 2.d_n \cdot \lambda_k$. The upper limit is not binding, and the corrected tableau is at

this stage tableau 17.1d.

TABLEAU 17.1 D

INITIAL OPTIMUM OF THE PRIMAL PARAMETRIC VARIATION QP PROLEM GIVEN IN TABLEAU 17.1 A, WITH THE 2T/L CELL SPECIALLY ADJUSTED.

NAME	CODE	D 1	D 2	L	S 1	VALUE
		-1	-2	3	1001	
X 1	1	-0.40	0.20	0.20	-0.20	0.40
X 2	2	0.20	-0.10	0.40	-0.40	1.30
D 3	-3	-0.20	-0.40	-0.40	0.40	-1.80
P 1	-1001	0.20	0.40	0.40	-0.40	1.80
2T		-0.40	-1.30	-3.60	1.80	12.90
BOUND		X	X	11.75	X	X

The quadratic term in (17.1.11) will have to be attended separately. If the parameter is treated as a variable, i.e. an explicit step is made, we now obtain tableau 17.1e.

TABLEAU 17.1 E

NEW VERTEX TABLEAU, NOT IN STANDARD FORM.

NAME	CODE	D 1	D 2	X 1	S 1	VALUE
		-1	-2	1	1001	
L	3	-2	1	5	-1	2
X 2	2	1	-0.50	-2	-	0.50
D 3	-3	-1	-	2	-	-1
P 1	-1001	1	-	-2	-	1
2T		-7.60	2.30	18	-1.80	20.10

Since the upper limit on λ is no longer relevant at this stage, we have suppressed reference to it. Tableau 17.1e contains incorrect entries in the 2τ row. The 2τ/value cell needs to be corrected, i.e. the third, quadratic term of (17.1.11) should be taken into account. For $t_{n,n} = -2/5$ and $\lambda=2$, this term is

evaluated as -1.60 and the 2τ entry in the value column should be 20.10 -1.60 = 18.50.

It is readily verified that for $x_1 = 0$, $x_2 = 0.50$, $\tau = 10 - 0.5 - 0.5^2 = 9.25$, i.e. $2\tau = 18.50$ is indeed the correct entry.

The other cells in the 2τ row will be attended to when the tableau is back in standard form.

We now adjust the prearranged value of the parameter, i.e. integrate $\lambda = 2$ with the value column and put the variable itself re-defined as λ_1 at -0.00.

Having eliminated x_1, d_1 becomes the next pivot column by the complementarity rule.

To maintain ordering we therefore interchange the x_1 and d_1 columns. Since we wish to eliminate the parameter again at the next step, λ becomes badname and stays in the "wrong" slot. The resulting tableau 17.1f is given below.

TABLEAU 17.1 F

NEW VERTEX TABLEAU, ADJUSTED
AND READY FOR THE ORDINARY STEP.

NAME	!		!!	X 1	D 2	D 1	S 1	!!	(L=2) VALUE
	!	CODE	!!	1	-2	-1	1001	!!	
L1	!	3	!!	5	1	(-2)	-1	!!	-0.00
X 2	!	2	!!	-2	-0.50	1	-	!!	0.50
D 3	!	-3	!!	2	-	-1	-	!!	-1
P 1	!	-1001	!!	-2	-	1	-	!!	1
2T	!		!!	18	2.30	-7.60	-1.80	!!	18.50

The normal quadratic programming step is now made, leading to tableau 17.1g.

TABLEAU 17.1 G

NEW OPTIMAL AND FEASIBLE TABLEAU,
WITH AN INCONSISTENT 2T-ROW.

NAME	!		!!	X 1	D 2	L1	S 1	!!	(L=2) VALUE
	!	CODE	!!	1	-2	3	1001	!!	
D 1	!	-1	!!	-2.50	-0.50	-0.50	0.50	!!	0
X 2	!	2	!!	0.50	-	0.50	-0.50	!!	0.50
D 3	!	-3	!!	-0.50	-0.50	-0.50	0.50	!!	-1
P 1	!-	1001	!!	0.50	0.50	0.50	-0.50	!!	1
2T	!		!!	-1	-1.50	-3.80	2	!!	18.50

This is a standard form tableau and it is now possible to correct the 2τ row. We can use the symmetry property (see section 16.4), and copy from the value column, while inverting the sign of the x_2 entry. We also add an upper limit on λ_1, calculated afresh from (17.1.10). The resulting tableau 17.1h is given below.

TABLEAU 17.1 H

NEW OPTIMAL AND FEASIBLE TABLEAU,
READY FOR THE NEXT PARAMETRIC STEP.

NAME	!		!!	X 1	D 2	L1	S 1	!!	(L=2) VALUE
	!	CODE	!!	1	-2	3	1001	!!	
D 1	!	-1	!!	-2.50	-0.50	-0.50	0.50	!!	0
X 2	!	2	!!	0.50	-	(0.50)	-0.50	!!	0.50
D 3	!	-3	!!	-0.50	-0.50	-0.50	0.50	!!	-1
P 1	!-	1001	!!	0.50	0.50	0.50	-0.50	!!	1
2T	!		!!	0	-0.50	-1	1	!!	38.50
BOUND	!		!!	100	X	3.64	X	!!	X

The upper limit on λ is now

$$\lambda_1 \leq \frac{-1 - \sqrt{1 + 18.50 \times 0.50}}{-0.50} = 3.64$$

It will be noted that, with an "ordinary" leaving variable, as distinct from an upper limit distance (e.g. x_1), the exogenous adjustment of the value column and the 2τ-row (e.g. calculating tableau 17.1c and putting the 38.50 in the cell marked "???")

is the simpler method compared with introducing λ into the basis (e.g. calculating tableau 17.1e).

Since λ_1 is limited by the normal rule of the smallest quotient (on the x_2 row), to be not more than 1, a further parametric step can be made, and λ will become 2 + 1 = 3.

Exercise

Continue the example, until at $\lambda = 3$, $x_1 = x_2 = 0$, the parametric adjustment problem becomes unbounded.

17.2 Parametric variation of the linear component of the objective function of a quadratic programming problem

We are now concerned with finding the family of solutions to the problem

Maximise

$$\tau(\underline{x}, \lambda) = \underline{w}'\underline{x} + \lambda\, \underline{g}'\underline{x} + \tfrac{1}{2}\, \underline{x}'D\underline{x} \qquad (17.2.1)$$

Subject to

$$A\underline{x} \leq \underline{b} \qquad (7.2.2)$$

$$(\underline{x} \geq 0)$$

Now recall section (15.6), concerning non-linear duality. We formulate the Lagrangean expression

$$L(\underline{x}, \underline{p}, \lambda) = \underline{w}'\underline{x} + \lambda\underline{g}'\underline{x} + \tfrac{1}{2}\, \underline{x}'D\underline{x} + \underline{p}'(\underline{b} - A\underline{x}) \qquad (17.2.2)$$

and the dual requirements are

$$\frac{\partial L}{\partial \underline{x}} = \underline{w} + \lambda\underline{g} + D\underline{x} - A'\underline{p} \leq 0 \qquad (17.2.3)$$

The expression $L - \underline{x}' \frac{\partial L}{\partial \underline{x}}$ referred to in (15.6.7) turns out to be

$$L(x, \underline{p}, \lambda) - \underline{x}' \frac{\partial L}{\partial \underline{x}} =$$

$$\underline{w}'\underline{x} + \lambda\underline{g}'\underline{x} + \tfrac{1}{2}\, \underline{x}'D\underline{x} + p'\underline{b} - \underline{p}'A\underline{x}$$

$$-\underline{w}'\underline{x} - \lambda\underline{g}'x - x'D\underline{x} \qquad + \underline{x}'A\underline{p} =$$

$$\underline{b}'\underline{p} - \tfrac{1}{2}\, \underline{x}'D\underline{x} \qquad (17.2.4)$$

The dual of a quadratic programming problem therefore is

Maximise

$$\mu = -\underline{b}'\underline{p} + \tfrac{1}{2}\,\underline{x}'D\underline{x} \qquad (17.2.5)$$

Subject to

$$D\underline{x} - A'\underline{p} \leq -\underline{w} - \lambda\underline{g} \qquad (17.2.6)$$

In quadratic programming there is an even closer analogy between the "primal" and the "dual" parametric variation problem, than is the case with linear programming.

We proceed as follows:

Firstly, a parametric restriction, specifying that a certain linear function of the variables is to be greater than or equal to zero, is entered as an (m^{th}) dummy-restriction.

$$\tau^* = \underline{g}'\,\underline{x} \geq 0 \qquad (17.2.7)$$

As in the linear programming case (compare Section 13.5), this "restriction" is initially exempt from search operations. Its associated row serves as an operator. The transposition and inversion of the sign of that row as part of the set-up tableau automatically leads to the generation of a tableau in which we may interpret the dual requirements as the equivalent of (17.2.6), i.e.

$$D\underline{x} - A'\underline{p} + \lambda\underline{g} \leq -\underline{w} \qquad (17.2.8)$$

Example

Maximise

$$\tau^{**} = -x_1 - x_2 - x_1^2 - x_2^2 + \lambda x_1$$

Subject to

$$-x_1 - 2x_2 \leq -3$$

(This is substantially the same example as used in the previous section).

The initial set-up tableau is as follows:

TABLEAU 17.2 A

SET-UP TABLEAU FOR DUAL PARA-
METRIC VARIATION OF A QP-PROBLEM.

NAME			X 1	X 2	P 1	L		VALUE
	CODE		1	2	-1001	-1002		
D 1	-1		-2	-	1	1		1
D 2	-2		-	-2	2	-		1
S 1	1001		-1	-2	-	-		-3
T*	1002		-1	-	-	-		0
2T			1	1	3	-		-

The vector $\underline{g}$ is a unit vector $\underline{e}_1$, i.e. the unity element in position 1.

In the QP case it is in fact practical (though by no means the only possible method to never actually activate the parametric restriction in the sense of making it binding.

The above dual problem shows that we may extend the approach which we followed in section 13.3 for linear programming, to the quadratic case. All that is needed is to properly account for the change in the right-hand side of the dual requirements. The interpretation which we mentioned in section 13.5, i.e. the parametric component of the objective function to attain a certain value is also applicable in both cases.

Using only the convex mode of operation, the similarity between the parametric variation of the right-hand side of the primal, or the dual requirements, is enhanced even more, because in quadratic programming we actually start a quadratic programming step, by bringing a variable into the list of basic variables.

That variable is in the case of parametric variation of (the linear component of) the objective function, the dual variable associated with the parametric restriction. The parametric restriction itself is in fact never activated.

The example's initial optimal tableau, with the first parametric pivot marked ☐ , is given below, in tableau 17.2b.

TABLEAU 17.2 B

INITIAL OPTIMUM OF THE DUAL PARAMETRIC VARIATION QP-PROBLEM, GIVEN IN TABLEAU 17.2 A.

NAME		D 1	D 2	S 1	L	VALUE
	CODE	-1	-2	1001	-1002	
X 1	1	-0.40	0.20	-0.20	-0.40	0.40
X 2	2	0.20	-0.10	-0.40	0.20	1.30
P 1	-1001	0.20	0.40	-0.40	0.20	1.80
T*	1002	-0.40	0.20	-0.20	-0.40	0.40
2T		-0.40	-1.30	1.80	-0.40	-7.10

We do not, in the case of variation of the objective function, implement a non-negativity restriction on the objective function, therefore there is no upper limit on p_2, except the fancyhigh upper limit implied in the basic quadratic programming algorith. (There would be no problem in extending the analogy with variation of the right-hand side to cover this point as well.) We do, however, need a generalization of (17.1.8) to calculate the correct value of the objective function.

$$\tau^{**}(\lambda_k) = \tau^{**} + \tau^{*} \cdot \lambda_k - \tfrac{1}{2}\, t_{m+n,m+n} \cdot \lambda_k^2 \qquad (17.2.9)$$

Here, τ^* is the current value of the parametric component of the objective function (2/5 in the example) and $t_{m+n,\ m+n}$ is the m+n, m+n cell of the tableau. This assumes that the parametric restriction is stored - as in the case of linear programming - as the last restriction. Note the change in sign, compared with (17.1.8). This is because a generalization of (17.1.6), would refer to $\partial\tau/\partial\lambda$ being $+\tau^*$, instead of $-d_n$, as in (17.1.6).

The computational implementation of the parametric variation of the objective function is largely analogous to the procedure discussed in the previous section for the variation of the right-hand side. Thus, the λ entry in the 2τ row is doubled, becoming -0.80. Reference to the $2\tau/\lambda$ cell of the tableau rather than to the value of τ^* automatically covers the difference in the sign of the linear terms of (17.1.8) and (17.2.9). The parametric step is now made, yielding incorrect results for the 2τ-row. Note the difference with section 13.3, where, in the LP case dual parametric steps are made by updating the objective function row only. In a QP problem, a change in the objective

function generally leads to a change in the primal solution vector, and an upper limit distance could be eliminated. We develop tableau 17.2c.

TABLEAU 17.2 C

INTERMEDIATE TABLEAU OF THE DUAL PARAMETRIC VARIATION QP-PROBLEM, NOT IN STANDARD FORM.

NAME	!	!!	D 1	D 2	S 1	X 2	!!	VALUE
	! CODE	!!	-1	-2	1001	2	!!	
X 1	! 1	!!	-	-	-1	2	!!	3
L	!-1002	!!	1	-0.50	-2	5	!!	6.50
P 1	!-1001	!!	-	0.50	-	-1	!!	0.50
T*	! 1002	!!	-	-	-1	2	!!	3
2T	!	!!	0.40	-1.70	0.20	4	!!	-1.90

The correction for the quadratic component turns out to be (for 2τ) $0.40 \times 6.50^2 = 16.90$. We now re-define the objective function, thereby eliminating p_2 ($=\lambda$) without affecting the Lagrangean. The tableau now becomes (after column-reordering):

TABLEAU 17.2 D

RE-DEFINED TABLEAU OF THE DUAL PARAMETRIC QP-PROBLEM, READY FOR THE ORDINARY STEP.

								(L=6.50)
NAME	!	!!	D 1	X 2	S 1	D 2	!!	VALUE
	! CODE	!!	-1	2	1001	-2	!!	
X 1	! 1	!!	-	2	-1	-	!!	3
L1	!-1002	!!	1	5	-2	(-0.50)	!!	-0.00
P 1	!-1001	!!	-	-1	-	0.50	!!	0.50
T*	! 1002	!!	-	2	-1	-	!!	3
2T	!	!!	0.40	4	0.20	-1.70	!!	15

After making the ordinary step, we develop tableau 17.2e.

TABLEAU 17.2 E

NEW STANDARD FORM TABLEAU OF THE
DUAL PARAMETRIC VARIATION QP PROBLEM.

NAME			D 1	X 2	S 1	L1		(L=6.50) VALUE
	CODE		-1	2	1001	-1002		
X 1	1		-	2	-1	-		3
D 2	-2		-2	-10	4	-2		0
P 1	-1001		1	4	-2	1		0.50
T*	1002		-	2	-1	-		3
2T			-3	-13	7	-3.40		15

Tableau 17.2e is a standard form tableau, except for the pseudo-objective function row. To obtain the correct $2\tau^{**}$ row, one needs to re-form this row from the value-column, with the help of the symmetry rules. (The reference to this row as the "2τ" row is in fact incorrect.)

Exercise:

Continue the example, until at $\lambda = 7.00$, $x_1 = 3$, $x_2 = 0$, the parameter-direction becomes unbounded.

17.3 Strict convexity in the parameter subspace

The examples discussed so far have not involved the diagonal cell of the parametric variable itself as pivot.

If we follow the normal QP rules (of the convex mode) as near as possible this would be an acceptable parametric pivot. Whether or not we accept it is basically a question of whether we want the information provided by such a step - the next solution-point is not affected. This point arises in connection with parametric variation of the (primal) righthand-side vector $\underline{b}$.

A parametric variation of the righthand side may be associated with an increase in the value of the objective function, and in the presence of strict convexity in the λ-subspace, this increase (if present) ceases at some point, without change in the collection of binding restrictions.

Example

Maximise $\tau = -x_1^2 - x_2^2 + 15x_1 - 4x_2 + 500$

Subject to $\left.\begin{array}{l} x_1 + x_2 \geq 10 + \lambda \\ x_1 \leq 9 + 3\lambda \end{array}\right\}$ $x_1, x_2 \geq 0$

The initial optimal solution of this problem is $x_1 = 9$, $x_2 = 1$, and the tableau, inclusive of the $x_3 = \lambda$ variable and its dual restriction is the following

TABLEAU 17.3 A

THE SHADOWPRICE OF THE PARAMETER BEING ELEMINATED.

NAME	!!	D 1	D 2	L	S 1	S 2	!!	VALUE
X 1	!!	-	-	-3	-	1	!!	9
X 2	!!	-	-	2	-1	-1	!!	1
D 3	!!	3	-2	[-26]	4	10	!!	-3
P 1	!!	-	1	4	-2	-2	!!	6
P 2	!!	-1	1	10	-2	-4	!!	3
2T	!!	-9	-1	-3	6	3	!!	1098

By the normal rules of the QP algorithm the d_3/λ-cell is the obvious pivot. If λ is then re-defined as being minus zero, and declared to be the badname-variable, the same step is made in opposite direction and the next solution is as given in tableau 17.3b.

TABLEAU 17.3 B

PARAMETRICALLY ADJUSTED SOLUTION, MADE ON INDICATION OF THE OBJECTIVE FUNCTION ONLY.

NAME	!!	D 1	D 2	L1	S 1	S 2	!!	(L=0.12) VALUE
X 1	!!	-	-	-3	-	1	!!	9.35
X 2	!!	-	-	2	-1	-1	!!	0.77
D 3	!!	3	-2	-26	4	10	!!	0.00
P 1	!!	-	1	4	-2	-2	!!	5.54
P 2	!!	-1	1	10	-2	-4	!!	1.85
2T	!!	-9.35	-0.77	-	5.54	1.85	!!	1098.35

This tableau differs from the previous one, only in the righthand-side and the 2τ-row.

If λ is increased further, the d_3 row is no longer eligible as pivotal row, provided the entry for d_3 (written as 0.00), is classified as positive. The dual variable p_2 is then eliminated instead. Had the d_3 row been excluded in the first place, the same result would have arisen in only one parametric step.

It is in fact a question of signalling the point at which the value of the objective function changes sign.

In the case of parametric variation of the objective function, the similar issue arises with respect to the value of the parametric component of the objective function. In the QP case, the incoming variable in the parametric step is selected by the complementarity rule, and there is no need to put any other figure in the τ^* row/value column cell, than the actual value of τ^*. Elimination of this figure by applying the rule of the smallest quotient, then indicates the value of λ, for which τ^* changes sign, from negative to positive.

17.4 Elimination of dual variables during parametric variation

The rules concerning the elimination of dual variables are slightly different when making parametric steps as compared to the normal rules on this point.

This is because the argument raised in Section 16.9, - when increasing the value of a primal driving variable in the presence of a dual badname variable it is efficient to fly through dual restrictions -, is not applicable for a parametric step.

The object of making parametric steps is to obtain information about the solution at suitable values of the parameter, not simply to increase the value of the parameter. Values of the parameter at which exclusion of certain variables from the list of basic variables ceases to be optimal would appear to come under the general heading "suitable" values of the parameter and dual variables will be accepted as leaving variables.

In a convex problem, the strict interpretation of the rule of the smallest quotient, as outlined above, ensures that, if parametric variation is initiated at an optimal and feasible solution, all variables stay non-negative all the time, except for a "parametric zero" entry of -0.00 for one variable at each change of vertex.

However, in a non-convex problem, it may happen that the critical value of λ, at which a particular dual variable ceases to be positive, does not mark a new solution which is convex in some $A_{11}\underline{x}_1 = \underline{b}_1$ subspace, but a local minimum or a saddle-point. It may or may not be possible then to exchange the dual variable in question against the corresponding primal variable, but if we wish to maintain the property of subspace convexity, we must either stop at such a point, or alternatively, breach the dual restriction in question.

This point is best discussed in the context of the "explicit" method of making parametric steps, this permits the generalization of the results of Chapter XVI to the parametric variation problem.

The parameter is the first variable which is brought into the basis in a standard form tableau, i.e. we treat it as the driving variable. By implication, the variable which is complementary to the parameter i.e. the shadowprice of the parameter in the primal parametric variation problem, and the value of the parametric component of the objective function in the case of dual parametric variation, is the badname variable. If the problem were pursued on the lines discussed in section 13.5 for the LP case, the normal search operations of the basic QP algorithm would assign that status to the variables in question in any case.

We may then invoke the driving variable increment qualifier, i.e. a dual variable is not acceptable as leaving variable, if subsequent application of the complementary rule were to lead to the selection of a primal incoming variable which would cause a reduction in the value of the parameter. Then, if this qualifier is accepted, we may be sure that the re-defined parameter can be increased (from -0.00 to exactly zero), if not in the immediately succeeding step, then in the next step, permitting return to standard form in at most two steps.

The term driving variable increment qualifier may not apply if the direct method of making parametric steps by changing the righthand side exogenously is employed, but the need to avoid the development of "improper vertices" where the objective function is not conved in the $A_{11}\,\underline{x}_1 = b$ subspace, exists for both methods of making parametric steps.

Example

Maximise $\quad \tau = x_1^{\,2} - 20\,x_1 + 500$

Subject to $\quad \left.\begin{array}{l} x_1 \geq 1 + \lambda \\ x_1 \leq 12 \end{array}\right\} \; x_1 \geq 0$

Starting at $\lambda = 0$, we note that this problem has two local optima, $x_1 = 1$ and $x_1 = 12$. Between these two local optimal, there is a minimum, at $x_1 = 10$. Normal application of the basic QP algorithm leads to the global optimum, which is $x_1 = 1$. The initial optimum tableau is given below in tableau 17.4a.

TABLEAU 17.4 A

INITIAL OPTUMUM OF A NON-CONVEX PARAMETRIC QP-PROBLEM.

NAME	!!	D 1	L	S 1	P 2	!!	VALUE
X 1	!!	-	-1	-1	-	!!	1
D 2	!!	1	2	2	-1	!!	18
P 1	!!	1	2	2	-1	!!	18
S 2	!!	-	1	1	-	!!	11
2T	!!	-1	18	18	-11	!!	962

Now let us assume that the degeneracy is broken in favour of eliminating p_1, and that this variable is accepted as leaving variable without paying attention to convexity.

If the parametric step is then made by exogenous adjustment of the righthand-side, we develop tableau 17.4b, if the step is made explicitly, we develop tableau 17.4c.

If the usual feature of declaring the zero which is to be eliminated as being a very small negative number is applied here, neither presentation actually permits its elimination. If the entries of -0.00 are replaced by exact zeros which are then classified as positive numbers, and the qualifier is again disregarded, both methods permit return to standard form, developing a "quadratic programming" tableau which describes the local minimum at $x_1 = 10$.

The ambiguous status of dual variables in a non-convex problem gives rise to the concepts of normal, proper, and improper (non-optimal) boundedness (or unboundedness). A parametric variation column associated with a (locally or globally) optimal and feasible solution basis is normally bounded, if we may develop (without violating subspace convexity), for some positive value of the parameter, without further change in the actual value of any primal variable, another (locally or globally) optimal and feasible solution basis, for which the collection of primal basic variables differs from the one in the present tableau either by the exchange of two primal

TABLEAUX 17.4 B AND C

DISPLACED OPTIMUM OF A NON-CONVEX QP PROBLEM, NOT RESPECTING THE DRIVING VARIABLE INCREMENT QUALIFIER.

17.4 B EXOGENOUS ADJUSTMENT.

NAME	D 1	L1	S 1	P 2	(L=9) VALUE
X 1	-	-1	-1	-	10
D 2	1	2	2	-1	-
P 1	1	2	2	-1	-0.00
S 2	-	1	1	-	2
2T	-10	-	-	-2	800

17.4 C EXPLICIT STEP.

NAME	D 1	P 1	S 1	P 2	(L=9) VALUE
X 1	0.50	0.50	-	-0.50	10
D 2	-	-1	-	-	-
LA	0.50	0.50	1	-0.50	-0.00
S 2	-0.50	-0.50	-	0.50	2
2T	-10	-9	-	-2	800

variables, or by one primal variable leaving or entering the basis. The new optimal and feasible solution, thus identified is then associated with a normal successor tableau. If the leaving variable identified by the parametric variable is a primal variable, and if that primal variable cannot be eliminated because the problem becomes empty at that value of the parameter, the parametric column is said to be bounded by the end of the feasible space area.

(In the example this is the case for $\lambda = 11$, s_2 becomes negative, but cannot be eliminated, x_1 being already at its maximum value of $x_1 = 12$.)

Boundedness at the end of the feasible space area gives rise to an empty end of algorithm tableau. In the example given above, the empty end of algorithm tableau would be either tableau 17.4d or tableau 17.4e, depending on the method of making the parametric step.

TABLEAUX 17.4 D AND E

EMPTY END OF ALGORITHM TABLEAUX.

17.4 D EXOGENOUS ADJUSTMENT

NAME	!!	D 1	L1	S 1	P 2	!!	(L=11) VALUE
X 1	!!	-	-1	-1	-	!!	12
D 2	!!	1	2	2	-1	!!	-4
P 1	!!	1	2	2	-1	!!	-4
S 2	!!	-	1	1	-	!!	-0.00
2T	!!	-12	-4	-4	0	!!	808

17.4 E EXPLICIT STEP METHOD

NAME	!!	D 1	S 2	S 1	P 2	!!	(L=11) VALUE
X 1	!!	-	1	-	-	!!	12
D 2	!!	1	-2	-	-1	!!	-4
P 1	!!	1	-2	-	-1	!!	-4
L1	!!	-	1	1	-	!!	-0.00
2T	!!	X	X	X	X	!!	X

The computational implementation offered here would normally supply the equivalent of tableau 17.4d, but if the leaving variable in the parametric search operation is an upper limit distance, the equivalent of tableau 17.4e would be supplied, the technical cause of the end of the algorithm being that the p_2 column activated by the complementary rule, is unbounded. (The 2τ row has been marked with X entries as this row would contain invalid entries on account of the coefficient doubling device).

If an empty end algorithm tableau describes a solution vector where all dual requirements are satisfied, the corresponding parametric column is said to be properly bounded by the end of the feasible space area.

If the empty end of algorithm tableau describes a solution which violated one of more dual restrictions (e.g. as is the case with the p_1 restriction in the example), the corresponding parametric column in the preceding tableau is said to be improperly (or non-optimally) bounded by the end of the feasible space area.

The notion of proper and improper (non-optimal) boundedness

may obviously be extended to non-empty successor tableaux and indeed to unbounded columns. A parametric column is improperly bounded by a non-empty successor solution if it is not normally bounded but if some other primal feasible solution which is convex in some $A_{11}\ \underline{x}_1 = \underline{b}_1$ subspace exists, (which differs in only one variable from the current one, in the same way as defined above for normal boundedness), but which breaches one or more dual restrictions.

In a convex problem, there is never any need to breach dual restrictions it therefore follows that, in a convex problem, boundedness implies proper boundedness.

The similar distinction also applies to unbounded parametric columns. If all entries in primal and dual variables' rows are non-positive (and, as explicitly written upper limit restrictions are obtained by turning the signs around), zeros in all rows associated with bounded variables), the parametric column is unbounded. If there are positive entries in one or more dual variables rows, but these dual variables cannot be eliminated on account of the qualifier, we have an improperly unbounded column.

A solution vector at which a dual variable changes sign, but where it is not possible to introduce the corresponding primal variable at zero value without violating subspace convexity might be called an improper vertex. (An improper vertex is therefore a solution basis which describes a local minimum or a saddle-point).

It is obviously possible to activate the basic QP algorithm just beyond an improper vertex (e.g. in tableau 17.4b one would re-enter with $\lambda = 9$ and select p_1 as badname-variable).

The typical parametric feature of exchanging the solution basis without actually changing the primal solution vector at the point where one changes basis, would not, however apply in that case.

17.5 Parametric equivalence and the number of steps on parametric re-entry

We have so far assumed that the normal way of making parametric steps is to change the solution vector during the parametric variation step, and then to make one ordinary step which merely effectuates a change in the solution basis without actually causing any change in the solution vector.

In fact two ordinary steps may be needed besides the parametric variation step, and one of these ordinary steps may involve a

change in the dual solution vector even where neither of the two causes any further change in the primal solution vector.

We now state four alternative methods of making parametric steps and their relationship to each other, somewhat more formally, as follows:

Method 1

Bring the parameter into the solution-basis, treating it as the driving variable. (Exit if the parameter is found unbounded). Select leaving variables by the rule of the smallest quotient, subject to the driving variable increment qualifier, and incoming variables by the complementarity rule. Re-define the value of the parameter to $\lambda_k = -0.00$ immediately after its entry into the solution basis, and exit as soon as the parameter has again been eliminated.

Method 2

Change the right-hand side exogenously, until one variable, which is also acceptable as leaving variable under method 1, ceases to be non-negative, now being observed as being -0.00 instead. Then eliminate the now negative-valued variable either by one step with a negative pivot on the main-diagonal, or by the application of a complementary pair.

The first incoming variable of the pair is the complement of the parametrically eliminated variable, and the corresponding leaving variable is selected by the rule of the smallest quotient, but the driving variable increment qualifier is replaced by a verification of the applicability of the complementary pair situation.

Method 3 (not recommended)

Change the righthand-side as in method 2, and find a new optimal and feasible solution by re-entering the basic QP algorithm in the convex mode of operation. Exit on return to standard form.

Method 4

As method 3, but use the non-convex mode of operation.

All four methods have the search operation for the parametrically eliminated leaving variable in common, and if none is found all four versions of the algorithm terminate.

In a convex problem, this alarm exit obviously corresponds to either an unbounded parametric variation problem (in the case

of primal parametric variation), in the same way as discussed in section 13.2 for the LP case or, (with an unbounded shadowprice of the parametric restriction), to having attained the maximum value of the parametric component of the objective function. No statement concerning the non-existence of solutions for higher values of λ, is, however, applicable in the case of an improperly unbounded parametric column, although such a solution - if existing - would not be obtainable by any of the methods listed under 1, 2, 3 or 4.

The following properties concerning the 4 stated methods apply therefore under the prerequisite that the parametric column is bounded (properly bounded or improperly bounded). We also exclude the complement of the parameter as leaving variable, although it should be remembered that this case, as discussed in section 17.3, exists and presents no particular problems.

In the interest of concise formulation the actual formulation of the relevant properties is preceded by a tableau summary which refers to the results of the parametric search operation: We first consider that search operation as such

Name	com	λ	Value
plv	d	[p]	v

We shall use the abbreviation plv("Parametric leaving variable") to refer to the variable which is driven to zero by the parametric variation step, and com ("complement") for its associated complementary variable; the current value of plv variable is indicated as v; p is the pivot found in the parametric search operation; d is the diagonal cell on the intersection of the plv-row and the com-column. Note that these abbreviations will be used in relation to all four methods, even though they derive their significance mainly from their role in method 1.

We may distinguish two cases, $d = 0$, and $d < 0$. The case $d > 0$ does not arise; if plv is a primal variable it does not exist at all, as it would contradict the subspace convexity properties of the current solution basis, if plv is a dual variable its selection as leaving variable is prohibited by the driving variable increment qualifier.

We shall refer to the case $d < 0$, as "the (negative) definite case" and the case $d = 0$, as "the semi-definite case".

We now state some properties concerning the relations between the four algorithms that were described.

Equivalence property concerning the negative definite case

In the negative case, d < 0, all four methods effectuate a return to a standard form successor solution-basis, in just one step, following the parametric variation-step.

(The proof of this statement is obvious; the one pivot is d/p in method 1, eliminating λ, and d in method 2, 3 and 4 eliminating plv).

Before stating a similar property relating to the semi-definite case, it is useful to give some further tableau-summaries.

Name	com	oiv	λ	Value
plv	o	-a	[p]	v
olv	a	c	?	?

semi-definite tableau on entry to the parametric step

Name	com	oiv	plv	Value
λ_k	o	-a/p	1/p	-0.00
olv	a	?	?	?

semi-definite tableau in exit from the parametric variation step by method 1

Name	com	oiv	λ	Value
plv	o	-a	p	-0.00
olv	a	c	?	?

semi-definite tableau exit from the parametric variation step, by method 2, 3 or 4

Here olv ("ordinary leaving variable") is the variable selected as leaving variable by the application of the rule of the smallest quotient, in the first step after the parametric step, following method 1. oiv ("ordinary incoming variable") is the variable which is thereupon activated by the complementarity rule.

Note that the fully symmetric arrangement

Name	com	oiv	λ	Value
plv	o	a	[p]	v
olv	a	c	?	?

i.e. assuming the parametric and the ordinary leaving variable to be of the same class, does not apply. It would contradict the driving variable parameter theorem. (Recapitulation of the proof on that point: the assumption that both plv and olv are

primal variables contradicts the semi-definite case; the assumption that both are dual variables is wrong because the qualifier inhibits that choice of pivots, i.e. olv would be refused).

The very writing of the applicable summaries for the semi-definite case assumes that a further step by method 1 can be made.

This is not a trivial condition, because primal parametric variation may result in the formulation of an empty problem, emptiness becoming apparent when a primal plv variable - or the parameter - has the value -0.00, and cannot be increased to exactly zero. Method 1 will then exit with the com variable being found unbounded, the boundedness proof of section 16.10 does not apply because the variable which is technically the badname-variable is not eligible as leaving variable. Methods 2 and 4 will react to emptiness by not finding a primal incoming variable.

However, the following statement is true:

Equivalence property for the semi-definite case;

If the parametric variation step has been made, and has not resulted in the formulation of an empty problem, while the semi-definite case is applicable, then irrespective of the convexity or non-convexity of objective function methods 1, 2 and 4 each find a new solution basis in two ordinary steps (following the parametric variation step). Of these two steps only one involves a non-trivial search for a smallest ratio, the other is of zero length and involves either a trivial search (in method 1), or no search at all (the second step of a pair in methods 2 and 4).

In the absence of degeneracy (other than the parametric pseudo zero), all three methods lead to the same successor solution. This statement appears to follow almost immediately from the summaries, but the significance point is the selection of the same olv. variable by all three methods.

The search for the smallest dual ratio (methods 2 and 4) involves actually the same numbers as the search for the smallest quotient in method 1. This is because the parametric variation step leaves the com variable's column unaffected, insofar as its non-zero elements are concerned.

For a non-convex problem, the new solution-basis may be non-optimal, if the driving variable increment qualifier became active in the parametric variation step, but the equivalence

applies to improper boundedness as well. (This point has been discussed already in the section 17.4).

But even if optimal form is maintained, the equivalence property does not necessarily apply to method 3. This is because, under method 3 the driving variable increment qualifier is applicable with respect to the com variable as driving variable, and for $c > 0$, this may inhibit the activation of the complementary pair used in method 2. This will generally mean that a different local solution is reached, and it may take more than 2 steps to get there.

Example

Maximise $\tau = x_1^2 + x_2^2 - 20\,x_1 - 15\,x_2 + 500$

Subject to $x_1 + x_2 \geq 1 + \lambda$

$x_1 \geq x_2 - 1 + \lambda$

$0 \leq x_1 \leq 100; \quad 0 \leq x_2 \leq 2. \quad (\lambda \leq 100)$

A local initial optimum ($x_1 = 0$, $x_2 = 1$) is developed by normal application of the basic QP algorithm, this solution is summarized in tableau 17.5a below.

TABLEAU 17.5 A

ILLUSTRATION OF THE BREAKDOWN OF METHOD 3.

NAME	!!	X 1	D 2	LA	S 1	P 2	!!	VALUE	DIST
D 1	!!	4	-1	-2	-2	2	!!	7	X
X 2	!!	1	-	-1	-1	-	!!	1	1
D 3	!!	-2	1	2	2	-2	!!	13	X
P 1	!!	-2	1	2	2	-1	!!	13	X
S 2	!!	-2	-	2	1	-	!!	-	X
2T	!!	7	-1	13	13	-	!!	972	X
BOUND	!!	100	X	100	X	X	!!	X	X

If the zero in the s_2 row/value column cell is parametrically adjusted to -0.00 and the convex mode is activated (method 3), the p_2 column is bounded only by its artificial upper limit, the d_1 restriction needs to be breached on account of the driving variable increment qualifier.

In this small example, we then still get at the same solution in two steps, but we must obviously consider the possibility that the dual restriction associated with another non-basic variable is eliminated at this point, leading to a non-optimal successor

tableau, i.e. in effect selecting the wrong complementary pair.

Methods 2 and 4 on the other hand, both implement the complementary pair which exchanges x_1 against s_2, and interchanges their associated dual variables at the next step.

There is, in fact no meaningful distinction between methods 2 and 4, at least not if we assume that the previously existing solution basis was an optimum, method 4 will activate the complementary pair mentioned for method 2.

In activating the non-convex mode in the reentry-mode it is obviously necessary that one verifies whether the complementary situation is applicable; the "ordinary" QP code of section 16.15 contains that provision already.

Note also that method 1 is equivalent to methods 2 and 4, despite the fact that it calls for verifying the driving variable increment qualifier at the stage of searching for a leaving variable in the first ordinary step.

This is also apparent from the summaries given above: the qualifier merely serves to exclude the selection of a second dual variable as ordinary leaving variable (with the plv variable already being a dual variable) and is otherwise always satisfied.

The computational implementation offered in this chapter uses method 1 if the parametrically eliminated variable is an upper limit distance, otherwise method 4.

17.6 Parametric solution methods

Parametric methods can be used to solve quadratic programming problems in the first place.

In the computational context offered here this is not a practical proposition, if only because parametric steps are made by way of making ordinary steps. The topic has, however, had a certain amount of attention in the literature in particular by van de Panne, and it provides certain useful interpretations.

The two main issues are the representation of the driving variable as a parameter, and the artificially optimal and feasible starting solution.

The first issue is obviously related to the driving variable parameter theorem.

The driving variable parametric equivalence method (or, as van de Panne [35] Section 9.2), would say, the symmetric variant of the simplex method for Quadratic programming), may be summarized as follows:

We may select a badname-variable and a driving variable - by any criterion -. Instead of bringing in the driving variable as a basic variable we specify a parametric activity the coefficients of which are the same as for the selected driving variable (except for a change in sign if the parameter is presented on the right-hand side). We then make parametric steps until the badname variable has been eliminated. We must obviously restrict the method to those cases where the driving variable can be shown to be bounded - and then exchange the driving variable itself for the parameter.

Depending on the precise version of the convex mode of operation which is used, we may develop the same solutions as in the method outlined in Chapter 16. It is in fact a different presentation of the tableaux.

This is illustrated below in the tabulation 17.6a. The parameter has been presented on the right-hand side of the equation sign, the entries in the parameter column therefore have the same absolute value, but are of the opposite sign as the corresponding entries in the driving variable's column in the same tableau.

The one exception to that property is the 2τ-row/λ column entry, where proportionality with the incoming variable column cell in the direct tableaux has been maintained.

Except in the second step of a parametric pair, the incoming variable in the direct presentation of the tableau is proportional to the parametric column. Both columns indicate the change in the solution vector, but the parametric column gives the change per unit of increase of the driving variable, the ordinary incoming variable per unit of the incoming variable. (The reason that this property is not automatically maintained in the 2τ-row of the parametric tableaux, is that the 2τ-row is re-built at each step from the value column.)

A computational disadvantage of the driving variable parametric equivalence method as a computationally applied method is that pivots are taken on the main diagonal whenever a negative non-zero entry occurs in the diagonal cell of the leaving variable's row, irrespective of the absolute value of this cell. That may not be a particularly obvious drawback in the example given, but the issue of near zero pivots arises.

TABLEAU 17.6 A

ILLUSTRATION OF DRIVING VARIABLE PARAMTERIC EQUIVALENCE.

DIRECT TABLEAUX

NA	X1	X2	P1	VAL
D1	-2	-	1	1
D2	-	-2	(2)	1
S1	-1	-2	-	-3
2T	1	1	3	-

NA	X1	D2	X2	VAL
D1	-2	-0.5	(1)	0.5
S1	-1	-	-2	-3
P1	-	0.5	-1	0.5
2T	1	-1.5	4	-1.5

NA	D1	D2	X1	VAL
S1	2	-1	(-5)	-2
X2	1	-0.5	-2	0.5
P1	1	-	-2	1
2T	-4	0.5	9	-3.5

NA	D1	D2	S1	VAL
X1	-0.4	0.2	-0.2	0.4
X2	0.2	-0.1	-0.4	1.3
P1	0.2	0.4	-0.4	1.8
2T	-0.4	-1.3	1.8	-7.1

PARAMETRIC TABLEAUX

NA	X1	X2	P1	L=0 VAL	L	L=0.5 VAL
D1	-2	-	1	1	-1	0.5
D2	-	(-2)	2	1	[-2]	-0.0
S1	-1	-2	-	-3	-	-3
2T	1	1	3	-	-3	-1.5

NA	X1	D2	P1	L=0.5 VAL	L1	L=1 VAL
D1	(-2)	-	1	0.5	[-1]	-0.0
X2	-	-0.5	-1	0	1	0.5
S1	-1	-1	-2	-3	2	-2
2T	0.5	0.0	3	-2	-4	-3.5

NA	D1	D2	P1	L=1 VAL	L2	L=1.8 VAL
X1	-0.5	-	-0.5	0.0	0.5	0.4
X2	-	-0.5	-1	0.5	1	1.3
S1	-0.5	-1	(-2.5)	-2	[2.5]	0.0
2T	0.0	-0.5	2	-3.5	-4.5	-7.1

NA	D1	D2	S1	L=1.8 VAL	L3	L=0 VAL
X1	-0.4	0.2	-0.2	0.4	-	0.4
X2	0.2	-0.1	-0.4	1.3	-	1.3
P1	0.2	0.4	-0.4	0.0	-1	1.8
T	-0.4	-1.3	1.8	-7.1	-	-7.1

In particular, if the solution is in a corner of the feasible space area, and a primal variable is eliminated by a primal driving variable, mistaking a rounding error for a non-zero number may totally destroy the accuracy of the tableau.

Example

Maximise $\tau = 2x_1 + 4x_2$

Subject to $3x_1 + 2x_2 \leq 11,\ 2x_1 + 3x_2 \leq 4 \qquad (x_1, x_2 \geq 0)$

This is in reality an LP problem.

Now suppose we first exchange x_1 against s_2, - by whatever method, parametric or not -, and then parametrically introduce x_2 into the basis, in fact against x_1 as leaving variable. This is not an efficient path anyhow, the assumption merely serves to illustrate a potential risk of the parametric method. The resulting situation is summarized in tableau 17.6b.

TABLEAU 17.6 B

CHOICE OF AN UNDESIRABLE DIAGONAL PIVOT.

NAME	!!	D 1	X 2	P 1	S 2	!!	LA=0 VALUE	LA	LA=1.33 VALUE
X 1	!!	(-0.00)	1.50	-	0.50	!!	2	[-1.50]	-0.00
D 2	!!	-1.50	-	2.50	-	!!	-1	-	-1
S 1	!!	-	-2.50	-	-1.50	!!	5	2.50	1.67
P 2	!!	-0.50	-	1.50	-	!!	1	-	1
2T	!!	-2	-1	-5	1	!!	8	2	10.67

A computational implementation of the driving variable parameter method would therefore need some special cross-check on the solution-structure, i.e., whenever a corner solution is developed (the number of elements of $\underline{x}$ which have entered the basis, being equal to the number of elements of $\underline{s}$ which have left the basis), one would replace all intersections of primal variable's rows with dual variable's columns by exact zeros.

Van de Panne does not, however, put the emphasis on this method for computational reasons - he provides his proofs and theory in terms of the symmetric method, and then shows that the a-symmetric case is equivalent; we came the other way.

Another parametric solution method, developed independently by van de Panne [35] , section 11.2, and by Goncalves [17] might be called the artificial righthand-side method. In this

method, one superimposes arbitrary positive numbers on the negative elements of the righthand side, and parametrically moves from the artificial solution (which is therefore optimal and feasible at the outset), to the specified right-hand side.

The same drawback as mentioned above for the driving variable parameter method, also applies to this method.

17.7 Computational implementation of parametric QP

As in the LP case no full listing of the parametric version of the quadratic programming procedure is given; this would not be justified in view of the large overlap with the listing in section 16.15.

We also observe that, although it was possible to generate some example tableaux in section 17.6, the procedure offered here does not cater for parametric solution methods as such.

We now give a listing of the file of editing instructions which converts the procedure listed in section 16.15 into an equivalent procedure which caters for parametric re-entry, and a corresponding calling main programme.

```
TEXT-LISTING OF THE QUAD/QUAP EDITOR-FILE.

TC/'PROCEDURE' QUAD(/,R/QUAD/QUAP/
T.?REENTRY?,I?PR,PC,???,T.?)?,I?,LAMBDA?
TC/'INTEGER'/
I?'VALUE' NNEGD;
?
T.?REENTRY?,I?PR,PC,?
T.E,I???'REAL' LAMBDA;?
TC/'BEGIN'/,TC?;?,T.?;?,I?,NAC?
TS/'REAL'/,T.?;?,I?,TWOT?
TS/'BOOLEAN'/,T.?CONVEX?,I?PARAMETRIC STEP,?
T.?;?,I?,
POST PARAM ST?
TS/'COMMENT'/
T1,PC/;/
I?  QUADRATIC PROGRAMMING,WITH POSTOPTIMAL VARIATION.

  FOR AN OUTLINE AND DESCRIPTION OF THE MAIN ALGORITHM,
  SEE THE QUAD PROCEDURE.

  THE DIFFERENCES BETWEEN QUAP AND QUAD ARE AS FOLLOWS:
```

```
  PR AND PC STAND FOR PARAMETRIC ROW AND PARAMETRIC COLUMN.
  FOR PC=1 AND PR=0 AN EXTRA COLUMN, THE PARAMETRIC COLUMN
  IS INCLUDED.
  FOR REENTRY=0, THE PARAMETRIC VARIABLE IS NOT INCLUDED IN
  THE SEARCH OPERATIONS,
  BUT A PARAMETRIC STEP MAY BE MADE, BY SUBSEQUENTLY
  ENTERING THE PROCEDURE WITH REENTRY=2.
  FOR PR=1 AND PC=0, THE SAME APPLIES FOR THE PARAMETRIC
  RESTRICTION, WHICH REPRESENTS PARAMETRIC VARIATION IN
  THE LINEAR COMPONENT OF THE OBJECTIVE FUNTION.

  FOR PR=PC=REENTRY=0, THE ACTION OF QUAP IS THE SAME AS
  THE ACTION OF OF QUAD.

  THE VARIABLE NAC IS AN AUXILIARY VARIABLE,
  INDICATING THE NUMBER OF ACTIVE COLUMNS, WHICH ARE
  TO BE INCLUDED IN THE SEARCH OPERATIONS.
?
TS/FANCYHIGH:=/
I?  PARAMETRIC STEP := 'FALSE'; CONVEX := 'TRUE';
  POST PARAM ST := 'FALSE'; NAC:=N-PC;

?
TS/'IF' REENTRY=1/,T.?REENTRY?,I?'NOT' ?,R/1/0/
TS/QUADRATIC/,T.?RE-ENTRY?,I?NORMAL ?
TS/ADJUSTMENT/,R/./,/,T1,PC/;/,P1
I?     AND AS TWO IF PARAMETRIC REENTRY IS ASKED FOR.
    ;

    ENSURE CONVEX MODE IN DUAL PARAMATRIC LOOP:
    'IF' REENTRY=2 'AND' PR=1 'THEN' NNEGD:=M+N+1;
    'IF' REENTRY=2 'THEN' PARAMETRIC STEP := 'TRUE';
    'IF' REENTRY=1 'THEN' 'GOTO' START;

    'IF' PR=1 'THEN' 'BEGIN'
      MARK DUAL OF PARAMETRIC ROW:
      BADN := 1000+M; COLN := -1000-M;
      QUO := 10000000000000;  ROWN := 0;
      K:=B:=N+M;
      ATTEND DUAL LA DERIVATIVE OF 2T:
      T[N+M+1,K]:=2*T[N+M+1,K];
      'GOTO' PREPARE INV OF MODE; 'END';

    'IF' PC=1 'THEN' 'BEGIN'
      MARK PARAMETRIC COLUMN VARIABLE:
      BADN := -N;  COLN := N;
      QUO := 10000000000000;  ROWN := 0;
      D:=K:=B:=N;

      ATTEND PRIMAL LA DERIVATIVE OF 2T:
      T[M+N+1,K] := 2*T[N+M+1,K];

      ACCEPT PRESET UPPER LIMIT:
      'IF' T[N+M+2,K] > 0
      'THEN' 'GOTO' PREPARE INV OF MODE;
```

```
      SET UPPER LIMIT ON PARAMETER:
      'IF' T[N,N]>0 'AND' T[N,N+M+1]>0 'THEN' 'BEGIN'
        T[N+M+2,K] := 1 000 000;
        'GOTO' PREPARE INV OF MODE; 'END';
      'IF' T[N,N] < 0 'THEN'
      T[N+M+2,K] := T[N,N+M+1]/T[N,N] -
      SQRT(T[N,N+M+1]*T[N,N+M+1]-T[N,N]*T[N+M+1,N+M+1])
      /T[N,N]
      'ELSE'
      T[N+M+2,K]:=0.5*T[N+M+1,N+M+1]/T[N,N+M+1];
      'IF' T[N+M+2,K] < 0 'THEN' T[N+M+2,K]:=1000000;
      'GOTO' PREPARE INV OF MODE; 'END';

?
TS/COMPLETE TABLEAU:/
I?  INITIATE LAMBDA:
  'IF' 'NOT' PARAMETRIC STEP 'THEN' LAMBDA := 0;

?
TS/SELECT BADN/
T1,I?  'IF' (PC=1 'AND' ROWLST[N]=N) 'THEN' NAC:=N;
?
TS/'FOR' I:=/,P1
I?  'FOR' I:=1 'STEP' 1 'UNTIL' NAC,
  N+1 'STEP' 1 'UNTIL' N+M-PR 'DO' 'BEGIN'

?
TS/ABS VAR ST/
I?    'IF' PARAMETRIC STEP
    'THEN' 'GOTO' END OF BADNAME SELECTION LOOP;

?
TC/END OF QUAD/
I?  'IF' PARAMETRIC STEP
  'THEN' 'GOTO' TRY THE BADNAME OR PAR ROW;

?
R/OF QUAD/OF QUAP/
TS/'FOR' I:=NAV+1/
R/'UNTIL' N/'UNTIL' NAC/,TC/N+M/,R/N+M/N+M-PR/
TS/PHASE I SEARCH:/
TS/'FOR' I/,P2
I?  'FOR' I:=NAV+1 'STEP' 1 'UNTIL' NAC,
  N+1 'STEP' 1 'UNTIL' N+M-PR 'DO'
?
TS/PHASE I:/
TS/'FOR' J:=/,P1
I?  'FOR' J:=1 'STEP' 1 'UNTIL' NAC,
  N+1 'STEP' 1 'UNTIL' N+M-PR 'DO'

?
TC/END OF QUAD;/,R/OF QUAD/OF QUAP/
TS/PHASE II:/,T2
I?  'IF' POST PARAM ST 'THEN' 'BEGIN'
    'IF' PC=1 'THEN' D:=N 'ELSE' D:=N+M; 'END';
?
```

```
TC/TRY THE BADNAME/,R/BADNAME/BADNAME OR PAR/
TC/-0.000/
T.?'THEN' 'BEGIN'?
I?'AND' 'NOT' T[B,N+M+1]>0???
I?  ?

TS/SEEK SMALLEST QUO/,TS/'FOR' I/
P1,I?  'FOR' I:=1 'STEP' 1 'UNTIL' NAC,
  N+1 'STEP' 1 'UNTIL' N+M-PR 'DO'
?

TS/DO I NEED TO ELEMINATE:/
TS/'IF' BADN < 0 'THEN' 'GOTO' DONE/
T.?'THEN'?,I?'AND' REENTRY < 2 ?
TS/UPDATE VECTORS:/
I?     ATTEND 2T AND LAMBDA IN UB PARSTEP:
     'IF' PARAMETRIC STEP 'THEN' 'BEGIN'
       'IF' PC=1 'THEN'
       T[N+M+1,N+M+1]:=T[N+M+1,N+M+1]+QUO*QUO*T[K,K]
       'ELSE'
       T[N+M+1,N+M+1]:=T[N+M+1,N+M+1]-QUO*QUO*T[K,K];
       LAMBDA:=LAMBDA+QUO; 'END';

?
TS/STANDARD FORM DOUBLE STEP:/
TC/'FOR'/
TS/'IF' ROWN < 0/,T.?'THEN'?
I?'AND' 'NOT' (PR=1 'AND' ROWN=-3000-M)?,P.E
T1,T.?T?
I?'THEN' ?,T.E,I?;?,T1
R/'ELSE'//
I?'IF' ROWN>0 'AND' 'NOT' (PC=1 'AND' ROWN=2000+N)?
I?
       'THEN'?
TS/'IF' 'NOT' ABS(/
I?       'IF' PARAMETRIC STEP 'AND' PC=1 'AND' ROWN=2000+N
       'THEN' 'GOTO' END OF QUAP;

       'IF' PARAMETRIC STEP 'AND' PR=1 'AND' ROWN=-3000-M
       'THEN' 'GOTO' END OF QUAP;

?
TS/PERMUTE COLUMNS:/
I?  ATTEND PARAMETRIC REENTRY:

  'IF' PARAMETRIC STEP 'THEN' 'BEGIN'
    'IF' PC=1 'THEN'
    T[N+M+1,N+M+1]:=T[N+M+1,N+M+1]+QUO*QUO*T[B,B]
    'ELSE'
    T[N+M+1,N+M+1]:=T[N+M+1,N+M+1]-QUO*QUO*T[B,B];
    LAMBDA:=LAMBDA+QUO;

    PARSTEP BY VECTORS ONLY:
    'IF' (ABS(ROWN) < 2000 'AND' 'NOT' ROWN=BADN
    'AND' T[R,R] < -0.000001 )
    'OR' (ROWN>0 'AND' ROWN=ROWLST[R])
    'THEN' 'BEGIN'
      'FOR' I:=1 'STEP' 1 'UNTIL' N+M+1 'DO'
      'IF' 'NOT' T[I,K]=0 'THEN' 'BEGIN'
        T[I,N+M+1]:=T[I,N+M+1]-T[I,K]*QUO;
        T[I,N+M+2]:=T[I,N+M+2]+T[I,K]*QUO; 'END';
      T[R,N+M+1] := -0.000000001;
```

```
      'FOR' J:=1 'STEP' 1 'UNTIL' N+M 'DO'
      'IF' ROWLST[J] < 0 'THEN' T[N+M+1,J]:=T[J,N+M+1]
      'ELSE' T[N+M+1,J]:=-T[J,N+M+1];

      PARAMETRIC STEP := 'FALSE';  REENTRY := 1;
      T[N+M+2,K] := T[N+M+2,K] - QUO;
      BADN:=ROWN; B:=R;
      'IF' BADN>0 'AND' T[B,B] > -0.000001 'THEN' 'BEGIN'
        T[B,B]:=0; 'GOTO' PHASE I; 'END';
      'GOTO' PHASE II; 'END';
    'END';

?
TS/CHECK FOR STATUS:/
T1
I?
  'IF' PARAMETRIC STEP 'AND' PC=1 'AND' ROWN=-N
  'THEN' 'BEGIN'
    T[N,N+M+1]:=-0.0000001; BADN:=N; COLN:=-N; K:=R:=B:=N;
    PARAMETRIC STEP := 'FALSE';
    'GOTO' TRY THE BADNAME OR PAR ROW;  'END';

  TERMINATE PARAMETRIC REENTRY:
  'IF' (PC=1 'AND' ROWN=N) 'OR' (PR=1 'AND' ROWN=-1000-M)
  'THEN' 'GOTO' END OF QUAP;
?
TS/ATTEND PAIR IN NON CONVEX MODE:/
TS/'IF' 'NOT' CONVEX MODE/,T.?'THEN' 'BEGIN'?
I?
  'AND' 'NOT' PARAMETRIC STEP ?
TS/PREPARE NON STANDARD FORM STEP:/,T1
I?
  'IF' PARAMETRIC STEP 'THEN' 'BEGIN'
    'FOR' J:=1 'STEP' 1 'UNTIL' N+M+2 'DO' 'BEGIN'
      COP:=T[B,J]; T[B,J]:=T[R,J]; T[R,J]:=COP; 'END';
    NAME:=ROWLST[R]; ROWLST[R]:=ROWLST[B]; ROWLST[B]:=NAME;
    PARAMETRIC STEP := 'FALSE';
    POST PARAM ST := 'TRUE';
    TWOT := T[N+M+1,N+M+1];
    REENTRY:=1; BADN:=COLN; B:=R;
    T[R,N+M+1]:=-0.0000001; 'END';
?
TC/END OF QUAD:/,R/END OF QUAD/END OF QUAP/,T1
TC/FINAL END OF QUAD/
I?  'FOR' I:=1 'STEP' 1 'UNTIL' N+M 'DO'
  'IF' ROWLST[I]<0 'THEN' T[N+M+1,I]:=T[I,N+M+1]
  'ELSE' T[N+M+1,I]:=-T[I,N+M+1];
  'IF' PR=1 'OR' PC=1 'THEN' 'BEGIN'
    'IF' POST PARAM ST 'THEN' T[M+N+1,N+M+1]:=TWOT;
    'END';

?
R/END OF QUAD/END OF QUAP/
TE,E
```

TEXT-LISTING OF THE PARAMETRIC QP MAIN PROGRAMME.

```
'BEGIN' 'INTEGER' M,N,NAV,NEQ,REENTRY,I,J,PRINT,PR,PC,
NNEGD; 'REAL' FANCYHIGH,INITIAL VALUE,LAMBDA;
'BOOLEAN' TRULY FEASIBLE, TRULY BOUNDED;

'PROCEDURE' QUAP(T,M,N,NEQ,NAV,NNEGD,ROWLST,COLLST,PR,PC,
REENTRY,LAMBDA);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,NNEGD,PR,PC,REENTRY;
'REAL' LAMBDA;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'ALGOL';

'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;  'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,ER,RH,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,ER,RH;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';

'PROCEDURE' REPQ(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'COMMENT'
SIMPLEX ALGORITHM FOR QUADRATIC PROGRAMMING,
WITH POSTOPTIMAL PARAMETRIC VARIATION.

FOR DETAILS OF THE ALGORITHM,
SEE THE TEXT OF THE PROCEDURE QUAP.

PRESENTATION OF DATA:
FIRST THE NUMBER OF RESTRICTIONS I.E. M.
IF PARAMETRIC VARIATION OF THE OBJECTIVE FUNCTION
IS INTENDED, A PARAMETRIC ROW SHOULD BE INCLUDED AS A DUMMY-
RESTRICTION, BOTH IN THE NUMBER OF RESTRICTIONS AND IN THE
COEFFICIENTS MATRIX, AS THE LAST RESTRICTION.
THE NEXT NUMBER TO BE OFFERED IS THE NUMBER OF VARIABLES, N.
IF PARAMETRIC VARIATION OF THE RIGHTHAND-SIDE IS INTENDED,
A DUMMY-ACTIVITY SHOULD BE INCLUDED IN THE NUMBER OF VA-
RIABLES
AND -AS LAST COLUMN-, IN THE COEFFICIENTS MATRIX.
THESE NUMBERS SHOULD BE FOLLOWED BY
THE NUMBER OF EQUATIONS,  NEQ,
AND NAV, THE NUMBER OF VARIABLES TO WHICH
THE TACIT (NON-NEGATIVITY) RESTRICTION DOES   N O T  APPLY.
THE SERIES OF INFROMATION-PARAMETERS IS THEN CLOSED BY
THE INTEGER NUMBERS PR AND PC.
FOR PR=PC=0,
THE PROGRAM OPERATES AS A 'NORMAL' QUADRATIC PROGRAMMING
ALGORITHM.
FOR PR=1 WITH PC=0, PARAMETRIC VARIATION OF THE OBJECTIVE
FUNCTION IS ASKED,
AND FOR PR=0 WITH PC=1,
PARAMETRIC VARIATION OF THE RIGHT-HAND SIDE
IS ASKED.

BEFORE THE MAIN BODY OF NUMERICAL INFORMATION,
ONE SHOUD SUPPLY A SINGLE REAL NUMBER,
THE CONSTANT TO BE ADDED AS INITIAL VALUE TO
THE OBJECTIVE FUNCTION.
```

```
THEREAFTER PUNCH THE TABLEAU ITSELF.
THE REQUIRED PRESENTATION OF THE TABLEAU-MATRIX
IS THE SAME AS FOR 'ORDINARY' QUADRATIC PROGRAMMING,
EXCEPT FOR THE INCLUSION OF AN ADDITIONAL VECTOR, TO
CONTAIN THE PARAMETRIC ACTIVITY/RESTRICTION.
;


FANCYHIGH := 100;
M:=READ; N:=READ; NEQ:=READ; NAV:=READ;
NNEGD:= READ; PR:=READ; PC:=READ;
INITIAL VALUE := READ;

REENTRY:=0;
'BEGIN' 'ARRAY' TA[1:M+N+3,1:M+N+2];
  'INTEGER' 'ARRAY' ROWLST,COLLST[1:M+N];

  MATI(TA,M+N+2,N+1,0,0);
  PRINT:=TA[M+N+1,N+1];

  REORDER TO APPROPRIATE BLOCKS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' I:=2,1 'DO'
  TA[M+N+I+1,J]:=TA[M+N+I,J];
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+1 'DO' 'BEGIN'
    TA[I,M+N+1]:=TA[I,N+1]; TA[I,N+1]:=0; 'END';
  TA[M+N+1,N+1]:=0;

  RE INTERPRET:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'FOR' I:=1 'STEP' 1 'UNTIL' N+M+1 'DO'
    TA[I,N+M+1]:=TA[I,N+M+1]-TA[I,J]*TA[N+M+3,J];
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    TA[N+M+1,N+M+1]:=
    TA[N+M+1,N+M+1]+TA[I,J]*TA[J,I]*TA[N+M+3,J];
    TA[N+M+1,N+M+1]:=
    TA[N+M+1,N+M+1]-TA[N+M+1,J]*TA[N+M+3,J];
    'IF' TA[N+M+2,J]=0 'THEN' TA[N+M+2,J]:=FANCYHIGH;
    TA[N+M+2,J] := TA[N+M+2,J]-TA[N+M+3,J];
    'IF' TA[N+M+2,J] < 0 'THEN' 'BEGIN'
      NEWLINE(1);
      WRITETEXT('('YOU%HAVE%SUPPLIED%A%LOWER%LIMIT%IN%
      EXCESS%OF%THE%CORRESPONDING%UPPER%LIMIT')');
      NEWLINE(1);
      WRITETEXT('('THE%QP%PROBLEM%IS%THEREFORE%EMPTY.')');
      'END';
    'END';

  NOW SOLVE:
  FIRST CALL:
  QUAP(TA,M,N,NEQ,NAV,NNEGD,ROWLST,COLLST,
  PR,PC,REENTRY,LAMBDA);

  CORRECT 2T FOR INITIAL VALUE:
  TA[M+N+1,M+N+1] := TA[M+N+1,M+N+1] + 2*INITIAL VALUE;

  'IF' NNEGD > N 'THEN' NNEGD := N;
```

```
POINT OF OUTPUT:
CHECK ON STANDARD FORM:
'FOR' I:=1 'STEP' 1 'UNTIL' N+M 'DO'
'IF' 'NOT' ROWLST[I]=-COLLST[I] 'THEN' 'BEGIN'
  TRULY FEASIBLE := 'FALSE';
  'GOTO' SIGNAL UPPER LIMITS; 'END';

'IF' REENTRY=0 'THEN' 'BEGIN'
  CORRECT 2T ROW:
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' ROWLST[I]>0
  'THEN' TA[M+N+1,I]:=-TA[I,M+N+1]
  'ELSE' TA[M+N+1,I]:=TA[I,M+N+1]; 'END';

'IF' 'NOT' (PR=0 'AND' PC=0) 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('PARAMETER %VALUE')');
  WRITE(30,FORMAT('('S-NDDDDD.DD')'),LAMBDA); 'END';

CHECK FOR ARTIFICIAL FEASIBLITY:
TRULY FEASIBLE := 'TRUE';
'FOR' I:=1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' ROWLST[I]
> 3000 'THEN' 'BEGIN'
  TRULY FEASIBLE := 'FALSE';
  NEWLINE(1);
  WRITETEXT('('RESTRICTION')');
  WRITE(30,FORMAT('('SNDDDDS')'),ROWLST[I]-3000);
  WRITETEXT('('ONLY%ARTIFICIALLY%SATISFIED')'); 'END';

DITTO FOR SPECIFIED VARIABLES:
'FOR' I:=1 'STEP' 1 'UNTIL' N+M 'DO'
'IF' ROWLST[I]>2000 'AND' ROWLST[I]<3000 'THEN' 'BEGIN'
  TRULY FEASIBLE := 'FALSE';
  NEWLINE(1);
  WRITETEXT('('VARIABLE')');
  WRITE(30,FORMAT('('SNDDDDS')'),ROWLST[I]-2000);
  WRITETEXT('('ONLY%ARTIFICIALLY%POSITIVE')'); 'END';

SIGNAL UPPER LIMITS:
TRULY BOUNDED := 'TRUE';
'FOR' J:=1 'STEP' 1 'UNTIL' M+N 'DO' 'IF' COLLST[J]
> 2000 'AND' COLLST[J] < 3000 'THEN' 'BEGIN'
  'IF' TA[M+N+2,J] = FANCYHIGH
  'THEN' TRULY BOUNDED := 'FALSE';
  'IF' COLLST[J] = 2000+N 'THEN' TRULY BOUNDED := 'FALSE';
  'END';

REPS(TA,M,N,NEQ,NAV,ROWLST,COLLST);
'IF' PRINT > 0 'OR'
M+N < 14 'THEN' TABO(TA,M+N,M+N,M+N,0,0,1,1,ROWLST,COLLST)
'ELSE' TABO(TA,M+N,0,0,M+N,0,1,ROWLST,COLLST);

NEWLINE(1);
WRITETEXT('('SOLUTION%VALUE%IF%IN%STANDARD%FORM')');
WRITE(30,FORMAT('('S-NDDDD.DDD')'),TA[M+N+1,M+N+1]/2);
NEWLINE(1);
```

```
ENDLOOP FOR NORMAL QUADRATIC PROGRAMMING:
'IF' PR=0 'AND' PC=0 'THEN' 'GOTO' END OF PARQP;

ENDLOOP FOR PARAMETRIC VARIATION:
'IF' REENTRY=-1 'THEN' 'GOTO' END OF PARQP;
'IF' 'NOT' TRULY FEASIBLE 'OR' 'NOT' TRULY BOUNDED
'THEN' 'GOTO' END OF PARQP;

CHECK POSITIVE VALUE QUADRATIC FUNCTION:
'IF' PC=1 'AND'
'NOT' TA[M+N+1,M+N+1] > 0.0000000001 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('QUADRATIC%FUNCION%ZERO%OR%NEGATIVE%')');
  NEWLINE(1);
  'GOTO' END OF PARQP; 'END';


PARAMETRIC REENTRY CALL:
REENTRY:=2;
QUAP(TA,M,N,NEQ,NAV,NNEGD,ROWLST,COLLST,
PR,PC,REENTRY,LAMBDA);
'GOTO' POINT OF OUTPUT;

END OF PARQP:
'END'; 'END'
```

CHAPTER XVIII

GENERAL QUADRATIC PROGRAMMING

18.1 Statement and discussion of the general problem

In this chapter we discuss a method of solving the following problem

Maximise

$$\tau = \underline{w}' \underline{x} + \tfrac{1}{2} \underline{x}' D_o \underline{x} \qquad (18.1.1)$$

Subject to

$$\underline{c}_k' \underline{x} - \tfrac{1}{2} \underline{x}' D_k \underline{x} \leq b_k \qquad (18.1.2)$$

$$(k = 1, 2, \ldots m)$$

In practical numerical application it is extremely wasteful of both tableau-space and computational effort to treat linear restrictions as quadratic restrictions with a zero quadratic component. Thus, in practice (18.1.2) is split in

$$\underline{c}'_k \underline{x} \leq b_k \qquad (18.1.2a)$$

$$(k = 1, 2, \ldots, m_1)$$

and

$$\underline{c}'_k \underline{x} - \tfrac{1}{2} \underline{x}' D_k \underline{x} \leq b_k \qquad (18.1.2b)$$

$$(k = m_1+1, \ldots m)$$

The objective function is similar to the one used in Chapter 16, but now the side-conditions are also quadratic.

Example

Maximise

$$\tau = x_1 - (x_2-1)^2$$

Subject to

$$(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$$

$$(0 \leq x_1 \leq 10, \; 0 \leq x_2 \leq 10)$$

We first of all make a graphical mapping of the stated problem, with deletion of the upper limits. The one restriction is a circle, with a radius of 2, and the point $x_1 = 2$, $x_2 = 3$ as centre. The objective function is indicated by a series of parabolae (in fact, three) for $\tau = 0$ and $\tau = 1$, and $\tau = 3.1$. The highest one of these, insofar as it still touches the feasible space area (the circle) is to be chosen. In this example, that is also visually the highest but more generally, we should choose the highest value of τ. To conform to (18.1.1) and (18.1.2) it is necessary to reformulate the objective function as well as the restriction, i.e. work out the expressions within brackets.

Maximise

$$\tau =$$

$$x_1 + 2x_2 - x_2^2 - 1$$

Subject to

$$-4x_1 - 6x_2 + x_1^2 + x_2^2 + 4 + 9 \leq 4$$

i.e.

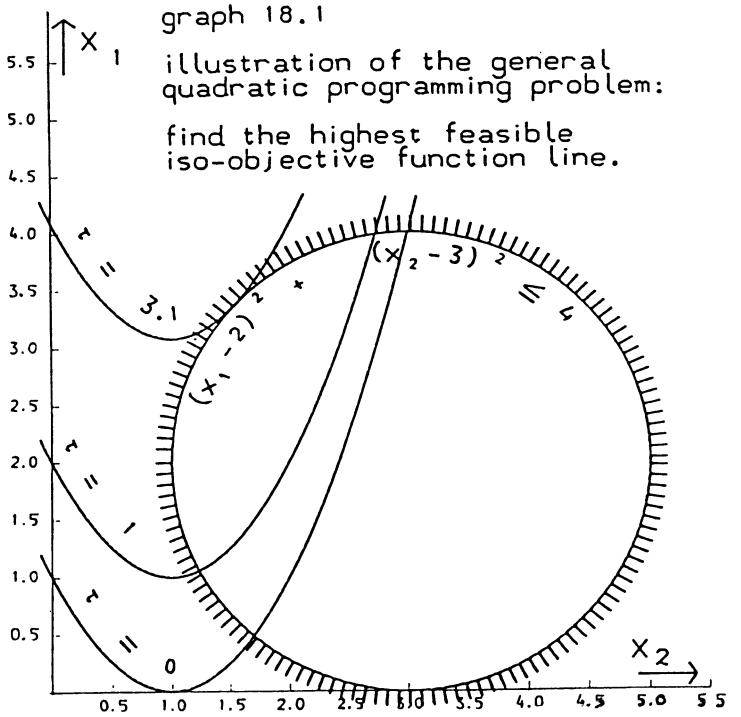
$$-4x_1 - 6x_2 + x_1^2 + x_2^2 \leq -9$$

Thus our initial set-up information is

$$\underline{w}' = \begin{bmatrix} 1 & 2 \end{bmatrix}, \qquad D_o = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix}$$

$$\underline{c}_1' = \begin{bmatrix} -4 & -6 \end{bmatrix} \qquad D_1 = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}$$

$$b_1 = -9$$



While this example is neatly convex, it is important to note that the efficacy of the van de Panne and Whinston/Cottle algorithm (on which this chapter heavily depends) does not require a convex preference function as such.

By and large, it is enough that each ordinary quadratic programming problem which will be generated possesses a proper optimum, and this requires convexity within the subspace defined by the binding linear restrictions, not convexity of the objective function as such. The algorithm to be developed in this chapter will therefore also be effective in many cases where some or all of the matrices D and D_k are indefinite, rather than negative (semi) definite.

Furthermore, it will be seen below, that with only a slight adaptation of the subsidiary quadratic programming algorithm we can in practice solve problems in which some of the restrictions are peripherally convex, or indeed anti-convex.

18.2 Pseudo-Lagrangean and Linear Approximation

The general quadratic programming algorithm revolves around the maximisation of a function which we might indicate as the pseudo-Lagrangian.

When the algorithm has converged the pseudo-Lagrangean is the Lagrangean function, less the terms which refer to the linear restrictions.

We do not so far know the values of any dual variables, and cannot directly verify whether the first-order optimality conditions are satisfied.

We therefore start with some provisional guesses at what the dual variables might be.

The pseudo-Lagrangean is the objective function, plus a non-negative combination of the quadratic restricting functions. As in the case of a proper Lagrangean, positive multipliers are restricted to restrictions which have been identified as possibly binding. In practice this means either violated or binding.

In the initial (trivial) solution we do not as yet identify any restriction as binding or violated and the pseudo-Lagrangean is the objective function.

The algorithm consists of, inter alia, a number of optimising phases. In each optimising phase we solve an "ordinary" quadratic programming problem, which we will indicate as a subsidiary problem. In this subsidiary "ordinary" quadratic programming problem, the objective function is the pseudo-Lagrangean.

The side conditions are 3 categories of linear restrictions, viz:

a) Any linear restrictions arising from (18.1.2a)

b) Linear approximations of quadratic side-conditions, and

c) Sometimes a "safety restriction" the nature of which will be discussed later on.

The presence of these additional linear restrictions (classes b and c) distinguishes the algorithm which is offered here from the group of algorithms surveyed by Fiacco and

McCormick [10]. Their algorithms are called sequential unconstrained minization techniques. By contrast the present algorithm might be called a sequentially constrained maximization method. We should, however, also mention a different method due to M.J.D. Powell [31], which is rather more similar to the one offered here. Although the programmed algorithm technically caters for "free" variables, this feature of the algorithm has not been tested, and indeed could give rise to boundedness problems.

Our discussion of the algorithm will assume that restrictions-class a) includes non-negativities and upper limits on the specified variables.

The "safety restriction " c) may not always be present (for a start it is not present in the initial problem) and the linear approximations may be totally inadequate initially.

To prevent unboundedness in the case of a semi-definite (or linear) objective function, we need to ensure that the feasible space area is bounded even without the quadratic restrictions.

We now discuss the linear approximations. Any linear approximation refers to vectors in the vicinity of some previously solved vector $\underline{x} = \underline{x}^*$. The initial linear approximation has to be supplied by the user, one would normally initiate at $\underline{x}^* = 0$. The actual value of $\underline{x}$ is then the sum of $\underline{x}^*$ and an increment, indicated as $\Delta\underline{x}$.

$$x = \underline{x}^* + \Delta\underline{x} \tag{18.2.1}$$

Substitution of the right-hand side of (18.2.1) for $\underline{x}$ into (18.1.2) yields the following expression

$$\underline{c}_k'(\underline{x}^* + \Delta\underline{x}) - \tfrac{1}{2}(\underline{x}^{*\prime} + \Delta\underline{x}')D_k(\underline{x}^* + \Delta\underline{x}) \leq b_k \tag{18.2.2}$$

Partial working out of (18.2.2) gives us:

$$\underline{c}_k'(\underline{x}^* + \Delta\underline{x}) - \underline{x}^{*\prime}D_k\Delta\underline{x} - \tfrac{1}{2}\underline{x}^{*\prime}D_k\underline{x}^* - \tfrac{1}{2}\Delta\underline{x}'D_k\Delta\underline{x} \leq b_k \tag{18.2.3}$$

We add a further term $-\frac{1}{2}\underline{x}^*{}'D_k\underline{x}^*$ to both sides of (18.2.3) causing the term $-\frac{1}{2}\underline{x}^*{}'D_k\underline{x}^*$ on the left-hand side to become a full term $-\underline{x}^*{}'D_k\underline{x}^*{}'$ and bring the term $\frac{1}{2}\Delta\underline{x}'D_k\Delta\underline{x}$ to the other side.

We again write $\underline{x}$ for $\underline{x}^* + \Delta\underline{x}$, and obtain after re-ordering:

$$(\underline{c}_k' - \underline{x}^*{}'D_k)\underline{x} \leq b_k - \tfrac{1}{2}\underline{x}^*{}'D_k\underline{x}^* + \tfrac{1}{2}\Delta\underline{x}'D_k\Delta\underline{x} \qquad (18.2.4)$$

The approximation consists in neglecting the term $\frac{1}{2}\Delta\underline{x}'D_k\Delta\underline{x}$, and the approximation is

$$(\underline{c}_k' - \underline{x}^*{}'D_k)\underline{x} \leq b_k - \tfrac{1}{2}\underline{x}^*{}'D_k\underline{x}^* \qquad (18.2.5)$$

Thus, in our example, our first approximation is

$$-4x_1 - 6x_2 \leq -9$$

and in general, for $\underline{x}^* = 0$, the initial approximation is

$$\underline{c}_k'\,\underline{x} \leq b_k \qquad (18.2.6)$$

For a properly convex quadratic restricting function D_k is negative semi-definite, hence the term $\frac{1}{2}\,\underline{x}'D_k\Delta\underline{x}$ is non-positive. But if the function is non-convex (or directionally convex), $\Delta\underline{x}'D_k\Delta\underline{x}$ may be positive. Thus for a convex restricting function the linear approximation is more liberal than the true quadratic restriction. The discrepancy between the true restriction and its approximation may go as far as finding the approximation not to be a meaningful restriction on the objective function at all, as may be illustrated in graph 18.2a below.

If the restriction is not convex, or only peripherally convex in the $\underline{x} \geq 0$ domain, the linear approximation may actually cut into the feasible space area. This may lead to complications. The problem may appear to be empty, despite the existence of a proper optimal and feasible solution in the true quadratic problem.

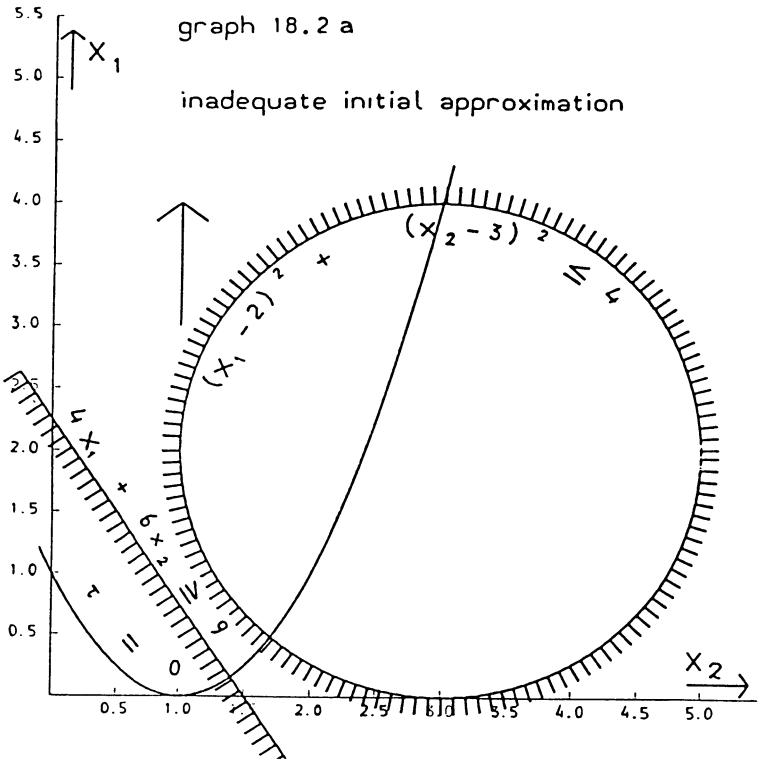
Example

Maximise

$$\tau = -(x_1 - 2)^2 - (x_2 - 3)^2 = 4x_1 + 6x_2 - x_1^2 - x_2^2 - 13$$

Subject to

$$-x_1 \cdot x_2 \leq -4$$



This problem has a proper solution, the unconstrained maximum of the objective function, $x_1 = 2$, $x_2 = 3$. However, the linear approximation of the hyperbola, taken at the point $x_1 = x_2 = 0$, is much more restrictive than the true non-linear restriction, it is $0\,x_1 + 0\,x_2 \leq -4$.

18.3 Verification and primal adjustment of subsidiary optima

When a subsidiary quadratic programming problem of maximising a pseudo-Lagrangean has been dealt with in the first instance, the linear approximations of non-convex restrictions may need adjustment. It should be borne in mind that when using the convex mode of operation the ordinary quadratic programming algorithm will come up with a "solution" irrespective of whether the problem is intrinsically solvable or not. For example, the first subsidiary optimisation problem for the quasi-convex example at the end of the previous section is

Maximise

$$\tau = 4x_1 + 6x_2 - x_1^2 - x_2^2 \quad (-13)$$

subject to

$$0 \leq -4$$

and the "solution" is $x_1 = 2$, $x_2 = 3$, with the dual variable p_1 at its fancy high upper limit, the slack-variable of the one restriction having been replaced by an artificial variable. The fancyhigh dual variable is not, however, particularly informative. In this somewhat odd and trivial example, the adjustment substantially amounts to the neutralisation of a nonsense-restriction. The "adjustment" will change the "restriction"

$0 \leq -4$ into $0 \leq 0$.

We will move a linear approximation restriction outward, whenever we find one binding, (or artificially satisfied), while the slack of the true quadratic restriction turns out to be positive. The operation need not be as apparently simple and trivial as in the example given above.

Consider the following example,

Maximise

$$\tau = -(x_1 - 1)^2 - (x_2 - 2)^2$$

Subject to

$$-x_1 - x_2 - 2x_1 \cdot x_2 \leq -4$$

i.e. $2(x_1 + \frac{1}{2})(x_2 + \frac{1}{2}) \geq 4\frac{1}{2}$

The subsidiary optimization problem, i.e.

Maximise

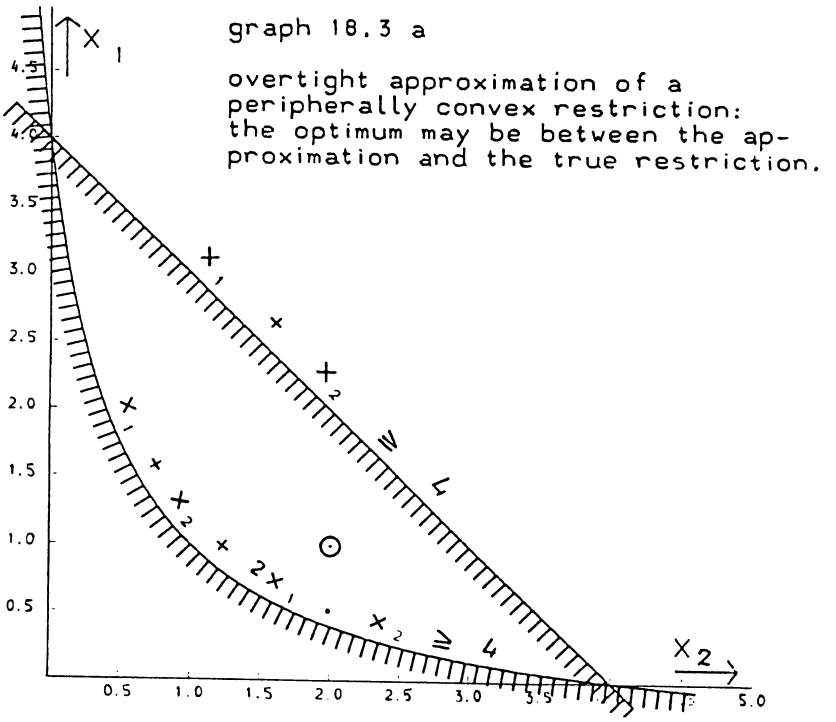
$$\tau = -(x_1 - 1)^2 - (x_2 - 2)^2 = 2x_1 - x_1^2 + 4x_2 - x_2^2 - 5$$

Subject to

$$-x_1 - x_2 \leq -4$$

is now first of all solved.

The set-up tableau and the subsidiary optimum tableau are given below in the tableaux 18.3a and 18.3b.



TABLEAUX 18.3 A AND 18.3 B

A SUBSIDIARY PROBLEM WITH AN OVERTIGHT OPPROXIMATION.

SET-UP TABLEAU

NA.	X1	X2	P1	VAL.
D1	-2	-	1	-2
D2	-	-2	1	-4
S1	-1	-1	-	-4
2T	-2	-4	4	-10

SUBSIDIARY OPTIMUM

NA.	D1	D2	S1	VAL.
X1	-0.25	0.25	-0.5	1.5
X2	0.25	-0.25	-0.5	2.5
P1	0.5	0.5	-1	1
2T	-1.5	-2.5	1	-1

We are now in a position to give precise operational significance to the notion of pushing the linear approximation restriction outwards. It means parametrically increasing the constant of the s_1-restriction, until the quadratic restriction becomes binding rather than amply fulfilled.

The s_1-column tells us that for every unit that the restriction is shifted outwards (back to the origin in this case), x_1 and x_2 will each be reduced by $\frac{1}{2}$.

In other words, if λ is the length of a parametric step in the indicated direction, we can put:

$$\Delta x_1 = -\tfrac{1}{2}\lambda, \quad \Delta x_2 = -\tfrac{1}{2}\lambda.$$

Confirm the notation of section 17.2 (parametric variation), we indicate a variation vector as $\underline{v}$, but in this case $\underline{v}$ is an updated vector, i.e.

$$\underline{x} = \underline{x}^* + \underline{v}\lambda \tag{18.3.1}$$

and

$$\Delta\underline{x} = \underline{v}\,\lambda \tag{18.3.2}$$

$$x_1 = 1\tfrac{1}{2} - \tfrac{1}{2}\lambda$$

$$x_2 = 2\tfrac{1}{2} - \tfrac{1}{2}\lambda$$

in the particular example.

We first implement the approximation-formula (18.2.5) at the current solution vector, and then proceed as follows:

Substitution of the right-hand side of (18.3.1), i.e. $\underline{x}^* + \lambda\underline{v}$ for $\underline{x}$, and of $\lambda\underline{v}$, the right-hand side of (18.3.2) for $\Delta\underline{x}$, into a binding form of (18.2.4) gives the following result:

$$(\underline{c}_k' - \underline{x}^{*\prime}D_k)(\underline{x}^* + \underline{v}\lambda) = b_k - \tfrac{1}{2}\underline{x}^{*\prime}D_k\underline{x}^* + \tfrac{1}{2}\underline{v}'D_k\underline{v}\lambda^2 \tag{18.3.3}$$

which may be re-ordered as

$$\tfrac{1}{2}\underline{v}'D_k\underline{v}\lambda^2 - (\underline{c}_k' - \underline{x}^{*\prime}D_k)\underline{v}\lambda + b_k$$

$$-\underline{c}_k'\underline{x}^* + \tfrac{1}{2}\underline{x}^{*\prime}D_k\underline{x}^* = 0 \tag{18.3.4}$$

A numerical evaluation of (18.3.4) is by far the most easily handled, if the vector $[\underline{c}_k' - \underline{x}^{*\prime}D_k]$, and the quadratic forms

$\underline{v}'D_k\underline{v}$ and $\underline{x}^{*\prime}D_k\underline{x}^*$ are calculated first, hence the implementation of (18.2.5) prior to (18.3.4).

To represent the term $2x_1x_2$ as $\frac{1}{2}\underline{x}'D\underline{x}$, we need $D = \begin{bmatrix} - & 2 \\ 2 & - \end{bmatrix}$,

and we calculate:

$$(\underline{c}_k' - \underline{x}^{*\prime}D_k) = \begin{bmatrix} -1 & -1 \end{bmatrix} - \begin{bmatrix} 1\frac{1}{2} & 2\frac{1}{2} \end{bmatrix} \begin{bmatrix} - & 2 \\ 2 & - \end{bmatrix} = \begin{bmatrix} -6 & -4 \end{bmatrix}$$

The quadratic form $\underline{v}'D_k'\underline{v}$ is evaluated as

$$\begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} - & 2 \\ 2 & - \end{bmatrix} \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix} = 1$$

and similarly

$$\underline{x}^{*\prime}D_k\underline{x}^* = \begin{bmatrix} 1\frac{1}{2} & 2\frac{1}{2} \end{bmatrix} \begin{bmatrix} - & 2 \\ 2 & - \end{bmatrix} \begin{bmatrix} 1\frac{1}{2} \\ 2\frac{1}{2} \end{bmatrix} = 15.$$

Accordingly, (18.3.4) is evaluated as

$$\tfrac{1}{2}\lambda^2 - \begin{bmatrix} -6 & -4 \end{bmatrix} \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix} \lambda - 4 - \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} 1\frac{1}{2} \\ 2\frac{1}{2} \end{bmatrix} + 7\tfrac{1}{2} = 0$$

or

$$\tfrac{1}{2}\lambda^2 - 5\lambda - 4 + 4 + 7\tfrac{1}{2} = 0$$

or

$$\tfrac{1}{2}\lambda^2 - 5\lambda + 7\tfrac{1}{2} = 0$$

We now solve λ as

$$\lambda = 5 \pm \sqrt{25 - 15} = 5 \pm \sqrt{10} = 5 \pm 3.162$$

$$\lambda_1 = 8.162, \quad \lambda_2 = 1.838$$

Both roots correspond to an exact solution of the binding form of the non-linear restriction. For the larger of the two roots, we find:

$$x_1 = 1.50 - 4.08 = -2.58$$

and

$$x_2 = 2.50 - 4.08 = -1.58$$

This point neatly satisfies the specified restriction - but not in the $\underline{x} \geq 0$ domain! When two positive roots are solved from (18.3.4) the smaller of the two is applicable.

If a restriction is non-convex rather than peripherally convex in the $\underline{x} \geq 0$ domain, a second positive root may also refer to a point in the $\underline{x} \geq 0$ domain. We then obviously take the one with the lowest absolute value.

The other case which may quite well arise in connection with a peripherally convex restriction is that we find a positive and a negative root. For example, if the restriction

$$\frac{1}{3} x_1 + x_2 \leq 2$$

is added to the previous restriction, the example becomes

Maximise

$$\tau = -(x_1 - 1)^2 - (x_2 - 2)^2 = -x_1^2 - x_2^2 + 2x_1 + 4x_2 - 5$$

Subject to

$$\frac{1}{3} x_1 + x_2 \leq 2$$

$$- x_1 - x_2 - 2x_1x_2 \leq -4$$

The first subsidiary problem is in that case the following:

Maximise

$$\tau = -(x_1 - 1)^2 - (x_2 - 2)^2$$

Subject to

$$\frac{1}{3} x_1 + x_2 \leq 2$$

$$- x_1 - x_2 \leq -4$$

and its optimal solution is

$$x_1 = 3, \; x_2 = 1.$$

Parametric variation of the constant of the second restriction now means to slide laterally along the binding restriction

$\frac{1}{3} x_1 + x_2 = 2$. This line actually cuts the quadratic restriction at two points, on account of the curvature of the quadratic restriction. Our underlying problem is that possibly the solution might be in the area which is satisfied by the true quadratic restriction, but not by the linear approximation. (This has been illustrated in graph 18.3a.) - Therefore, we want to relax the linear approximation, not to make it a more stringent restriction. This means algebraically that when one of the roots is negative and one positive, the positive root is the relevant one; see also graph 18.3b.

The roots of (18.3.4) may also be complex and their real parts may be positive or negative. Complex roots with a negative real part though possible, are not very useful, and one can in such cases, only hope that after a fresh start, new, more satisfactory approximations will avoid the problem. The case of a pair of complex roots with a positive real part is, however, useful to persue.

Example

Maximise $\tau = -(x_1 - 3)^2 - (x_1 - 2)^2$

Subject to $2x_1 - x_2 \leq 0$

$x_1 - x_2^2 + 2x_2 \leq 2$

It will be noted that the quadratic restriction in this example is non-covex, and indeed, anticonvex. This problem, nevertheless, has a unique optimal and feasible solution. The quadratic restriction is not even binding on the optimum, and in any case, even if it was e.g. if the linear restriction $2x_1 - x_2 \leq 0$ were left out, the strict convexity of the objective function would still ensure a unique optimum. The initial linear approximation of the quadratic restriction is $x_1 + 2x_2 \leq 2$ and the first subsidiary optimum is $x_1 = \frac{3}{8}$, $x_2 = \frac{3}{4}$. The optimal quadratic programming tableau corresponding to the maximization of $-(x_1-3)^2-(x_2-1)^2 + 10 = -x_1^2 + 6x_1 - x_2^2 + 2x_2$, subject to $2x_1 - x_2 \leq 0$ and $x_1 + 2x_2 \leq 2$ is given below, with the parametric column and the corresponding displaced solution. (Tableau 18.3c). Non-meaningful upper limits, as well as parametric variations in the objective function have been replaced by XX.

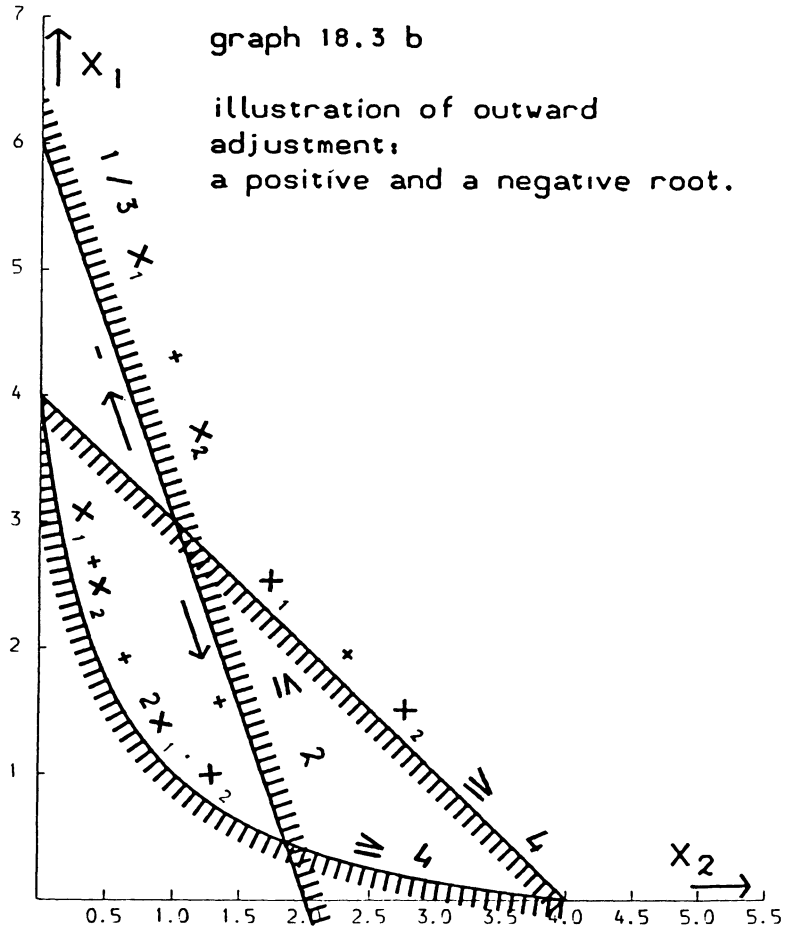

graph 18.3 b
illustration of outward adjustment:
a positive and a negative root.
X1
X2
1/3 X1 + X2
X1 + X2
2X1 + X2
X1 + X2 ≥ 2
X1 + X2 ≥ 4
2X1 + X2 ≥ 4
7
6
5
4
3
2
1
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

TABLEAU 18.3 C

A SUBSIDIARY OPTIMUM, WITH PARAMETRIC ADJUSTMENT,
BASED ON A PAIR OF COMPLEX ROOTS OF THE LIMIT EQUATION.

NAME	!!	D 1	D 2	S 1	S 2	!!	VALUE	L	VA(L=1.13)
X 1	!!	-	-	0.40	0.20	!!	0.40	0.20	0.63
X 2	!!	-	-	-0.20	0.40	!!	0.80	0.40	1.25
P 1	!!	-0.40	0.20	-0.40	-	!!	1.60	-	1.60
P 2	!!	-0.20	-0.40	-	-0.40	!!	2	-0.40	1.55
2T	!!	-0.40	-0.80	1.60	2	!!	9.60	XX	XX
UB	!!	XX	XX	XX	XX	!!	XX	1.13	XX

Relaxation of the binding s_2-restriction now means that for every unit that the s_2-restriction is pushed outwards, x_1 increases by $\frac{1}{5}$ and x_2 by $\frac{2}{5}$. This is due to moving along the binding linear restriction, $2x_1 - x_2 \leq 0$. Inspection of graph 18.3c indicates that this restriction never meets the quadratic restriction.

Application of (18.3.4) for:

$$D_k = \begin{bmatrix} - & - \\ - & 2 \end{bmatrix} \qquad \underline{v} = \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$

$$\underline{c}_k' = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

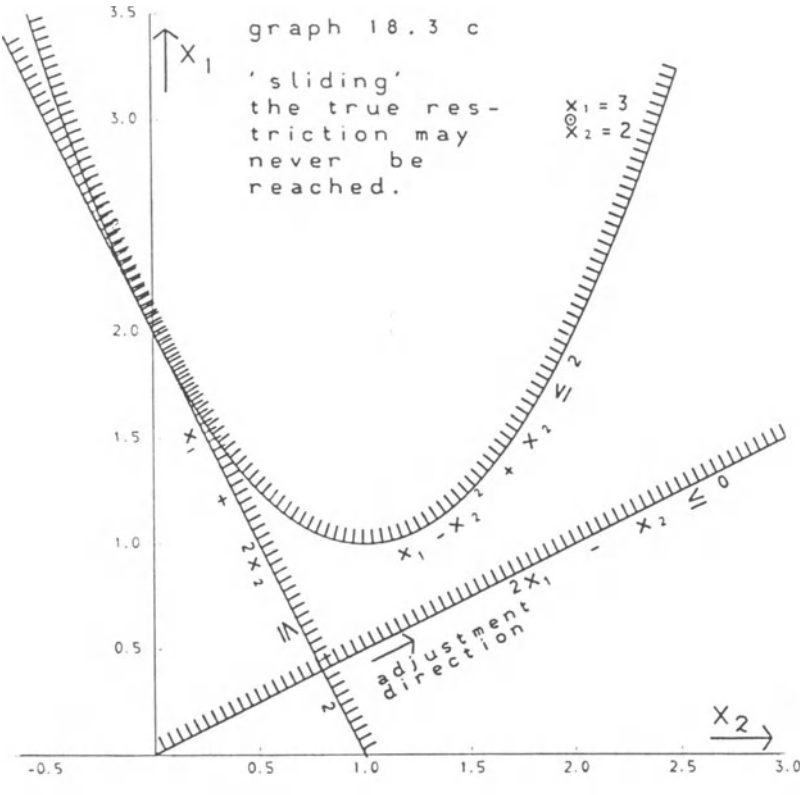
$$\underline{x}^* = \begin{bmatrix} \frac{2}{5} \\ \frac{4}{5} \end{bmatrix} \qquad b_k = 2$$

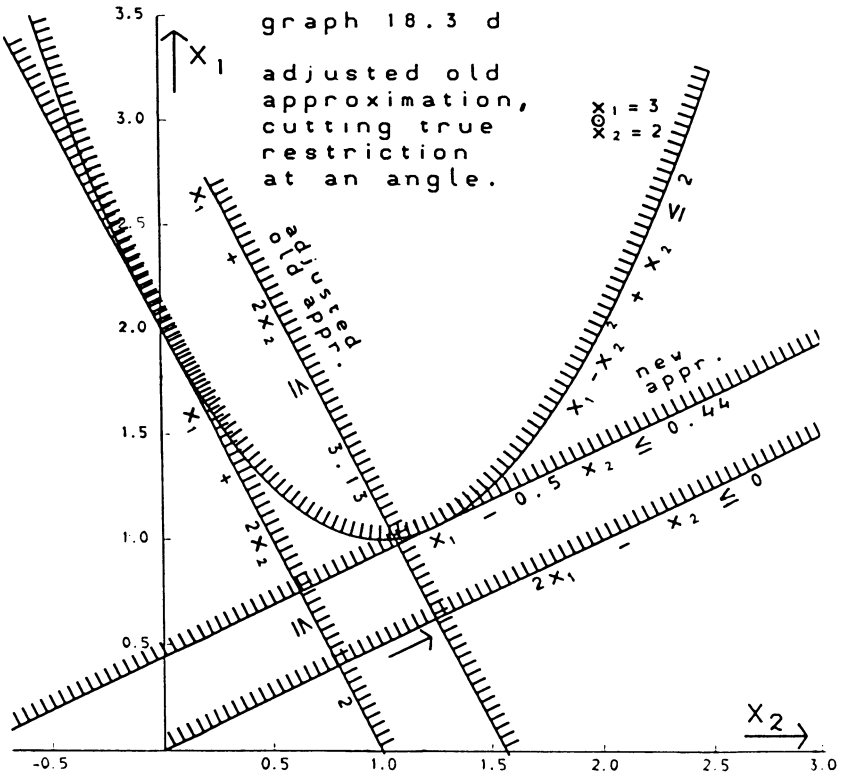
yields

$$\frac{4}{25}\lambda^2 - \frac{9}{25}\lambda + \frac{16}{25} = 0$$

This equation yields a pair of complex roots

$$\lambda = \frac{9}{8} \pm \frac{5}{8}\sqrt{-7}$$




graph 18.3 d
adjusted old approximation, cutting true restriction at an angle.
X_1
X_2
$x_1 = 3$
$x_2 = 2$
adjusted old appr.
new appr.
$x_1 + 2x_2 \leq 3.13$
$x_1 - x_2^2 + x_2 \leq 2$
$x_1 - 0.5\,x_2 \leq 0.44$
$2x_1 - x_2 \leq 0$
3.5
3.0
2.5
2.0
1.5
1.0
0.5
-0.5
0.5
1.0
1.5
2.0
2.5
3.0

The algebraic equivalent of this situation is finding a pair of complex roots to (18.3.4).
The problem is solvable nevertheless.

When we meet a pair of complex roots in this way we take the real part only or what amounts to the same, we put the first-order differential of the left-hand side of (18.3.4) at zero

$$\lambda = \frac{(\underline{c}_k - \underline{x}^{*\prime} D_k)\underline{v}}{\underline{v}' D_k \underline{v}} \qquad (18.3.5)$$

We then subsequently take a new linear approximation and repeat the operation. Thus for $\lambda = 1\frac{1}{8}$ we find $x_1 = 0.63$, $x_2 = 1.25$. The new graphical mapping below gives both the adjusted linear approximation and the new one, taken at $x_1 = 0.63$ and $x_2 = 1.25$.

The new linear approximation may or may not come to be used at that point. The reason for this qualification is that the calculated value of λ, whether obtained as a real root or as the real part of a pair of complex roots is a maximum value. It may or may not happen that the ordinary quadratic programming vertex changes before the full adjustment is reached. There may be another restriction in between, or the adjusted solution may not be optimal. The parametric adjustment algorithm of Chapter 17 is applicable, and only one parametric step is made. In the example at hand, the adjusted vertex $x_1 = 0.63$ and $x_2 = 1.25$ is the new subsidiary optimum. Accordingly, the next subsidiary problem is

Maximise

$$-(x_1 - 3)^2 - (x_2 - 2)^2$$

Subject to

$$2x_1 - x_2 \leq 0$$

$$x_1 - 0.50x_2 \leq 0.44$$

The optimum solution of this problem is

$$x_1 = 1.4 \quad x_2 = 2.8 \text{ with } p_1 = 1.6,\ p_2 = 0$$

The quadratic restriction, as well as its approximation are now amply fufillled and no dual variable has so far been introduced into the pseudo Lagrangean. Hence this particular subsidiary optimum is actually the true global optimal and feasible solution. The problem at hand has been solved.

One related issue which comes up under the heading of verifying the solution of a subsidiary problem is the recognition of emptiness.

The obvious way of recognising an empty problem is by way of finding a subsidiary problem empty. The following statements apply in that connection.

If a restricting function is properly convex the set of vectors which satisfy the corresponding true restriction is entirely included in the set of vectors which satisfy an approximation (the approximation is more liberal than the true restriction).

Therefore, in the convex case, the feasible space area of the true problem is entirely included in the feasible space area of any subsidiary problem and we may conclude to emptiness of the true problem, as soon as we find a subsidiary problem to be empty.

Note however, that this indication does not always arise in the first subsidiary problem.

Example

Maximise $\tau = x_1 + (x_2-1)^2$

Subject to $x_1 \geq 5$

$(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$

$(0 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 10)$

This problem is a modification of the one put at the opening of this chapter, the modification consisting of the introduction of the linear restriction $x_1 \geq 5$, making the problem empty.

Whilst the linear restriction contradicts the true quadratic restriction, it does not contradict the initial approximation of the quadratic restriction $(4x_1 + 6x_2 \geq 9)$, and the initial subsidiary problem attains an optimal and feasible solution at $x_1 = 10$, $x_2 = 1$.

For the convex case, convergence of the sequentially constrained maximization method arises on account of progressive reduction of the attainable value of the objective function, a topic to be discussed in more detail later in this chapter.

The algorithm has two permitted endings, finding the optimum, and finding the true problem empty, by way of finding a subsidiary problem empty. (The issue of unboundedness does not

arise on account of the presence of (artificial) upper limits on all specified primal variables). Finding the optimum of the true problem is impossible for an empty problem, it therefore follows that emptiness will be signalled in the way indicated, if not for the initial subsidiary problem then for a later subsidiary problem.

There are, however, complications for non-convex problems or indeed, as soon as not all functions are properly convex. It is in this connection necessary to differentiate between true linear restrictions and linear approximations. This was in fact, done by employing the convex mode of operation in solving subsidiary problems including and indeed in particular with non-convex problems.

The method of dual upper limits and artificial variables then permits to put the penalty-coefficient for not meeting a true linear restriction an order of magnitude higher than the penalty coefficient for not meeting an approximation restriction.

If we then find that a true linear restriction is met only artificially, we assume that the true linear restrictions contradict each other, and conclude to emptiness, irrespective of convexity (we strictly lack proof on this point, see section 16.10).

For quadratic restrictions the adjustments discussed earlier in this section then comes before we investigate emptiness. This obviously leaves the possibility that emptiness of a non-convex problem leads to non-convergence.

We adjust one restriction, formulate a new subsidiary problem, with the same pseudo-Lagrangean, and find that the modified problem again leads to a solution where (the same or another) true restriction is amply fulfilled by a solution vector associated with a binding or artificially met approximation. Adjustment is then again called for once more.

Whether there are artificial variables left in association with restrictions where the approximation and the true restriction are violated, is then never investigated.

However, emptiness will be recognized, even in a non-convex problem, if an artificially feasible "optimum" arises, for which all binding or artificially met approximations relate to binding or violated true restrictions.

18.4 Reapproximations

Let us now suppose that we have found the optimal and feasible solution of a subsidiary problem, and that we have verified the solution-structure (or if necessary re-defined the problem) to the extent that no approximation-restriction is binding on a solution for which the true quadratic restriction is amply fulfilled.

In such a situation there are generally restrictions for which the true quadratic restriction is violated, while the approximation is binding or amply fulfilled. One possible approach towards getting nearer to the true solution is to impose new approximations as additional restrictions, or as we will call such restrictions reapproximations. There are two versions of this technique. If the current approximation of a quadratic restriction is not binding on the optimum of the subsidiary problem (while the true quadratic restriction is violated, the value of the restricting function being negative non-zero), we speak of a loose approximation.

If the current approximation is binding, but we expect to be able to replace it by a new approximation, we speak of superimposing a new approximation. In general, reapproximation means that we do not (as yet) formulate a new subsidiary problem to be solved from the trivial basis onwards, but re-enter the ordinary QP algorithm in the re-entry mode, with a new violated approximation restriction. The loose approximation is the simplest of the two cases, and will be discussed first.

We again take the same example i.e.

Maximise $\tau = x_1 - (x_2 - 1)^2$, subject to

$$(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$$

$$(0 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 10)$$

The $p = 0$ subsidiary optimum, with the so-far unused "extra" restriction slot is given below as tableau 18.4a.

As we saw already earlier in this chapter the initial approximation $4x_1 + 9x_2 \geq 4$ is not binding on the optimal solution, the positive value of $s_1 = 37$ confirms this. In the interest of containing the number of restrictions to be administered, and of keeping the option of imposing more in future we simply discard a loose approximation, and put a new one in its place.

TABLEAU 18.4 A

ILLUSTRATION OF LOOSE APPROXIMATION.

NAME	!!	B1	D2	P1	P2	!!	VALUE	DIST
U1	!!	-	-	-4	-	!!	1	X
X2	!!	-	-0.50	-3	-	!!	1	9
S1	!!	4	-3	-18	-	!!	37	X
S2	!!	-	-	-	-	!!	0	X
2T	!!	1	-1	-37	-	!!	22	X
UB	!!	10	X	100	100	!!	X	X

Application of (18.2.5), for $\underline{x}^{*\prime} = [10, 1]$

yields a new approximation, which is $16x_1 - 4x_2 \leq 92$

(The reader is invited to verify this calculation referring to 18.2.5)

Formulae for calculating a currently updated part of the tableau from a known block-pivot, (the inverse of which may be extracted from the current tableau), and the non-updated form of the tableau, may be obtained by generalisation of the similar problem as it arises in the LP case.

From section 11.1 we derive the following formulae for calculating those elements of a "missing" row which refer to non-basic (primal and dual) slack variables.

$$\underline{t}'_{2,1} = - \left[\underline{a}'_{2,1}, 0\right] \; P^{-1} \qquad (18.4.1)$$

Here $\underline{t}'_{2,1}$ is a row of the second block-row, first block-column block of the current tableau. The composite vector $\left[\underline{a}'_{2,1}, 0\right]$ arises because in the QP case, the rows which relate to primal restrictions need to be extended with zeros, insofar as dual variables are concerned. (The straightforward application of the matrix-formulae from section 11.1 to the vectorial case would be

$$\underline{t}'_{2,1} = - \underline{a}'_{2,1} \, A_{1,1}^{-1}.$$

In applying (18.4.1) in the presence of a binding upper-limit restriction we must also consider the unwritten x_1-row, which is a unit-vector, with the unity element in the b_1 column.

The full pivot inverse therefore is

	d_1	d_2	b_1
x_1	-	-	1
x_2	-	-0.50	-
u_1	-1	-	-

Application of (18.4.1) therefore results in

$$-s_1 \begin{matrix} \\ \end{matrix} \begin{matrix} x_1 & x_2 & u_1 \end{matrix} \begin{bmatrix} 16 & -4 & 0 \end{bmatrix} \quad \begin{matrix} \\ x_1 \\ x_2 \\ u_1 \end{matrix} \begin{matrix} d_1 & d_2 & b_i \end{matrix} \begin{bmatrix} - & - & 1 \\ - & -0.50 & - \\ -1 & - & - \end{bmatrix} = s_1 \begin{matrix} d_1 & d_2 & b_1 \end{matrix} \begin{bmatrix} 0 & -2 & -16 \end{bmatrix}$$

However, since the d_1 column is not stored as a currently updated column, we do not need to calculate an additional element to it, which is anyhow systematically zero. Also multiplication by zero in relation with dual variables elements of the non-updated form of the row may be suppressed and the non-trivial part of this calculation is

$$-s_1 \begin{matrix} x_1 & x_2 \end{matrix} \begin{bmatrix} 16 & -4 \end{bmatrix} \quad \begin{matrix} x \\ x_2 \end{matrix} \begin{matrix} b_1 & d_2 \end{matrix} \begin{bmatrix} 1 & 0 \\ 0 & -0.50 \end{bmatrix} = s_1 \begin{matrix} b_1 & d_2 \end{matrix} \begin{bmatrix} -16 & -2 \end{bmatrix}$$

The calculation of an additional column is in a QP tableau facilitated by the symmetry rules; having done the row already we just copy with, where appropriate, a change in sign.

Only for one element do we need a further application of the matrix formulae from section 11.1, i.e. to calculate an element of a non-basic variable's column which does not figure as an eliminated slack-variable in the pivot inverse.

In view of the symmetry between $a_{2,1}$ as non updated row and the corresponding dual variable's column, the formulae for the

diagonal element is

$$t_{2,2} = \left[\underline{a}_{2,1},\ 0\right] \quad P^{-1} \begin{bmatrix} \underline{a}_{2,1} \\ 0 \end{bmatrix} \qquad (18.4.2)$$

In practice, we obtain this result as the inner product between the already updated row i.e. $-\left[\underline{a}'_{2,1} \mid 0\right]P^{-1}$, and the unupdated form of the dual variables' column.

In this example that calculation is

$$\begin{array}{c} \\ s_1 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & d_2 \end{array} \\ \begin{bmatrix} -16 & -2 \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{c} p_1 \end{array} \\ \begin{array}{c} b_1 \\ d_2 \end{array} \begin{bmatrix} - \\ 4 \end{bmatrix} \end{array} = \begin{array}{c} \begin{array}{c} p_1 \end{array} \\ s_1 \end{array} \begin{bmatrix} -8 \end{bmatrix}$$

Since the dual variables' column contains zero entries for all primal variables' rows, this calculation can be restricted to its non-trivial part only and becomes

$$\begin{array}{c} \\ s_1 \end{array} \begin{array}{c} \begin{array}{c} d_2 \end{array} \\ \begin{bmatrix} -2 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ d_2 \end{array} \begin{array}{c} \begin{array}{c} p_1 \end{array} \\ \begin{bmatrix} 4 \end{bmatrix} \end{array} = \begin{array}{c} \begin{array}{c} p_1 \end{array} \\ s_1 \end{array} \begin{bmatrix} -8 \end{bmatrix}$$

Similar calculations for the value column cell are possible, but not necessary. Reference to (18.2.4) makes clear, that, at $\underline{x} = \underline{x}^*$ itself the current slack of the true restriction and the approximation are the same. Since we needed to calculate the quadratic slack in the first place in order to establish its sign, it is now known (to be - 64).

The tableau now becomes as given in tableau 18.4b.

TABLEAU 18.4 B

REENTRY TABLEAU FOR IMPOSING THE REAPPROXIMATION OF A LOOSE APPROXIMATION RESTRICTION.

NAME	!!	B1	D2	P1	P2	!!	VALUE	DIST
U1	!!	-	-	16	-	!!	1	X
X2	!!	-	-0.50	-2	-	!!	1	9
S1	!!	-16	-2	-8	-	!!	-64	X
S2	!!	-	-	-	-	!!	0	X
2T	!!	1	-1	64	0	!!	22	X
UB	!!	10	X	100	100	!!	X	X

Normal re-entry of the ordinary QP algorithm now leads to a new optimal and feasible solution, for which the calculation tableau is given in tableau 18.4c

TABLEAU 18.4 C

A REOPTIMIZED SUBSIDIARY PROBLEM, AFTER IMPOSING THE REAPPROXIMATION OF A LOOSE APPROXIMATION RESTRICION.

NAME	!!	D1	D2	S1	P2	!!	VALUE	DIST
X1	!!	-0.03	-0.12	0.06	-	!!	6.03	3.97
X2	!!	-0.12	-0.50	-	-	!!	1.13	8.88
P1	!!	-0.06	-	-	-	!!	0.06	X
S2	!!	-	-	-	-	!!	0	X
2T	!!	-6.03	-1.13	0.06	0	!!	14.03	X
UB	!!	X	X	X	100	!!	X	X

The approximation of the quadratic restriction is now binding and we cannot simply discard it as not being useful.

What we can, however still do, is to impose a new approximation as well. We therefore now occupy the $(m+1)^{th}$ (second) restriction slot.

We apply (18.2.5) again, now for $\underline{x}^{*\prime} = [6.03, 1.13]$ and find $8.06\ x_1 - 3.75\ x_2 \leq 28.64$.

The same procedure of updating an additional restriction now results in tableau 18.4d

TABLEAU 18.4 D

ILLUSTRATION OF THE SEARCH FOR SUPERIMPOSITION.

NAME	!!	D1	D2	S1	P2	!!	VALUE	DIST
X1	!!	-0.03	-0.12	0.06	-0.22	!!	6.03	3.97
X2	!!	-0.12	-0.50	-	-0.87	!!	1.13	8.88
P1	!!	-0.06	-	-	0.50	!!	0.06	X
S2	!!	-0.22	-0.87	-0.50	-1.50	!!	-15.77	X
2T	!!	-6.03	-1.13	0.06	15.77	!!	14.03	X
UB	!!	X	X	X	100	!!	X	X

At this point we investigate whether or not there is a reasonable prospect of actually replacing the old approximation by the new one. In the general m-variable case, we might wish to repeat the operation for several restrictions and this is possible, only if after re-entry of the ordinary QP algorithm, we again have one binding approximation for one restriction, in which case the now amply fulfilled old approximation can be discarded.

To assess the likelihood of that to happen we investigate the implications of two tentative pivots, by comparing their associated ratios with the value column.

These two pivots are: the n + m + 1/n + m + 1 cell i.e. the elimination of (the s_2 variable) the slack variable of the new approximation by its own dual variable, and the m + 2/n + m + 1 cell, i.e. the elimination of the dual variable of the old approximation of the offending r^{th} restriction by the dual variable of the new approximation of the same quadratic restriction, slotted in place n + m + 1.

In the example, this leads to comparison of the ratios - 15.77/-1.50 = 10.5 in the s_2 row and 0.06/0.50 = 0.12 in the p_1 - row. The one in the p_1 row is the smaller one. Therefore we can at least be sure that re-entry of the ordinary QP algorithm will not result in simply exchanging the slack of the new approximation against the dual variable of the new approximation, by way of activating the diagonal pivot. This would *a fortiori* apply in the case of a zero diagonal cell, and the ratio is classified as infinite in that case. We do not investigate any other ratios and accept the new approximation as probably fully superimposable.

In anticipation of subsequently discarding the old approximation, we reorder the tableau, now classifying the new approximation as the "proper" one, and the old one as the "additional" one. This re-ordering results in tableau 18.4e.

TABLEAU 18.4 E

REENTRY TABLEAU FOR SUPERIMPOSING

NAME	!!	D1	D2	P1	S2	!!	VALUE	DIST
X1	!!	-0.03	-0.12	-0.22	0.06	!!	6.03	3.97
X2	!!	-0.12	-0.50	-0.87	-	!!	1.13	8.88
S1	!!	-0.22	-0.87	-1.50	-0.50	!!	-15.77	X
P2	!!	-0.06	-	0.50	-	!!	0.06	X
2T	!!	-6.03	-1.13	15.77	0.06	!!	14.03	X
UB	!!	X	X	100	X	!!	X	X

The resulting new optimal and feasible solution (not summarised in detail) is indeed characterized by an amply fulfilled s_2 restriction this being the old approximation $16x_1 - 4x_2 \leq 92$.

We then say that the new approximation ($8.06x_1 - 3.75x_1 \leq 28.64$), has been fully superimposed over the old one. If this were not the case, i.e. if both approximations were binding on the optimal solution we would speak of a blocked superimposition and the operation could not be repeated, for lack of a slot to put a new re-approximation in.

Note however, that the old approximation may sometimes again be unblocked, by imposing a new approximation of a different restriction, where the previous approximation became loose in the process of superimposing.

If we carry on superimposing new approximations we run the obvious risk of "slicing off" smaller and smaller areas of the feasible space area. The computational implementation offered here contains a search operation for the most violated restriction for which a probably fully superimposable approximation may be identified, and a limit of at most 2 q (twice the number of quadratic restrictions) superimpositions for any pseudo-Lagrangean, superimposing new approximations whenever the $(m+1)^{th}$ slot is free.

In the example at hand is limit is 2, and it was operative, i.e. a second probable fully superimposable approximation of the same restriction is identified and actually leads to superimposing the new approximation once more, but thereafter no more are made until a new pseudo-Lagrangrean has been formed.

One further refinement of the approximation-technique which is useful to discuss at this point, is the inwardly adjusted reapproximation, and more generally the adjusted reapproximation.

In section 18.3 we assumed that the whole tableau and solution-vector was adjusted, using the parametric variation method from chapter 17. Although a drastic change in the slope of an approximation-restriction is generally undesirable it is not necessary that reapproximations refer to solution-vectors which satisfy the (approximations of) other restrictions. Furthermore, it is at least as important that they come in some sense near the restriction to which they refer and this may well be achieved by taking approximations of different restrictions at different points. For reasons to be explained later on in this chapter this is especially important in the case of peripheral convexity where it is desirable to obtain tangential approximations. The technique of inward adjustment is applicable on both properly convex and on peripherally convex restrictions. We now illustrate it in relation to proper convexity.

Example (the one with which this chapter was opened)

Maximise $\tau = x_1 - (x_1 - 1)^2$

Subject to $(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$

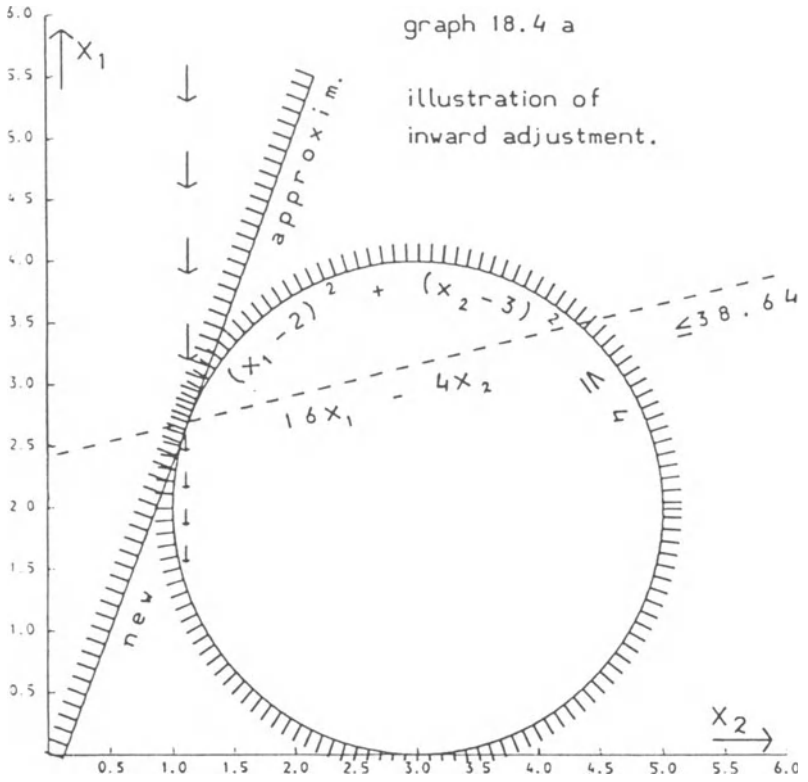
$(0 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 10)$

The first subsidiary problem has the optimal solution $x_1 = 10$, $x_2 = 1$, which is nowhere near the true solution. We put a new linear approximation

$$16x_1 - 4x_2 \leq 92$$

in place of the initially loose approximation.

This restriction is binding on the subsidiary optimum reported tableau 18.4e ($x_1 = 6.03$, $x_2 = 1.13$). We initially discussed superimposition with respect to a new approximation taken at that point. The further refinement of (inwardly) adjusted reapproximation consists of taking the new approximation to be superimposed, not at ($x_1 = 6.03$, $x_2 = 1.13$), but at ($x_1 = 2.70$, $x_2 = 1.13$), the solution which is obtained by adjusting the existing approximation-restriction,


graph 18.4 a
illustration of
inward adjustment.
X_1
X_2
approxim.
new
$(x_1-2)^2 + (x_2-3)^2 \leq 38.64$
$4x_2 - 16x_1 \leq 5$
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0

$16\ x_1 - 4\ x_2 \leq 92$, inwardly to $16\ x_1 - 4\ x_2 \leq 38.64$, where the true quadratic restriction is exactly fulfilled.

Note (see graph 18.4 a), that the adjusted old approximation e.g. $16\ x_1 - 4\ x_2 \leq 38.64$ generally cuts through the true quadratic restriction, the new approximation is tangential to it. As in the case of outward adjustment only the smaller of two positive roots of the quadratic adjustment equation (18.3.4) is applicable. The adjustment of the solution-vector from $x_1 = 6.03$, $x_2 = 1.12$ is marked in the graph by the downwards pointing arrows, the adjustment ends in fact at the end of the line of bigger arrows, the continuation merely indicates the significance of the other root.

Before we finalize this section, we may note that there are two versions of the (inwardly) adjusted reapproximation.

The new approximation may be brought in by superimposing it over the old one, or we may formulate new approximations for all restrictions and solve the same problem again from the trivial basis. The latter version will apply if a return to start has already been indicated for some other reason. In that case new approximations will be taken with inward adjustment for violated true restrictions for which approximations are binding, and with outward adjustment if any overtight approximations are encountered.

Hence the general name adjusted reapproximation.

18.5 The objective function limit

Let us denote as $\underline{x} = \underline{x}^{**}$, the optimal solution vector of a particular subsidiary problem, in which

$P(\underline{x}) = \tau(\underline{x}) + \underline{p}'\ \underline{f}$ figures as the objective function.

The corresponding vector of true slacks (the values of the restricting functions $f_i(\underline{x}^{**})$ is indicated as $\underline{f}^{**}$.

Let us further assume that the problem is property convex, in $\tau(\underline{x})$ as well as in each $f_i(\underline{x})$. We also assume that the particular solution vector $\underline{x}^1 = \underline{x}^{**}$ satisfies the condition of optimal form $p_i\ f_i^{**} \leq 0$, $(p_i \geq 0)$, all i (we cannot achieve the complementary slackness condition as such, $p_i = 0$ if $f_i > 0$, $f_i = 0$ if $p_i > 0$, until we have solved the problem). We then have, (for a convex problem when a subsidiary problem has been found to satisfy the condition of optimal form), the following

Theorem

In a convex problem all vectors $\underline{x}$ which satisfy the true restrictions $f_i(\underline{x}) \geq 0$, all i, also satisfy

$$\left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^{**})\right]' \underline{x} \leq \left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^{**})\right]' \underline{x}^{**} + \underline{p}' \underline{f}^{**} \qquad (18.5.1)$$

i.e. a linear approximation of

$$\tau(\underline{x}) \leq \tau(\underline{x}^{**}) + \underline{p}' \underline{f}^{**} \qquad (18.5.2)$$

taken at $\underline{x} = \underline{x}^{**}$.

Proof 1 (assuming optimal form)

Consider the mathematical programming problem (hereafter named the displaced and constrained problem).

Maximise $\tau(\underline{x})$,

Subject to

$$f_i(\underline{x}) \geq f_i(\underline{x}^{**})$$

as well as the restrictions of the subsidiary problem. (If the latter group of restrictions is absent, we will speak of the displaced problem.)

The Lagrangean expression, associated with the optimal solution of the displaced and constrained problem, differs from the Lagrangean expression which is associated with the optimal solution of the subsidiary problem only in its constant term.

We simply re-group each term $p_i f_i$, as figuring in the objective function of the subsidiary problem with a term $-p_i f_i^{**}$, to obtain the corresponding term of the Lagrangean associated with the optimal solution of the displaced and constrained problem.

It follows that $\underline{x} = \underline{x}^{**}$ also is an optimal solution of the displaced and constrained problem, the feasible space area of which includes that of the true problem entirely. A tangential approximation of the aggregate restriction of the displaced and constrained problem is:

$$\left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^{**})\right]' \underline{x} \leq \left[\frac{\partial \tau}{\partial \underline{x}}(\underline{x}^{**})\right]' \underline{x}^{**} \qquad (18.5.3)$$

This restriction (18.5.3) is also obtainable as a non-negative combination of the restrictions of the old subsidiary problem (with the corresponding linear shadowprices as

multipliers), and tangential approximations of $f_i(\underline{x}) \geq f_i(\underline{x}^{**})$, (with the p_i as multipliers). The similar combination of the restrictions of the old subsidiary problem, and new approximations of the true restricting functions $f_i(\underline{x}) \geq 0$, results in (18.5.1).
q.e.d.

Note that if $\tau(\underline{x})$ is a convex function an upper limit on the objective function i.e. (18.5.2) is an anti-convex restriction and its tangential approximation (18.5.1) is more not less restrictive than an upper limit on $\tau(x)$ as given in (18.5.2) itself. All $\underline{x}$ satisfying (18.5.1) also satisfy (18.5.2), but some $\underline{x}$ may satisfy (18.5.2), without satisfying (18.5.1).

It follows that if (18.5.1) is added as an additional restriction to the next subsidiary problem, such a device ensures, for $\underline{p}' \underline{f}^{**} < 0$, an actual reduction in the value of the objective function between one subsidiary problem and its successor.

We will therefore, in the rest of this chapter, refer to (18.5.1) as "the objective function limit" as it implies (in the convex case), (18.5.2).

Note

The above proof becomes invalid if the assumption of optimal form is dropped. This is because the feasible space area of the displaced and constrained problem includes that of the true problem, only if optimal form is assumed.

If some of the displaced restrictions (associated with $p_i > 0$) require $f_i(x) \geq f_i(\underline{x}^{**}) > 0$ we have to consider the possibility that the optimal solution of the displaced and constrained problem is associated with a solution value which is below the value of the true optimum.

In fact (18.5.1) is still valid provided the restrictions are convex. In this connection we supply

Proof 2 (not assuming optimal form)

Since $P(\underline{x})$ is the constrained maximum of the pseudo-Lagrangean,

$$P(\underline{x}) = \tau(\underline{x}) + \underline{p}' \underline{f} \leq \tau(\underline{x}^{**}) + \underline{p}' \underline{f}^{**} \qquad (18.5.4)$$

holds for all feasible solution vectors of the subsidiary problem and therefore by implication, for all feasible solution vectors of the true problem.

Since $\underline{p}' \underline{f}$ is required to be non-negative (for a feasible solution) the non-linear form of the objective function limit (18.5.2) follows. The equivalence between the two interpretations of (18.5.3) - a tangential approximation of an upper limit on the objective function, or of the aggregate restriction of the displaced and constrained problem - holds irrespective of optimal form. It follows that (18.5.1), understood as a non-negative combination of the restrictions of the subsidiary problem and of new approximations of the quadratic restrictions, taken at $\underline{x} = \underline{x}^{**}$ also applies irrespective of optimal form.
q.e.d.

The validity of the objective function limit in the presence of peripheral convexity will be discussed in the next section.

18.6 Consistency and optimality

The optimal solution of a subsidiary problem is said to be consistent with the true problem if the solution vector $\underline{x} = \underline{x}^{**}$ results in $f_i(x) \geq 0$, for all i, in other words if it satisfies the primal restrictions of the true problem.

The conditions of optimal form ($p_i f_i \leq 0$, $p_i \geq 0$), and consistency ($f_i \geq 0$) together amount to the familiar complementary slackness conditions ($p_i = 0$ if $f_i > 0$, $f_i = 0$ if $p_i = 0$, $p_i f_i \geq 0$).

Despite the fact that there may be binding linear restrictions in the optimum of the subsidiary problem, a consistent solution also is an optimal solution of the true problem, at least as far as the primal solution vector $\underline{x}$ is concerned.

The following cases arise:

Case a:

No binding restrictions in the optimal solution of the subsidiary problem

If the optimum solution of the subsidiary problem is the unconstrained maximum of the pseudo-Lagrangean, the pseudo-Lagrangean is revealed to be the optimal Lagrangean of the true problem.

Case b:

Some approximation-restrictions are binding, the objective function limit is not binding

Since loose approximations are always replaced by new

approximations, consistency implies that for all restrictions for which dual variables p_i have been stated, there are corresponding exactly met linear approximations.

Comparison of (18.2.4) and (18.2.5) now shows that consistency implies

$$\Delta \underline{x}' D_i \Delta \underline{x} = (\underline{x} - \underline{x}_i^{**})' D_i (\underline{x} - \underline{x}_i^{**}) = 0 \qquad (18.6.1)$$

For all i for which $p_i >$ applies.

(This quadratic form is the error term in the approximation formula, if the approximation fits exactly the error is by implication zero).

Reference to (18.2.4) also shows that, if the error term vanishes conform (18.6.1), the true restricting function and its approximation have the same vector of differentials.

There are only two possibilities for $\Delta \underline{x}' D_i \Delta \underline{x}$ to vanish: Either $\Delta \underline{x} = 0$ (the approximation was taken at $\underline{x} = \underline{x}^{**}$ itself), or $\Delta \underline{x}$ is a characteristic vector associated with a zero latent root of D_i. (see also sections 14.4 and 14.7).

In either case, we find, denoting the restricting function associated with an approximation restriction as $f_i^* (\underline{x}_i)$

$$\frac{\partial f_i^*}{\partial \underline{x}} = \frac{\partial f_i}{\partial \underline{x}}$$

The dual variables associated with the true optimal solution are then related to the ones assigned to the pseudo-Lagrangean by the following relationship.

$$p_i^{**} = p_i + p_i^* \qquad (18.6.2)$$

where p_i^{**} are the true dual variables, p_i are the (estimates of the) dual variables that were assigned to the pseudo Lagrangean, and p_i^* are the shadowprices of the linear approximation restrictions as obtained from the dual solution of the optimum of the subsidiary problem.

The left-hand side of (18.6.2) gives the dual variables as they figure in the Lagrangean of the true problem, the righthand side gives two separate terms as figuring in the Lagrangean associated with the subsidiary optimum, where a term $p_i f_i$ is grouped with the objective function and $p_i f_i^*$ with the rest of the Lagrangean.

Case c:

The objective function limit is binding

We only have a meaningful use of the objective function limit if certain properties of convexity are satisfied.

We postulate that each approximation restriction which is subsumed in the objective function limit (which is a non-negative combination of approximation-restriction), permits an area which includes all of the area permitted by the corresponding true restriction. (For properly convex restricting functions, this assumption is always satisfied, but we shall need to discuss the situation which arises in the case of peripheral convexity in more detail).

With that provisio, a binding objective function limit implies that all previous approximation-restrictions that are subsumed in the objective function limit, are exactly met. (The stated convexity property means that none of their slacks can be negative, the corresponding true restricting functions being non-negative on account of the assumed consistency, if their sum is zero each separate term must also be zero).

The argument put forward under case b, may now be extended to case c. Note, however, that (18.6.2) will in case c, with a binding objective function limit, understate the true value of the dual variable p_1^{**}, as shadowprices of binding old approximation restrictions may be subsumed in the shadowprice of the binding objective function limit.

We now come to discuss the issue of the applicability of the objective function limit in the presence of peripheral convexity, and its implications for the optimality theorem stated in this section. By keeping a track-record of the applications of (18.2.2) we may establish whether or not a particular restriction is a tangential approximation. If (18.2.2) were in fact binding at $\underline{x} = \underline{x}^*$ itself - as would, in particular, be the case if the application of (18.2.2) was made either in the context of an inwardly adjusted reapproximation, or after outward adjustment, following the discovery of overtightness, we would be entitled to treat such a tangential approximation of a peripherally convex restriction on the same basis as an approximation of a convex restriction.

Under those conditions, the linear form of the objective function limit is also a tangential approximation of a peripherally convex restriction, i.e. of (18.5.2) interpreted as the aggregate restriction of the displaced and constrained problem displaced once more by adding the term $\underline{p}' \underline{f}^{**}$.

If a fully non-convex restriction - or a non-tangential approximation of a peripherally convex restriction has become active, the formulation of a new objective function limit is inhibited, and the issue of it being binding does not arise. We may therefore conclude that finding a solution-vector which satisifes both the requirement of optimal form, and is consistent with the true problem, is equivalent to having found the primal solution of the true problem.

18.7 The upward adjustment of dual variables

There are two noticeably distinct ways of changing the estimates of the dual variables assigned to the pseudo-Lagrangean, which we will indicate as normal transition under optimality, and correction of optimal form.

If the optimal solution of the last subsidiary problem satisfies the condition of optimal form, we assume that the collection of restrictions for which there are dual variables and/or binding approximation restrictions is the collection of restrictions which is binding on the optimum solution of the true problem. In that case we adjust the dual variables upwards, according to the rules to be laid down in this section. If optimal form has been lost we proceed in a quite different way, adjusting more downwards than upwards, a subject to be discussed in more detail in the next section.

The formulation of a pseudo-Lagrangean in which in particular convexly-shaped restrictions are represented by positive non-zero dual variables, is of particular importance if the solution-structure of the optimal solution of the true problem is characterised by an excess of the number of non-zero elements of $\underline{x}$, over the number of binding restrictions, while the objective-function is linear or anti-convex.

Example

Maximise $\tau = -x_1 - x_2$

Subject to $-4x_1 - 4x_2 + x_1^2 + x_2^2 \leq -7$

$(0 \leq x_1 \leq 100, \quad 0 \leq x_2 \leq 100)$

Any subsidiary problem which consists of maximising the objective function, subject to a linear approximation of the one restriction will put either x_1 or x_2 at zero. Yet the true solution ($x_1 = x_2 = 1-\sqrt{2}$) assigns non-zero values to both variables. The non-linear component of the restriction is essential.

We now address ourselves to the question of formulating suitable rules for adjusting dual variables upwards under normal transition. It is intuitively obvious that the shadow-prices of binding approximation restrictions are relevant information here. We may also observe that once a consistent solution without a binding objective function limit has been found the problem is essentially solved. We only need to add the shadowprices of the approximation restrictions to the dual variables, i.e. we apply (18.6.2).

It would therefore be desirable that any rules to be laid down would converge to (18.6.2) in the vicinity of the optimal solution of the true problem - or nearly so, for reasons which will become clearer in the next section.

This desideratum is achieved by proceeding on the assumption that the current solution vector $\underline{x} = \underline{x}^{**}$ is the optimal solution of the displaced problem, i.e. the true problem modified by displacing the quadratic restrictions to $f_i(\underline{x}) \geq f^{**}_i = f_i(\underline{x}^{**})$, and that the approximation-restrictions are proportional to new approximations of $f_i(\underline{x}) \geq f_i(\underline{x}^{**})$, taken at $\underline{x} = \underline{x}^{**}$.

This assumption leads to a generalization of (18.6.2), namely

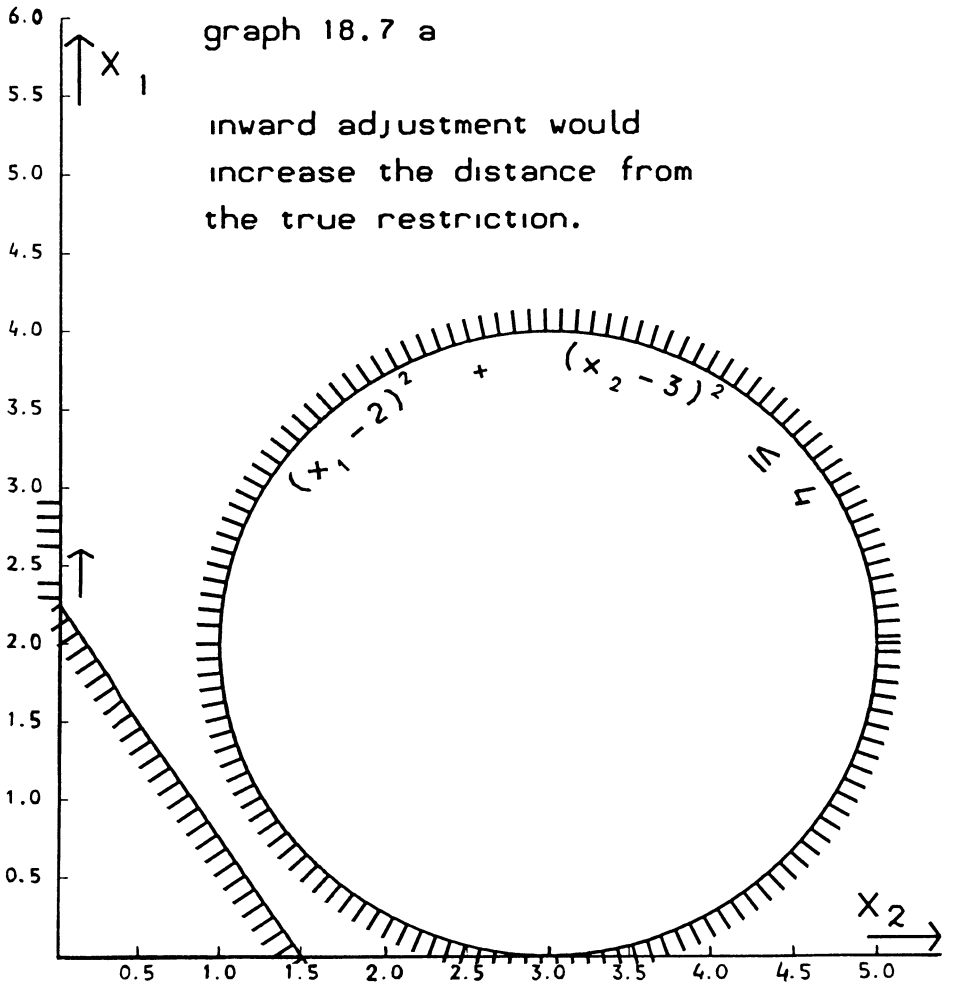
$$p^{**}_i = p_i + p^*_i / \frac{df_i}{db^*_i} \qquad (18.7.1)$$

where df_i/db^*_i may be obtained by evaluating the change in the true restricting function f_i, per unit of increase in b^*_i, as given by the f^*_i column of the optimal tableau of the subsidiary problem.

If that maintained assumption is actually true, the current solution vector is (at least in the convex case), the unconstrained maximum of the new pseudo-Lagrangean, but a further approximation to the true optimal and feasible solution is obtained by introducing new approximation-restrictions, which do not admit $\underline{x} = \underline{x}^{**}$ as a feasible solution-vector.

The relation (18.7.1) may be indicated as the correction equation, and the reciprocal of the differential df_i/db^*_i is the correction-factor - if (18.7.1) is applied in that form.

In fact, certain modifications of the correction equation may be desirable, in view of the fact that the maintained assumptions that the current solution is the optimum of the displaced problem and that the approximation-restrictions are proportional to (= geometrically coincide with) tangential approximations of the displaced quadratic restrictions, may not be true.


graph 18.7 a
inward adjustment would increase the distance from the true restriction.
X 1
X 2
$(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

The following notes summarise modifications to (18.7.1) and the reasons, therefore, in order of obviousness and importance.

Firstly, it is even possible for the differential df_i/db^*_i to be negative

Example

Minimise $x_1 + 3x_2$

Subject to $(x_1 - 2)^2 + (x_2 - 3)^2 \leq 4$

$(0 \leq x_1 \leq 100, \quad 0 \leq x_2 \leq 100)$

This problem has been graphically summarised in graph 18.7a. The initial approximation is $4x_1 + 6x_2 \geq 9$ and the initial subsidiary optimum is found at $x_1 = 2.25$, $x_2 = 0$.

If the initial approximation is adjusted inwards in the sense of making it more stringent, by adjusting its constant we increase the distance from the true restriction.

This problem of the adjustment going the wrong way is solved in the case of primal adjustment by simply not adjusting (see sections 18.3 and 18.4). That will not do for the dual; as the example also makes clear, we need some non-zero value for the dual variable of, in particular, a convex restriction. When the calculated correction factor is found to be negative, we substitute an arbitrary positive number for it; the number 0.5 was chosen.

Secondly, unduly big calculated correction factors may arise. In the extreme, we could theoretically meet the borderline-case between a positive and negative correction-factor, which would in this case be infinite, with df_i/db^*_i being zero.

We obviously need to impose an upper limit. This limit was eventually set at 0.999, just below the figure which still permits (18.7.1), as modified, to converge to (18.6.2) for a consistent solution.

18.8 Loss and correction of optimal form

The rules for adjusting dual variables upwards, which were discussed in section 18.7, are, as far as practicable, designed to ensure that no estimate of a dual variable is formulated which is actually in excess of the true value. These are, however, not proof-supported features of the algorithm.

If, despite these attempts, an estimate of a dual variable is assigned to a quadratic restriction and entered into the pseudo Lagrangean, and such an estimate exceeds the true value of the dual variable in question as it relates to the optimal solution of the true problem, we say that the dual variable in question has been overstated.

We may distinguish two distinct varieties of overstating. If we have assigned a dual variable to a restriction which is not binding on the true optimal solution of the true problem, and there ought therefore not to be a non-zero dual variable for the restriction in question at all, we speak of overstatement by mis-identification.

Over-statement of a dual variable of a restriction which is binding on the true optimum, is then indicated as simple overstatement.

The calculation of correction factors as discussed in the previous section, may, or may not, lead to avoiding simple overstatement. There are, however problems for which over-statement by mis-identification cannot well be avoided.

Example

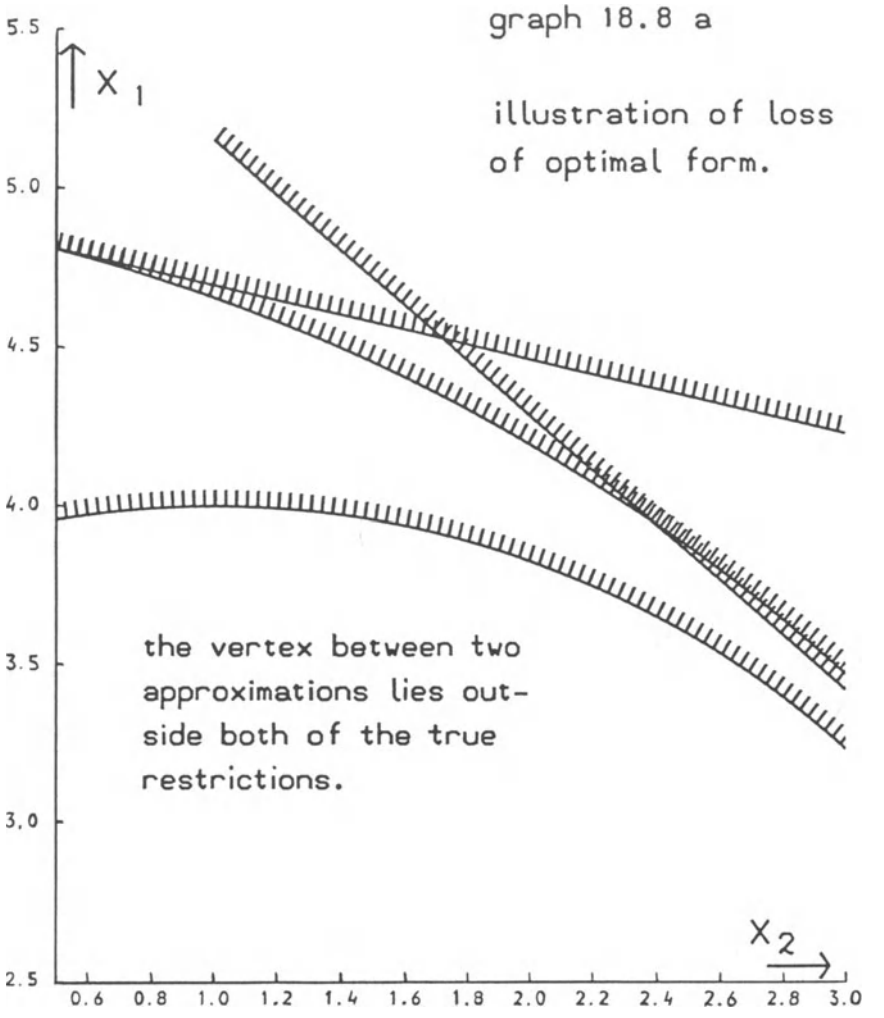
Maximise $\tau = 2\,x_1 + x_2$

Subject to $x_1^2 - 2\,x_1 + x_2^2 - 2\,x_2 \leq 7$

$x_1^2 + 2\,x_1 + x_2^2 + 2\,x_2 \leq 34$

$(0 \leq x_1 \leq 100, \quad 0 \leq x_2 \leq 100)$

This problem is graphically surveyed in graph 18.8a. The first restriction is equivalent to $(x_1 - 1)^2 + (x_2 - 1)^2 \leq 9$ and is represented in the graph by a circle with radius 3, and the centre at the point $x_1 = 1$, $x_2 = 1$. The second restriction is equivalent to $(x_2 + 1)^2 + (x_2^2 + 1)^2 \leq 36$ and is represented by a circle with radius 6, and the centre at the point -1, -1.


graph 18.8 a
illustration of loss of optimal form.
X 1
X 2
5.5
5.0
4.5
4.0
3.5
3.0
2.5
0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
the vertex between two approximations lies outside both of the true restrictions.

The graph makes it quite clear that the second restriction is in fact redundant. However, a finite maximum of the linear objective function requires 2 binding restrictions, and the direction of the objective function does not make it likely that the non-negativities of x_1 and x_2 itself will become binding. Indeed, the initial subsidiary problem, modified by a series of reapproximations, finds its optimum solution at $x_1 = 4.53$, $x_2 = 1.72$, the objective function at this point being contained by two approximation restrictions, one for each quadratic restriction.

Both of the true restrictions are violated by the subsidiary optimum $x_1 = 4.53$, $x_2 = 1.72$ and dual variables are stated for both restrictions ($p_1 = 0.069$, $p_2 = 0.097$).

We now have a strictly convex pseudo Lagrangean and one binding linear restriction is sufficient to contain the pseudo-Lagrangean. The second subsidiary problem finds its optimal solution, at $x_1 = 3.82$, $x_2 = 2.43$, and this time only the approximation of the first restriction is binding, and we find $f_1 = -0.999$, $f_2 = +1$.

We now establish loss of opitmal form ($p_2 = 0.097$, $f_2 = 1$).

There are a number of modalities of loss of optimal form , we may meaningfully distinguish the following:

Subdominant loss of optimal form: $\underline{p}' \underline{f} < 0$, some $p_i f_i > 0$

dominant " " " " : $\underline{p}' \underline{f} \geq 0$, some $p_i f_i > 0$

total loss of optimal form : all $p_i f_i \geq 0$, some $p_i f_i > 0$.

Subdominant loss of optimal form still permits the formulation of a new objective function limit. This <u>may</u> also apply for dominant loss of optimal form.

For $\underline{p}' \underline{f} \geq 0$, a new objective function limit does not require a reduction of the value of the objective function limit below its value in the last subsidiary problem but that level itself may be well below the value required by the previous objective function limit.

Nevertheless, when optimal form has been lost, and in particular when dominant or total loss of optimal form have arisen, we can no longer look for a continuous reduction in the objective function value as our guarantee of not repeating the same solutions again. Instead, we require a continuous reduction in the sum of all dual variables.

To analyse the problem of optimal form, we denote

as $\underline{x} = \underline{x}^{***}$, the primal solution vector of the true problem
as $\underline{x} = \underline{x}^{**}$, the primal solution vector of the subsidiary problem.
as $\underline{p}^{***}$ the vector of true dual variables, and as $\underline{p}$ the vector of estimates assigned to the pseudo-Lagrangean.

Absence of overstatement is then given as $\underline{p} \leq \underline{p}^{***}$. In fact, there are complications with non-unique subsidiary optima, precisely for $\underline{p} = \underline{p}^{***}$, and we will for the moment assume $\underline{p} \leq \underline{p}^{***}$, $p \neq \underline{p}^{***}$.

Theorem

In a convex problem $p_i < p_i^{***}$, if $p_i^{***} > 0$ ensures that optimal form is maintained.

Proof

We first state a much weaker theorem, which refers to total loss of optimal form.

Lemma

Total loss of optimal form implies that one of the two following conditions is present

Either (a) $\underline{p} \geq \underline{p}^{***}$, $\underline{p} \neq \underline{p}^{***}$

at least one dual variable has been overstated.

or

(b) $\underline{p} = \underline{p}^{***}$

Proof (of the lemma)

If (a) applies, no further proof is needed.
Now assume not (a) and in particular absence of overstatement by mis-identification.

We then find

$$P(\underline{x}^{***}) = \tau(\underline{x}^{***}) = L(\underline{x}^{***}) \tag{18.8.1}$$

We also find, since $\underline{x} = \underline{x}^{***}$ maximises $L(\underline{x})$,

$$L(\underline{x}^{**}) = \tau(\underline{x}^{**}) + \underline{p}^{***\prime}\, \underline{f}(\underline{x}^{**}) \leq \tau(\underline{x}^{***}) = L(\underline{x}^{***}) \tag{18.8.2}$$

We also find, since $\underline{x}^{***}$ is a feasible solution of the subsidiary problem, and $\underline{x}^{**}$ the constrained maximum of $P(\underline{x})$,

$$P(\underline{x}^{***}) = \tau(\underline{x}^{***}) \leq \tau(\underline{x}^{**}) + \underline{p}' \, \underline{f}(\underline{x}^{**}) \qquad (18.8.3)$$

Consolation of (18.8.1), (18.8.2) and (18.8.3) yields:

$$\tau(\underline{x}^{**}) + \underline{p}^{***\prime} \, \underline{f}(\underline{x}^{**}) \leq \tau(\underline{x}^{***}) \leq \tau(\underline{x}^{**}) + \underline{p}' \, \underline{f}(\underline{x}^{**}) \qquad (18.8.4)$$

From which

$$\underline{p}^{***\prime} \, \underline{f}(\underline{x}^{**}) \leq \underline{p}' \, \underline{f}(x^{**}) \qquad (18.8.5)$$

Since total loss of optimal form is defined as $f_i(\underline{x}^{**}) > 0$ if $p_i \leq 0$, we find that (18.8.4) can only be satisfied in the presence of total loss of optimal form, if

$$\underline{p}^{***\prime} \leq \underline{p}' \qquad (18.8.6)$$

applies, which is equivalent to (a) or (b)
q.e.d. (for the lemma)

The more general theorem now follows by induction.

If loss of optimal form is not total there are some i for which $p_i \leq 0$, $f_i \leq 0$ and therefore binding approximation restrictions. Within the subspace of those binding approximation restrictions any loss of optimal form is total loss of optimal form.
q.e.d.

We now come to discuss the problem of the non-unique subsidiary optimum. In a problem which is convex, but not strictly convex, the true Lagrangean often has no strict maximum. The most extreme case arises with a linear problem.

Example

Maximise $\quad \tau = x_1 + x_2$

Subject to $\quad x_1 \leq 1; \; x_2 \leq 1$

$\qquad (0 \leq x_1 \leq 100, \quad 0 \leq x_2 \leq 100)$

If the upper limit on the correction factor for calculating dual variables were set at 1 instead of at 0.999, our second subsidiary problem would be

Maximise $P(x_1\ x_2) = x_1 + x_2 + 1\ (1 - x_1) + 1(1 - x_2) - 2$

Subject to $x_1 \leq 1,\quad x_2 \leq 1 \quad (x_1, x_2 \geq 0)$

$x_1 + x_2 \leq 2$ (objective function limit)

The pseudo-Lagrangean reduces to a constant, and the origin is co-equally optimal with the optimum of the true problem.

It is clearly undesirable to treat linear restrictions as quadratic in this way, but one assumes that this problem can also arise with semi-definiteness. Hence the downward bias in calculating estimates of dual variables. We normally do not calculate exactly the true vector $\underline{p}^{***}$, but continue to obey the condition $p_i < p_i^{***}$, if $p_i^{***} > 0$, and find the problem solved when both consistency of the primal solution and zero residual shadowprices of approximation restrictions are satisfied with a set tolerance, we accept not exactly meeting these latter conditions. Note also that in a non-convex problem, there may be several local optima.

Regardless of the fact that we lack a proof, the plausible weaker theorem that the pseudo-Lagrangean attains a local maximum which respects optimal form does not inform us to which local maximum is the "right" one. It certainly is not always the global maximum of the pseudo-Lagrangean.

Example

Maximise $\tau = 5x_1 + 4x_2 - x_2^2$

Subject to $10x_1 - 10x_2 \leq x_2^2 - x_1^2 - 16$

$(0 \leq x_1 \leq 4;\ 0 \leq x_2 \leq 100)$

Here the one restriction is peripherally convex in the $x_1 \leq 4$ domain as may be seen by writing it as $(x_2 - x_1)(10 - x_1 - x_2) \geq 16$; the graph (not given) is a rectangular hyperbola with asymptotic axes $x_1 = x_2$ and $x_1 + x_2 = 10$, the upper part where $x_2 - x_1 < 0$ and $10 - x_1 - x_2 < 0$ may coincide is excluded by the upper limit on x_1.

The second subsidiary problem, with $p_1 = 0.50$ is:

Maximise $P(\underline{x}) = 9x_2 + 0.50\ x_1^2 - 1.50\ x_2^2$

Subject to $8.06\ x_1 - x_2 \leq 3.31$

$(0 \leq x_1 \leq 4;\quad 0 \leq x_2 \leq 100)$

The global optimum of this problem is $x_1 = 0$, $x_2 = 3$ with $f_1 = 55 > 0$.

There is however, a second constrained local maximum, $x_1 = 2.9$, $x_2 = 4.5$, with $f_1 = -11.84$ which is rather more helpful. (The true optimum is $x_1 = 0.56$, $x_2 = 3.08$, with $p_1 = 0.56$). We may thus classify loss of optimal form as being due to three possible causes, viz: Overstatement of dual variables by mis-identification, simple overstatement of dual variables, and (in a non-convex problem only) getting at the "wrong" local solution of a subsidiary problem. Overstatement by mis-identification cannot be avoided, we can only correct it afterwards.

Simple overstatement is, we hope, normally prevented by the calculation of correction-factors, for normal transition under optimality, but it can also be corrected, should it arise.

There are a number of features of the SCM algorithm - and of the adaptation of the ordinary QP algorithm employed by it, which are aimed at developing local optima in which approximation-restrictions are binding, in preference to other restrictions. However, we lack a proof on that point, and at the same time correction of optimal form is, in effect adapted to ameliorate the first two causes of loss of optimal form only: we start to reduce dual variables, assuming that some are overstated.

In short the SCM algorithm although clearly capable of solving some non-convex problems, is proof-supported, only in the convex case. We now discuss the correction of optimal form in more detail.

We proceed as follows:

We first formulate a "correction restriction". This correction restriction is a linear approximation of a combination of restrictions $f_i \leq 0$

i.e.

$$\Sigma\, m_i\, f_i \leq 0 \qquad (18.8.7)$$

where

$$m_i = p_i, \quad \text{if} \quad f_i(\underline{x}^{**}) > 0$$

$$m_i = 0, \quad \text{if} \quad f_i(\underline{x}^{**}) \leq 0$$

The calculation of this correction-restriction is performed at the start of a correcting dual adjustment round (before each separate correcting dual adjustment round). Each correcting dual adjustment round then consists of:

(1) A semi-proportional downward adjustment of all dual variables p_i, for which $f_i(\underline{x}^{**}) > 0$ applies.

The first downward adjustment is associated with dual parametric variation of the subsidiary optimum, the linear approximation of (18.8.7) being the parametric restriction. (See also section 17.2).

The parameter λ, (= dual variable of (18.8.7)) is limited by an upper limit, which is $\lambda \leq 2$, if no other limit applies. In the absence of approximation-error $\lambda = 1$ corresponds to complete removal of reference to the "offending" group of restrictions from the Pseudo-Lagrangean. We would therefore not normally expect this limit to be reached.

The parameter λ is also restricted by the parametric displacement leading to one of the amply fulfilled restrictions becoming exactly fulfilled $\lambda \leq \lambda_k$, $\lambda_k = \lambda(f_k = 0)$.

This gives rise to a search operation. For each i for which $f_i(\underline{x}^{**}) > 0$, $p_i > 0$ applies we evaluate the roots of (18.3.4), if any positive root is found, or the smallest of two positive roots, or the real and positive part of a pair of complex root - for a non-convex restriction, the calculated root is compared with the existing limit on λ, if a lower limit is identified in this way, we conclude k = i, and reduce the limit on λ.

If the actual parametric step results in $\lambda > 1$, $\lambda = 1$ is substituted for it.

The parameter, now interpreted as a proportionality factor, is now adjusted upwards according to the following correction formula

$$\lambda(c) = \lambda \text{ sum } t/\text{sum } p \qquad (18.8.8)$$

where sum t is the total sum of all the (estimates of) the dual variables, sum p is the sum of all those dual variables p_i, for which $f_i > 0$ applies. The first downward adjustment is now performed on those p_i for which $f_i(\underline{x}^{**}) > 0$ applies, as follows:

for i = k (the restriction which eventually limited λ, if any)

$$p_k(\text{new}) = (1 - 0.25\,\lambda(c))\, p_k(\text{old}) \qquad (18.8.9)$$

substitute zero instead, if p_k(new) found negative, and for $i \neq k$,

$$p_i(\text{new}) = (1 - 0.5\ \lambda(c))\ p_i(\text{old}) \qquad (18.8.10)$$

substitute zero instead, if p_i(new) found negative.

Note that this first downward correction never leads to the complete removal of all dual variables.

If there was no total loss of optimal form some dual variables are left-undisturbed, if there was total loss of optimal form, we find sum t = sum p, hence $\lambda(c) = \lambda \leq 1$, and not more than 25% is taken of p_k, and not more than 50% of any other p_i. We then have a second (middle) upward correction. The dual variables of which $f_i \leq 0$ applies are adjusted upwards in the same way as would apply in the case of normal transition under optimality.

Finally, the adjustment round is ended with a concluding proportional downward adjustment. The total of all dual variables as it was at the start of the round (or at the end of the previous round, see below), is compared with the total of all dual variables as it has become.

If the new sum is 90% of the old sum or less, the round of adjustment is complete, otherwise all dual variables are adjusted proportionally downward, making the new sum equal to 90% of the old sum. The "old sum" is in this connection:

either the sum calculated when optimal form was found to be violated for the first time

or the figures assigned to the old sum at the end of the round, which is either the new sum at that stage or 90% of the old sum, as the case may be. Adjustments of dual variables by other loops of the programme (e.g. increasing the dual variable of a particular restriction when a second leg of its approximation is found to be binding) are then classified as being part of the middle correction.

When a round of correcting optimal form is complete and no new objective function limit has been formulated at its start, we generally refer back to older approximations, rather than forming new ones. We start with the approximations that were in force when the last objective function limit was formulated (which are recorded at that moment).

If any current approximations (current at the moment of establishing loss of optimal form) are found to describe a

permitted area which is entirely included in the one permitted by the older one, the old approximation is replaced by the current one.

The same applies when optimal form is lost in first instance (as distinct from not as yet having restored it) for tangential approximations, and more generally for approximations taken nearer to the true restriction than the older one.

The latter feature ensures in particular that a restriction which is instrumental in limiting the dual parametric adjustment, is represented by a tangential approximations at least if this happens in the first round.

This procedure ensures that, if several rounds of parametric adjustment be needed the restrictions which were in force at the start of the first round continue to be obeyed.

A proof of convergence to optimal form (and thereby in the convex case to being able to identify a new objective function limit), may now be stated, as follows:

Suppose by contra-assumption, that an endless series of correcting dual adjustments rounds took place. The primal solution vector would then converge to the constrained maximum of the objective function (relative to the set of restrictions stated at the beginning of the first round or in a more limited feasible space area).

Since new dual variables, not approaching zero would be stated in the middle upward correction round, the correction factor sum t/sum p in (18.8.4) would approach infinity. Therefore any remaining dual variable associated with an amply fulfilled restriction would be completely removed, and given the convergence of the primal solution to the constrained maximum of the objective function only, optimal form would then be restored.

18.9 Overview of the SCM algorithm

In this section we recapitulate the sequentially constrained maximization algorithm and its capabilities. We first offer an outline scheme. This outline primarily covers its application to convex problems. It also contains some loops which specifically relate to non-convexity, but only insofar as they are computationally integrated with the main body of the algorithm.

There are also some adaptations of the "ordinary" QP algorithm, as described in Chapters 16 and 17; these are discussed later in this section.

The outline scheme is as follows:

Stage 0: Initiate

Set all matrices and vectors which do not contain initial data at zero, and all logical variables in their "normal" preferred state.

Set the namelists at zero, thus indicating that there are no binding restrictions. Set any relevant indices at zero.

Stage 1: Approximate

Apply (18.3.4) afresh for each quadratic restriction. Quadratic restrictions which are currently associated with binding approximations are made with adjustment. For $f_i > 0$, $\underline{x} = \underline{x}^*_i$ is calculated with outward adjustment, for $f_i < 0$, $\underline{x} = \underline{x}^*_i$ is calculated with inward adjustment. If there is no binding approximation, $\underline{x}^*_i$ is taken to be the current solution vector $\underline{x} = \underline{x}^{**}$, the same applies for $f_i = 0$, or if the approximation restriction is artificially satisfied.

Stage 2: Start a new or modified subsidiary problem

Assemble the pseudo-Lagrangean. Enter the primal and the dual restrictions and the objective function limit, if any, in the ordinary QP tableau. Set the upper limits of the dual variables. Set the re-entry-parameter at REENTRY = 0; marking the normal entry-mode, starting from the trivial basis.

Stage 3: Reapproximate and solve

When coming here in the normal re-entry-mode (REENTRY = 1), reapproximate any quadratic restrictions which are not represented by binding approximations, and enter their updated form in the appropriate slots of the ordinary QP tableau. Enter the ordinary QP algorithm, record the primal solution vector obtained, and calculate the corresponding vector $\underline{f}(\underline{x})$.

Stage 4: Verify the primal solution vector

If the ordinary linear restrictions cannot be met, declare the problem empty and exit. If $f_i > 0$ applies in relation with some binding approximation-restriction, search for the "most overtight" one (= maximal value of $f_i(\underline{x}) \cdot (1 + p_i)$). Move the most overtight approximation parametrically outwards, record the solution vector obtained, and if the corresponding true restriction is actually reached by the parametric adjustment step, adjust dual variables upwards as follows: If $f_i < 0$ applies both before and after the parametric step, increase

the dual variable by 50% of what would apply with normal transition under optimality. For the adjusted restriction itself, use only 25% of the correction factor. If any overtight approximation restriction has been signalled go back to Stage 1. If, having established absence of overtightness there are nevertheless artificial variables in the basis, declare the true problem empty and exit.

Stage 5: Check on looseness

If, for some i, we find $f_i < 0$ in association with a non-binding approximation-restriction, set REENTRY = 1, and go back to Stage 3.

Stage 6: Check status $(m+1)^{th}$ restriction slot

If the $(m+1)^{th}$ restriction is binding, in association with IN-INDEX > 0 (= a second leg of a blocked superimposition), increase the dual variable in question in the same way as applies for normal transition under optimal form (see section 18.7), and go back to Stage 2. If the $(m+1)^{th}$ restriction is not binding, in association with IN-INDEX > 0, set IN-INDEX = 0, put the updated form of the objective function limit in the $(m+1)^{th}$ slot, set REENTRY = 1, and go back to Stage 3. If the $(m+1)^{th}$ restriction is binding, in association with IN-INDEX = 0, (= the objective function limit), proceed directly to Stage 8.

Stage 7: Superimpose

If CUTN = 2 • Q (= the number of superimpositions made since last going through either Stage 0 or Stage 8 already is twice the number of quadratic restrictions), proceed directly to Stage 8.

If DUAL R I > 0, and for R = DUAL R I we find $f_r(x) < 0$, (= optimal form has not yet been restored after loss of optimal form, and this restriction was critical in limiting the downward adjustment of the dual variables, by becoming exactly fulfilled in a parametric step) superimpose a new approximation of $f_r(\underline{x})$, regardless of the probability of successful superimposition. If DUAL R I = 0, search for the most violated true restriction for which successful superimposition appears probable. If no superimposition is indicated, proceed to Stage 8, otherwise put the old binding approximation in the $(m+1)^{th}$ slot, the new approximation in the "regular" slot for the restriction to be superimposed, set IN INDEX according to the restriction in question, set REENTRY = 1, and go back to Stage 3.

Stage 8: Verify optimal form

Set CUTN = 0.

If optimal form is confirmed, set OPTIMAL = true DUAL R I = 0, DUAL REDUCED = false. Copy the available set of current approximations in the matrix of back-copies, and proceed to stage 9.

If optimal form is not confirmed, set OPTIMAL = false (do not as yet disturb the logical variable DUAL REDUCED, this permits to distinguish between not yet having restored optimal form and having just lost optimal form).

Stage 8a: Correction of optimal form

If DUAL REDUCED = false (= optimal form was lost after last going through Stage 1, as distinct from not yet having restored it), set OLDSUM as being the sum of all dual variables. Adjust the dual variables that are associated with $f_i \leq 0$ upwards in the same way as would apply for normal transition under optimal form (= middle round, but for technical reasons, i.e. needing the undisturbed ordinary QP tableau, this is done first).

Form the correction restriction, put the updated form of its approximation in slot m+2, make the dual parametric step, and perform the first semi-proportional downward adjustment as outlined in Section 18.8. If a particular f_i is critical in limiting the downward adjustment of the dual variables, assign its index to DUAL R I, and make a new (tangential) approximation.

Set NEWSUM equal to the sum of all dual variables. If NEWSUM < 0.9 × OLDSUM, set OLDSUM equal to NEWSUM, otherwise multiply all dual variables by 0.9 × OLDSUM/NEWSUM and OLDSUM by 0.9.

Assemble a combined set of approximations from the current set and the back-copy as follows: If DUAL REDUCED = false, choose for each ith restriction the approximation taken at the lowest absolute value of f_i (thus preferably selecting a tangential approximation). Otherwise start with the back-copy, and replace individual approximations by newer ones, only if a scan of their coefficients proves that they exclude all vectors $\underline{x}$ excluded by the old one. Put the selected set of approximations in the back-copy. Set DUAL REDUCED = true, and go back to Stage 2.

Stage 9: Test for final optimality

If any f_i is found to be $f_i < -EPS$ (= negative and differing from zero with a set tolerance, e.g. EPS = 0.000001) proceed to Stage 10.

If any residual shadowprice of a binding approximation-restriction is found to be $|p^*_i| > EPS$, also proceed to Stage 10, the same applies to the shadowprice of the objective-function limit, if binding. If none of these conditions is present, declare the problem solved and exit. (N.B. $p^*_i < 0$ is possible with a non-convex pseudo-Lagrangean.)

Stage 10: Attend to the objective function limit

If the convexity requirements for formulating an objective function limit are met (= all restrictions convex or peripherally convex with a tangential approximation), copy the current set of approximations in the back-copy, and write a new objective function limit. If OPTIMAL = false, go back to Stage 8a.

Stage 11: Normal transition under optimality

Adjust the dual variables upwards, conform Section 18.7. Go back to Stage 1.

END OF OUTLINE SUMMARY

It will be noted from the above outline-summary that there is no exit on indication of unboundedness. The ordinary QP algorithm has such an exit but in fact its activation is limited to problems which contain variables of type absolute, "free" variables. The computational implementation of the SCM algorithm offered in the next section carries the facility to declare variables "free" over into the SCM algorithm, but its use is not part of the proof-supported application of the algorithm.

Equations can be used readily as far as linear equations are concerned, but quadratic restrictions are inequalities only! Initial approximations can be wide-out and to declare them as equations gives rise to obvious problems of spurious emptiness.

We now come to discuss non-convexity and the adaptations in the ordinary QP algorithm made to the purpose of solving non-convex problems. The QURO procedure (= Quadratic Restrictions and Objective function), as listed in the next section, employs an adaptation of the QUAP procedure from Chapter 17. These adaptations relate not only to non-convexity but also to the dependence of the objective function limit on the approximation restrictions.

We now summarise these adaptations as follows:

Firstly, when entering the adapted QUAP procedure in the normal mode with REENTRY = 0, PR = PC = 0 (= asking for essentially the same action as QUAD, solving a QP problem, starting from the trivial basis), the primal restrictions are divided into three distinct categories, viz., the proper linear restrictions, the approximations of the quadratic restrictions, and the objective function limit. The main QURO procedure calls in the compulsory "forced" convex mode, but the non-convex mode of operation is employed nevertheless, until all linear restrictions are satisfied.

During this "non-convex mode Phase I", the two other classes of restrictions are totally disregarded by the search operations. They may at this stage be flown through in either direction, from satisfied to violated or vice versa.

Once the proper linear restrictions are satisfied, the convex mode of operation is resumed. The result of this separate treatment of proper linear restrictions is that emptiness of a subsidiary problem comes in two modalities. If the proper linear restrictions cannot be met, exit from the adapted QUAP procedure is via the alarm exit for emptiness with REENTRY = -1. Emptiness of the true problem is then established immediately, irrespective of (non) convexity. If the approximation restrictions or the objective function limit cannot be met exit from the adapted QUAP procedure is technically successful, but artificial variables are left in the basis. It is then left to the main QURO procedure to analyse the implications of this situation.

In a convex problem emptiness is also proven, but in a non-convex problem the issue of over-tightness arises, and we may try to have a go at getting better approximations. The differential treatment of the approximations on the one hand and the last restriction on the other hand is mainly motivated by reasons which are at least as pertinent to convex problems as to non-convex ones.

In the vicinity of the optimum, the objective function limit converges to a linear combination of the other restrictions. This gives rise to two problems, viz. (in)accuracy and a danger of non-convergence. The accuracy problem might arise if the objective function limit became binding as well as the restrictions on which it is (almost) dependent. At the optimal solution itself such a solution cannot be developed as zero pivots are not accepted, but, with incomplete convergence, an ill-conditioned basis might be developed, giving rise to accuracy problems.

The other problem which might arise is that a consistent solution might be developed, where on account of rounding error, the objective function limit was more stringent than the corresponding combination of the other restrictions.

One could end up with the pseudo-Lagrangean being constrained by the objective function limit only, yet no final optimality as a positive shadowprice of the objective function limit indicates understatement of dual variables. (This problem can only arise with a consistent solution as otherwise some approximation would be classified as loose.)

Of these two problems the first one did actually arise during testing with elements of the QP tableau-matrix indicating multipliers above 1000 and falling out of the format-range normally catered for by the tableau-printing procedure; the second did not.

To counter both problems, the last restriction is (during normal entry) excluded as a pivotal row as well as badname-variable, until at the very end of the call to the ordinary QP algorithm, when it is, if violated, declared badname.

However, if the absolute value of the slack is then found to be less than a set tolerance, it is declared to be a zero irrespective of its sign and return to the main algorithm follows, without actually making the last restriction binding.

The third adaptation concerns the problem of getting the "right" local optimum of the pseudo-Lagrangean. This is understood as a local optimum in which approximation restrictions rather than non-negativities or proper linear restrictions are binding.

In terms of the internal structure of the ordinary QP algorithm, there are two groups of textual adaptations which relate to this problem. One is the selection of the badname variable: shadow-prices of approximation-restrictions are not selected as badname-variable until all negative valued slack-variables (other than the objective function limits), and negative-valued shadowprices of non-negativities or proper linear restrictions have been eliminated or made positive. The other group of textual adaptations relates to "flying through". The version of the ordinary QP algorithm discussed in Chapter 16, permits and indeed prefers, to turn negative valued variables directly into positive-valued variables.

The adaptation then consists of applying the "conservative" version of the rule of the smallest quotient, insofar as approximation-restrictions are concerned, i.e. to eliminate

rather than turn positive, negative-valued slacks of approximation-restrictions.

Finally by calling the ordinary QP algorithm in the "forced" convex mode of operation, shadowprices of approximation-restrictions are permitted to stay negative, if the restriction in question displays manifest non-convexity in its slack-variable subspace, by way of a positive non-zero diagonal cell of the QP tableau.

This is a feature of the QUAP procedure in the version listed in Chapter 17 as well, but it has been taken out as far as proper linear restrictions and the last restriction (which can be the second leg of an approximation in the re-entry mode) are concerned.

For approximation-restrictions this device comes into its own; these negative shadowprices are used in exactly the same way as positive ones, leading, of course, to a reduction in the dual variable in question.

In the convex case, convergence of the SCM algorithm is a fairly obvious result of the presence of the objective function limit and the ensured return to optimal form. We therefore end this section by reviewing the likely effectiveness of the SCM algorithm on non-convex problems where it lacks the support of a proof of its effectiveness.

If we forget the constraint qualification problem, the necessary and sufficient conditions for a constrained local maximum are: complementary slackness (= consistence + optimal form), the first-order conditions (= zero residual shadowprices of approximation restrictions), and convexity within a subspace of tangential approximations, which is ensured by finding a locally constrained maximum of the subsidiary problem.

In short, activation of the successful exit of the SCM algorithm proves that a local maximum of the true problem has been found. There is, however, the problem of actually getting there. To be precise, there are two problems on that score, viz. spurious emptiness and lack of convergence.

The first problem arises because an approximation of a non-convex problem can actually face the "wrong" way, i.e. impose a requirement not to get anywhere near the true solution.

Example (from Section 15.5)

Maximise $\tau = -x_1^2 - x_2^2$

Subject to $(x_1-3)^2 + (x_2-4)^2 \geq 25$

$x_1 + x_2 \geq 4$

$(0 \leq x_1 \leq 100, \quad 0 \leq x_2 \leq 100)$

If the restriction $x_1 + x_2 \geq 4$ is removed from this problem (it was added in Section 15.5, solely to exclude the point $x_1 = x_2 = 0$), the trivial basis of the first subsidiary problem is correctly identified as an optimum. But if the linear restriction is put, starting as one normally would at $x_1 = x_2 = 0$, leads to finding the true problem empty.

The first subsidiary problem then is

Maximise $-x_1^2 - x_2^2$

Subject to $x_1 + x_2 \geq 4$

$6x_1 + 8x_2 \leq 0$

$(0 \leq x_1 \leq 100), \quad 0 \leq x_2 \leq 100).$

Even though the ordinary QP procedure, as adapted will satisfy the linear restriction $x_1 + x_2 \geq 4$, this results in $6x_1 + 8x_2 \leq 0$ being left with an artificial variable $a_2 = 24$ $(x_1 = 4)$. Since, at $x_1 = 4$, $x_2 = 0$ the true restriction is amply fulfilled, no overtightness is diagnosed and the problem is declared empty.

If, however, a starting solution is taken in the area $x_1 > 3$, $x_2 > 4$, this problem does not arise and we obtain a solution. Which solution depends obviously on the particular starting solution chosen.

To counter this possibility of spurious emptiness, the code has been amended at a late stage suppressing the initialization of $\underline{x}$ as $\underline{x} = 0$, thus permitting the user to suggest a different starting solution.

Concerning the other problem of a possible lack of convergence we may observe that despite the absence of an objective function limit and therefore of a convergence proof in the non-convex case, all examples both in this chapter and in chapter 15 were duly solved.

Exercise 18.9

The following general QP problem
(from section 15.3) is given

Maximise $\tau = x_1 + x_2$

Subject to $x_1 \leq 2,\ x_2 \leq 3,\ x_1 + x_2^2 \leq 5$

$(x_1,\ x_2 \geq 0)$

Starting with $x_1^* = 2,\ x_2^* = 2$

apply the algorithm discussed in this chapter to its solution, until neither the primal nor the dual solution can within the tolerance of rounding to two digits after the decimal point, be distinguished from the true solution $x_1 = 2$, $x_2 = 1.73$, $p_1 = 0.71$, $p_3 = 0.29$, $(p_2 = 0)$ with the correct set of binding restrictions.

Answer with tableau-listings at the end of this chapter; note the implicit upper limits i.e. $(0 \leq x_1 \leq 2;\ 0 \leq x_2 \leq 3)$; instead of explicit linear restrictions, the index of the quadratic restriction therefore becomes 1.

18.10 Code listing of the SCM algorithm

Two stretches of programme-text are listed in this section. They are a calling main programme, and the QURO procedure. No text-listing of the modified version of the ordinary QP algorithm is provided, but the substance of the difference between the QUAP procedure as reported in section 17.7 and the version employed here was surveyed in section 18.9.

The main programme listed below contains the lower limit reinterpretation facility, as discussed in section 10.4 for LP and in section 16.14 for ordinary QP. The generalizations of this device to quadratic side-restrictions is catered for by the code-text following the label REINTERPRET. The call to the QURO procedure as surveyed in the previous section then relates to the problem in the y variables with the lower limits at zero.

The required presentation of the input-data is specified in the comment in the main programme.

The output comes in two stages. First the dual variables of the quadratic restrictions are reported by the main programme. Thereafter the result reporting procedure discussed in section 16.15 is called in unamended form, i.e. the primal and

dual solutions of the last subsidiary problem are reported, with re-interpretation according to the true lower limits. The primal solution of this problem is identical to the solution-vector of the true problem. However, the dual solution is identical only insofar as this refers to the shadowprices of linear restrictions be it non-negativities of specified lower or upper limits, or binding linear side-restrictions. The reported dual variables associated with the linear approximations would normally be zero within the confines of rounding; the exception would be where the problem was assumed to be empty.

The text listings indicated are now given, as follows:

TEXT-LISTING OF THE MAIN GENERAL QP PROGRAMME.

```
'BEGIN' 'INTEGER' M,Q,N,NAV,NEQ,REENTRY,R,I,J,PC;
'BOOLEAN' OPTIMAL,FEASIBLE,EMPTY;

'PROCEDURE' QURO(M,Q,N,NEQ,NAV,PC,T,AB,D,X,U,P,ROWLST,COLLST,
OPTIMAL,FEASIBLE,EMPTY);
'INTEGER' M,Q,N,NEQ,NAV,PC; 'ARRAY' T,AB,D,X,U,P;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'BOOLEAN' OPTIMAL,FEASIBLE,EMPTY;
'ALGOL';


'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;  'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWL,COLL);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'PROCEDURE' REPQ(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL; 'ALGOL';

'COMMENT'
GENERAL QUADRATIC PROGRAMMING ALGORITHM.
CONVEX QUADRATIC OBJECTIVE FUNCTION AND LINEAR AS WELL AS
CONVEX QUADRATIC SIDE RESTRICTIONS.
WHEN USED ON NON-CONVEX PROBLEMS, THE ALGORITH NOR-
MALLY WORKS, BUT THERE IS NO PROOF-SUPPORT IN THE
NON-CONVEX CASE.

FOR DETAILS OF THE ALGORITHM SEE THE TEXT OF THE PROCEDURE QURO.

PRESENTATION OF DATA:

FIRST THE TOTAL NUMBER OF RESTRICTIONS I.E. M,
THEN THE NUMBER OF QUADRATIC RESTRICTIONS, I.E. Q,
THEN THE NUMBER OF VARIABLES, I.E. N,
FOLLOWED BY THE NUMBER OF EQUATIONS,  NEQ,
```

THE NUMBER OF EQUATIONS SHOULD NOT EXCEED THE NUMBER OF
LINEAR RESTRICTIONS,
AS ONLY LINEAR EQUATIONS ARE ACCOMMODATED.

THEREAFTER, PUNCH NAV, THE NUMBER OF VARIABLES TO WHICH
THE TACIT (NON-NEGATIVITY) RESTRICTION DOES N O T APPLY.
(IF NAV>0 IS SUPPLIED, THE USER SHOULD ENSURE
BOUNDEDNESS OF ALL SUBSIDIARY PROBLEMS, AS MALFUNCTIONING
MIGHT OTHERSISE ARISE.)

THEREAFTER PUNCH EACH ROW OF THE
M+3 BY N+2 COMPOSITE MATRIX

```
     A      B  0
     C      F  0,  -1, OR +1
    -W      0  0
     U      0  0
     L      0  0
```

THE M-Q BY N+1 LEADING BLOCK A,B REPRESENTING

A.X < OR = B
THE LINEAR RESTRICTIONS, (THE ZERO BEING IRRELEVANT)

AND THE Q BY N+1 BLOCK C, F REPRESENTING
C.X < OR = F,
THE LINEAR COMPONENT OF THE QUADRATIC RESTRICTIONS,
THE K TH QUADRATIC RESTRICTION BEING

$$C(K)' \cdot X - 0.5\ X' \cdot D(K) \cdot X < \text{OR} = F(K).$$

HERE THE (N+2)ND ELEMENT HAS SIGNIFICANCE, AS FOLLOWS:
0 = CONVEXITY OF THIS RESTRICTION UNKNOWN,
CONVEXITY ASSUMED UNTIL SHOWN TO BE NON-CONVEX.
-1 = THIS RESTRICTION IS CONVEX,
INVESTIGATION OF CONVEXITY INHIBITED.
+1 = THIS RESTRICTION IS NOT CONVEX.

THE M+1 TH ROW REPRESENTS THE LINEAR COMPONENT OF THE
OBJECTIVE FUNCTION,

MAXIMIZE $W' \cdot X + 0.5\ X' \cdot D \cdot X$

HERE THE (N+1)TH CELL CONTAINS THE INITIAL VALUE OF THE
OBJECTIVE FUNCTION, THE (N+2)ND CELL IS IRRELEVANT.

THE M+2 ND ROW OF THE LINEAR BLOCK IS RESERVED FOR THE
VECTOR U, REPRESENTING UPPER LIMITS

X < OR = U. (CELLS N+1 AND N+2 IRRELEVANT)

THE M+3 D ROW IS RESERVED FOR THE VECTOR L, OF LOWER
LIMITS, REPRESENTING

X > OR = L. (CELLS N+1 AND N+2 IRRELEVANT)

```
THEREAFTER,NOT AS PART OF THE MAIN INPUT-MATRIX,
PUNCH THE SUGGESTED STARTING SOLUTION FOR X.


THEN PUNCH THE Q+1 SQUARE MATRICES
D,  D(1), D(2), ........  D(Q),
REPRESENTING THE QUADRATIC COMPONENTS OF THE OBJECTIVE FUNCTION,
AND THE Q QUADRATIC RESTRICTIONS.

THESE MATRICES SHOULD BE PUNCHED WITH THE SIGNS IN SUCH A
WAY, THAT THEY ARE NEGATIVE-SEMIDEFINITE IN THE CONVEX
CASE.
;

M:=READ; Q:=READ; N:=READ; NEQ:=READ; NAV:=READ;


  NEWLINE(1);
  WRITETEXT('('NUMBER%OF%LINEAR%RESTRICTIONS')');
  WRITE(30,FORMAT('('S-NDDDD')'),M-Q);
  NEWLINE(1);
  WRITETEXT('('OF%WHICH%EQUATIONS')');
  WRITE(30,FORMAT('('S-NDDDD')'),NEQ);
  SPACE(5);
  'IF' NEQ > M 'THEN' WRITETEXT('('TOO%MANY%EQUATIONS')');
  NEWLINE(1);
  WRITETEXT('('NUMBER%OF%QUADRATIC%RESTRICTIONS')');
  WRITE(30,FORMAT('('S-NDDDD')'),Q);
  SPACE(5);
  'IF' Q > M
  'THEN' WRITETEXT('('TOO%MANY%QUADRATIC%RESTRICTIONS')');
  NEWLINE(2);
  WRITETEXT('('NUMBER%OF%VARIABLES')');
  WRITE(30,FORMAT('('S-NDDDD')'),N);
  NEWLINE(1);
  WRITETEXT('('OF%WHICH%OF%TYPE%ABSOLUTE')');
  WRITE(30,FORMAT('('S-NDDDD')'),NAV);
  SPACE(5);
  'IF' NAV > N
  'THEN' WRITETEXT('('TOO%MANY%OF%TYPE%ABSOLUTE')');
  NEWLINE(1);
  'IF' NAV > 0 'THEN' WRITETEXT('('CAUTION%%VARIABLES WITHOUT%
  SIGN%RESTRICTION%DO%NOT%HAVE%UPPER%LIMITS')');

  'BEGIN' 'ARRAY' TA[1:N+M+4,1:N+M+4],
    AB[1:M+3,1:N+2],X,U,L[1:N],P[1:M],D[1:N,1:N,0:Q+1];

    'INTEGER' 'ARRAY' ROWLST,COLLST[1:N+M+2];


    READ LINEAR PART:
    MATI(AB,M+3,N+2,0,0);
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' X[J]:=READ-AB[M+3,J];

    READ QUADRATIC PART:
```

```
'FOR' R:=0 'STEP' 1 'UNTIL' Q 'DO' 'BEGIN'
  MATI(TA,N,N,0,0);
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  D[I,J,R]:=TA[I,J]; 'END';

PUT FANCYHIGH UPPERBOUNDS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' AB[M+2,J]=0 'THEN' AB[M+2,J] := 1000;

PUT LOWER LIMITS IN PLACE:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' L[J] := AB[M+3,J];

RE INTERPRET:

LINEAR PART OF INITIAL VALUE OF OBJ F:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
AB[M+1,N+1] := AB[M+1,N+1] - AB[M+1,J]*L[J];

QUADRATIC PART OF INITIAL VALUE OF OBJ F:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
AB[M+1,N+1] := AB[M+1,N+1] + 0.5*L[I]*D[I,J,0]*L[J];

DIFFERENTIAL PART OF OBJ F:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
AB[M+1,J] := AB[M+1,J] - L[I]*D[I,J,0];

LINEAR PART OF RHS OF RES:
'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
AB[R,N+1] := AB[R,N+1] - AB[R,J]*L[J];

QUADRATIC PART OF RHS OF RESTR:
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
AB[R,N+1] := AB[R,N+1] + 0.5*L[I]*D[I,J,R-M+Q]*L[J];

DIFFERENTIAL PART OF RESTR:
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
AB[R,J] := AB[R,J] - L[I]*D[I,J,R-M+Q];

REINTERPRET UPPER LIMITS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  AB[M+2,J] := AB[M+2,J]-AB[M+3,J];
  'IF' AB[M+2,J] < 0 'THEN' 'BEGIN'
    NEWLINE(1);
    WRITETEXT('('YOU%HAVE%SUPPLIED%A%LOWER%LIMIT%IN%
    EXCESS%OF%THE%CORRESPONDING%UPPER%LIMIT')');
    NEWLINE(1);
    WRITETEXT('('THE%GENERAL%
    QP%PROBLEM%IS%THEREFORE%EMPTY.')');
    'END';
  'END';
```

```
    PUT UPPERBOUNDS IN PLACE:
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' U[J]:=AB[M+2,J];

    NOW SOLVE:
    QURO(M,Q,N,NEQ,NAV,PC,TA,AB,D,X,U,P,ROWLST,
    COLLST,OPTIMAL,FEASIBLE,EMPTY);

    POINT OF OUTPUT:
    INTERPRET BACK:
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    TA[N+M+PC+4,J]:=L[J];

    NEWLINE(1);
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
      NEWLINE(1); WRITETEXT('('DUAL%VARIABLE')');
      WRITE(30,FORMAT('('SNDDDD')'),I);
      WRITE(30,FORMAT('('S-NDDDD.DDD')'),P[I]); 'END';

    'IF' 'NOT' FEASIBLE 'THEN'
    WRITETEXT('('NO%FEASIBLE%SOLUTION')');

    'IF' N+PC+M < 14
    'THEN' TABO(TA,M+PC+N+1,M+PC+N+1,0,0,1,1,ROWLST,COLLST)
    'ELSE' TABO(TA,M+PC+N+1,0,0,M+PC+N+1,0,1,ROWLST,COLLST);

    NEWLINE(1);
    WRITETEXT('('SOLUTION%VALUE%IF%IN%STANDARD%FORM%
    AND%FEASIBLE')');
    WRITE(30,FORMAT('('S-NDDDD.DDD')'),TA[M+N+PC+2,M+N+PC+2]/2);
    NEWLINE(1);

    REPORT SOLUTION WITH REINTERPRETATION:
    REPQ(TA,M+1,N,NEQ,NAV,ROWLST,COLLST);

    'END'; 'END'


TEXT-LISTING OF THE QURO PROCEDURE.

'PROCEDURE' QURO(M,Q,N,NEQ,NAV,PC,T,AB,D,X,U,P,ROWLST,COLLST,
OPTIMAL,FEASIBLE,EMPTY);
'INTEGER' M,Q,N,NEQ,NAV,PC; 'ARRAY' T,AB,D,X,U,P;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'BOOLEAN' OPTIMAL,FEASIBLE,EMPTY;

'BEGIN' 'INTEGER' I,J,R,K,REENTRY,PR,NAME,CNAME,LAST,RR,
  IN IND,II,JJ,DUAL R I, NNEGD, CUTN, DIRECTION;
  'BOOLEAN' ADJUST,DUAL REDUCED;
  'REAL' FANCYHIGH,LAMBDA,EPS,MOST NEGATIVE,LEAST POSITIVE,
  MOST POSITIVE,CHECKNUM,COP,NUM,DISC,
  OLDSUM,NEWSUM, NEXT NEGATIVE;
  'ARRAY' S,CV[M-Q+1:M],APPR,OLDAPPR[M-Q+1:M+2,1:N+3],
  QUEQ[0:2],ROOT[1:2];
```

```
'PROCEDURE' QUAG(T,M,N,NEQ,NAV,NNEGD,ROWLST,COLLST,PR,PC,
NLINRES,REENTRY,LAMBDA);
'VALUE' NNEGD;
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,NNEGD,PR,PC,
NLINRES,REENTRY;
'REAL' LAMBDA;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'ALGOL';

'PROCEDURE' APPROXIMATE(R,PUT,ADJUST);
'VALUE' R; 'INTEGER' R,PUT; 'BOOLEAN' ADJUST;
'BEGIN' 'INTEGER' TAKE;
  'IF' R=0 'THEN' R:=M+1;
  TAKE := R;
  'IF' R>0 'AND' R<M+1 'THEN' R:=R-M+Q;
  'IF' R=M+1 'THEN' R:=0;
  'IF' R=M+2 'THEN' 'BEGIN'
    R:=Q+1;
    'GOTO' NOW PUT THE APPROXIMATION; 'END';

  'IF' R > 0 'AND' R < Q+1 'THEN' R:=R+M-Q;

  'IF' ADJUST 'THEN' 'BEGIN'
    'IF' S[R] > 0 'THEN' DIRECTION := -1
    'ELSE' DIRECTION := 1;
    'IF' PC=1 'OR' PR=1 'THEN' DIRECTION:=1;
    T[N+M+PC+PR+3,N+PC+R] := 1 000 000 000*DIRECTION;
    LIMIT(N+PC+R,R,DIRECTION);
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    'IF' ROWLST[I]=I 'AND' T[I,N+PC+R]>EPS
    'THEN' 'BEGIN'
      'IF' X[I]/T[I,N+PC+R] < T[N+PC+M+PR+3,N+PC+R]
      'THEN' T[N+PC+M+PR+3,N+PC+R]:=X[I]/T[I,N+PC+R]; 'END';
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    'IF' ROWLST[I]=I 'THEN' X[I] := X[I]-T[I,N+PC+R]*
    T[N+PC+M+PR+3,N+PC+R]; 'END';

  APPR[PUT,N+3] := APPR[PUT,N+2];

  'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
  APPR[PUT,J]:=AB[TAKE,J];

  'IF' R > M-Q 'AND' R < M+1 'THEN' R:=R-M+Q;

  NOW PUT THE APPROXIMATION:
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    APPR[PUT,J]:=APPR[PUT,J] - X[I]*D[I,J,R];
    APPR[PUT,N+1]:=APPR[PUT,N+1]-0.5*X[I]*D[I,J,R]*X[J];
    'END';

  'IF' R > 0 'AND' R < Q+1 'THEN' R:=R+M-Q;

  'IF' ADJUST 'THEN' 'BEGIN'
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    'IF' ROWLST[I]=I 'THEN' X[I] := X[I]+T[I,N+PC+R]*
    T[N+PC+M+PR+3,N+PC+R]; 'END';
```

```
  'IF' PUT < M+1 'THEN' 'BEGIN'
    APPR[PUT,N+2] := APPR[PUT,N+1];
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    APPR[PUT,N+2] := APPR[PUT,N+2]-APPR[PUT,J]*X[J]; 'END';

  'IF' R>0 'AND' R<M+1
  'THEN' 'GOTO' END OF APPROXIMATION;

  'IF' 'NOT' PUT=M+1 'THEN' 'GOTO' END OF APPROXIMATION;

  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  APPR[M+1,J] := -APPR[M+1,J];
  APPR[M+1,N+1] := 0;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  APPR[M+1,N+1] := APPR[M+1,N+1]+X[J]*APPR[M+1,J];
  'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
  APPR[M+1,N+1]:=APPR[M+1,N+1] + P[R]*S[R];
  APPR[M+1,N+2] := APPR[M+1,N+1];

  END OF APPROXIMATION: 'END';

'PROCEDURE' UPDATE(R,PUT);
'INTEGER' R,PUT;
'BEGIN'
  PUT AS COLUMN AND AS ROW:
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  T[I,PUT]:=T[PUT,I]:=APPR[R,I];
  'FOR' I:=N+1 'STEP' 1 'UNTIL' N+PC+M+PR+2 'DO'
  T[I,PUT]:=T[PUT,I]:=0;

  POSTMULTIPLY BY MINUS THE PIVOT INVERSE:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'IF' COLLST[J] = -J 'THEN' T[PUT,J]:=0;
    'IF' COLLST[J] = 2000+J
    'THEN' T[PUT,J] := -T[PUT,J]; 'END';

  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+PR+PC+1 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[I] = I  'THEN'
  T[PUT,J] := T[PUT,J] - T[I,J]*T[I,PUT];

  ATTEND DIAGONAL CELL:
  T[PUT,PUT]:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' ROWLST[I]=I
  'THEN' T[PUT,PUT] := T[PUT,PUT] - T[PUT,I]*T[I,PUT];

  REVERSE SIGN FOR ARTIFICIAL VAR COLUMNS:
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[N+I] = 3000+I 'THEN'
  T[PUT,N+I] := -T[PUT,N+I];

  PUT CORRESPONDING COLUMN:
  'FOR' I:=1 'STEP' 1 'UNTIL' N+PC+M+PR+1 'DO'
  'IF' ROWLST[I] > 0 'THEN' T[I,PUT]:=T[PUT,I]
  'ELSE' T[I,PUT] := -T[PUT,I];
```

```
  PUT DUAL UL:
  T[N+PC+M+PR+3,PUT]:=SQRT(FANCYHIGH);

  PUT UPDATED RHS :

  RHS OF NORMAL RESTR:
  'IF' R>M-Q 'AND' R<M+1
  'THEN' 'BEGIN'
    T[PUT,N+PC+M+PR+2]:=APPR[R,N+2];
    T[N+PC+M+PR+2,PUT]:=-APPR[R,N+2]; 'END';

  END OF UPDATE: 'END';

'PROCEDURE' PREPARE ADJUSTMENT(PUT,R);
'INTEGER' PUT,R;
'BEGIN'
  ATTEND COLUMN:
  'FOR' I:=1 'STEP' 1 'UNTIL' M+N+3 'DO'
  'FOR' J:=PUT 'STEP' 1 'UNTIL' N+M+3 'DO'
  T[I,N+M+4+PUT-J]:=T[I,N+M+3+PUT-J];
  'IF' PUT=N+1 'THEN'
  'FOR' I:=1 'STEP' 1 'UNTIL' N, N+2 'STEP' 1 'UNTIL' R-1,
  R+1 'STEP' 1 'UNTIL' N+M+3 'DO' T[1,N+1]:=-T[I,N+1+R];

  ATTEND ROW:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+4 'DO'
  'FOR' I:=PUT 'STEP' 1 'UNTIL' N+M+3 'DO'
  T[N+M+4+PUT-I,J]:=T[N+M+3+PUT-I,J];
  'IF' PUT=N+1 'THEN'
  'FOR' J:=1 'STEP' 1 'UNTIL'  N, N+2 'STEP' 1 'UNTIL' R,
  R+2 'STEP' 1 'UNTIL' N+M+4 'DO' T[N+1,J]:=-T[N+1+R,J];

  'IF' PUT=N+1 'THEN' T[N+1,N+1]:=T[N+1+R,N+1+R];

  ADJUST NAMELISTS:
  'FOR' I:=N+M+2 'STEP' -1 'UNTIL' PUT+1 'DO' 'BEGIN'
    ROWLST[I]:=ROWLST[I-1]; COLLST[I]:=COLLST[I-1]; 'END';
  'IF' PUT=N+1 'THEN' 'BEGIN'
    COLLST[N+1]:=N+1; ROWLST[N+1]:=-N-1; 'END'
  'ELSE' 'BEGIN'
    COLLST[PUT]:=-1000-PUT+N;
    ROWLST[PUT]:=1000+PUT-N;   'END';

  END OF ADJUSTMENT PREPARATION: 'END';

'PROCEDURE' LIMIT(K,R,DIREC);
'INTEGER' K,R,DIREC;
'BEGIN'

  'FOR' J:=0,1 'DO' QUEQ[J] := 0;  QUEQ[2] := APPR[R,N+1];

  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[I]=I 'THEN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[J]=J 'THEN'
  QUEQ[0]:=QUEQ[0] + 0.5*T[I,K]*D[I,J,R+Q-M]*T[J,K];
```

```
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[J]=J 'THEN'
  QUEQ[1]:=QUEQ[1] + APPR[R,J]*T[J,K];

  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  QUEQ[2] := QUEQ[2] - APPR[R,J]*X[J];

  'IF' ABS(QUEQ[0]) < 0.000000001 'THEN' 'GOTO' LIN ADJ;

  CALCULATE DISC:
  DISC := QUEQ[1]*QUEQ[1] - 4*QUEQ[0]*QUEQ[2];

  'IF' DISC < 0 'THEN' 'GOTO' NEAREST TO;
  ROOT[1] := 0.5*(-QUEQ[1]+SQRT(DISC))/QUEQ[0];
  ROOT[2] := 0.5*(-QUEQ[1]-SQRT(DISC))/QUEQ[0];

  'IF' ROOT[1]*DIREC < 0 'THEN' ROOT[1] := ROOT[2];
  'IF' ROOT[2]*DIREC < 0 'THEN' ROOT[2] := ROOT[1];
  'IF' ROOT[1]*DIREC < 0 'AND' PR=1
  'THEN' 'GOTO' END OF UPPER LIMIT CALCULATION;
  'IF' ROOT[2]*DIREC < ROOT[1]*DIREC
  'THEN' ROOT[1]:=ROOT[2];
  'IF' ROOT[1]*DIREC < T[N+PC+M+PR+3,K]*DIREC
  'THEN' T[N+PC+M+PR+3,K]:=ROOT[1];
  'GOTO' END OF UPPER LIMIT CALCULATION;

  LIN ADJ:
  'IF' QUEQ[1] = 0 'THEN'  'BEGIN'
    'IF' 'NOT' PR=1 'THEN' T[N+PC+M+PR+3,K]:=0;
    'GOTO' END OF UPPER LIMIT CALCULATION; 'END';
  'IF' DIREC*QUEQ[2]/QUEQ[1] > 0 'THEN' 'BEGIN'
    'IF' 'NOT' PR=1 'THEN' T[N+PC+M+PR+3,K] := 0;
    'GOTO' END OF UPPER LIMIT CALCULATION; 'END';
  'IF' -DIREC*QUEQ[2]/QUEQ[1] < T[N+PC+M+PR+3,K]*DIREC
  'THEN' T[N+PC+M+PR+3,K]:= -QUEQ[2]/QUEQ[1];
  'GOTO' END OF UPPER LIMIT CALCULATION;

  NEAREST TO:
  'IF' -DIREC*QUEQ[1]/QUEQ[0] < 0 'THEN' 'BEGIN'
    'IF' 'NOT' PR=1
    'THEN' T[N+PC+M+PR+3,K] := 0;
    'GOTO' END OF UPPER LIMIT CALCULATION; 'END';
  'IF' -DIREC*0.5*QUEQ[1]/QUEQ[0] < T[N+PC+M+PR+3,K]*DIREC
  'THEN' T[N+PC+M+PR+3,K]:=-0.5*QUEQ[1]/QUEQ[0];

  END OF UPPER LIMIT CALCULATION: 'END';

'REAL' 'PROCEDURE' CORRECTION(R,I);
'INTEGER' R,I;
'BEGIN' 'REAL' CF;
  CF := 0;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[J]=J 'THEN' CF:=CF+ABS[R,J]*T[J,N+I];

  'FOR' II:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' JJ:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ROWLST[JJ]=JJ 'THEN'
  CF := CF - X[II]*D[II,JJ,R-M+Q]*T[JJ,N+I];
  'IF' CF > 0.5 'THEN' CF := 1/CF 'ELSE' 'BEGIN'
    'IF' CF < 0 'THEN' CF:=0.5 'ELSE' CF:=0.999; 'END';
```

```
  'IF' CF>0.999 'THEN' CF:=0.999;

  CORRECTION := CF;
  END OF THE CORRECTION CALCULATION: 'END';


'PROCEDURE' RECORD;
'BEGIN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'IF' ROWLST[J]=J 'THEN'
    X[J]:=T[J,N+PC+M+PR+2];
    'IF' ROWLST[J] = -J 'THEN' X[J]:=0;
    'IF' COLLST[J]=2000+J 'THEN' X[J]:=U[J]; 'END';

  CALCULATE QUADRATIC SLACKS:
  'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    S[R]:=AB[R,N+1];
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    S[R]:=S[R]-X[J]*AB[R,J];
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    S[R] := S[R] + 0.5*X[I]*D[I,J,R+Q-M]*X[J];  'END';

  'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' S[I] > EPS 'AND' 'NOT' ROWLST[N+PC+I]=1000+I
  'AND' AB[I,N+2]=0 'THEN' AB[I,N+2] := S[I];

  END OF PRIMAL SOLUTION RECORDING: 'END';

'PROCEDURE' COPY BEST;
'BEGIN'
  'FOR' I := M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'

    'IF' 'NOT' DUAL REDUCED 'AND'
    ABS(APPR[I,N+2])<ABS(OLDAPPR[I,N+2])
    'THEN' 'GOTO' COPY;
    'IF' NAV > 0 'THEN' 'GOTO' DO NOT COPY;
    'IF' 'NOT' APPR[I,N+1]*OLDAPPR[I,N+1]>0
    'THEN' 'GOTO' DO NOT COPY;

    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'

      'IF' APPR[I,J]=0 'AND' 'NOT' OLDAPPR[I,J]=0
      'THEN' 'GOTO' DO NOT COPY;

      'IF' APPR[I,J]=0 'THEN' 'GOTO' NEXT;
      'IF' APPR[I,N+1] > 0  'AND' APPR[I,J] > 0 'AND'
      APPR[I,N+1]/APPR[I,J]>OLDAPPR[I,N+1]/OLDAPPR[I,J]
      'THEN' 'GOTO' DO NOT COPY;

      'IF' APPR[I,N+1] < 0 'AND' APPR[I,J] < 0 'AND'
      APPR[I,N+1]/APPR[I,J]<OLDAPPR[I,N+1]/OLDAPPR[I,J]
      'THEN' 'GOTO' DO NOT COPY;
      'IF' APPR[I,N+1] > 0  'AND' APPR[I,J] < 0 'AND'
      APPR[I,N+1]/APPR[I,J]<OLDAPPR[I,N+1]/OLDAPPR[I,J]
      'THEN' 'GOTO' DO NOT COPY;
      'IF' APPR[I,N+1] < 0 'AND' APPR[I,J] > 0 'AND'
      APPR[I,N+1]/APPR[I,J]>OLDAPPR[I,N+1]/OLDAPPR[I,J]
      'THEN' 'GOTO' DO NOT COPY;
```

```
      NEXT: 'END';

    COPY:
    'FOR' J:=1 'STEP' 1 'UNTIL' N+3 'DO'
    OLDAPPR[I,J]:=APPR[I,J];

    DO NOT COPY: 'END';

  'FOR' J:=1 'STEP' 1 'UNTIL' N+3 'DO'
  'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
  APPR[I,J]:=OLDAPPR[I,J];

  END OF COPY BEST: 'END';


'COMMENT'
QUADRATIC PROGRAMMING ALGORITHM.
THE ABBREVARION QURO STANDS FOR
QU  ADRATIC    R  ESTRICTIONS AND    O  BJECTIVE FUNCTION.

N SPECIFIED VARIABLES,
M RESTRICTIONS, OF WHICH M-Q LINEAR RESTRICRIONS,
AND Q QUADRATIC RESTRICTIONS.

THE SIGNIFICANCE OF THE PARAMETERS M,N,NEQ,NAV,ROWLST AND
COLLST WILL BE OBVIOUS,
WHEN SEEN IN COMBINATION WITH THE QUAG PROCEDURE.
AB,D, AND U ARE INPUT-ARRAYS, X AND P ARE OUTPUT-ARRAYS.

AS FAR AS THE OPERATIVE USE BY THE QURO-PROCEDURE
IS CONCERNED (AS DISTINCT FROM STORING LIMITS),
AB IS ASSUMED TO BE OF ORDER M+1 BY N+2.

THE (N+2)ND COLUMN IS OPERATIVELY USED, ONLY FOR THE
QUADRATIC RESTRICTIONS, TO STORE INFORMATION CONCERNING
THEIR CONVEXITY.
AB[R,N+2] < 0 MEANS CONVEX, NO VERIFICATION
AB[R,N+2] = 0 MEANS PROBABLY CONVEX, TO BE VERIFIED
AB[R,N+2] > 0 MEANS NON-CONVEX.

THE LEADING M-Q BY N+1 BLOCK-ROW IS RESERVED FOR THE
LINEAR RESTRICTIONS,
THE NEXT Q BY N+1 BLOCK-ROW FOR THE LINEAR PART OF THE
QUADRATIC RESTRICTIONS.

THE 'EXTRA' ROW (INDEX M+1) IS RESERVED
FOR THE LINEAR COMPONENT OF THE OBJECTIVE FUNCTION.

D IS ASSUMED TO BE A THREE-DIMENTIONAL ARRAY,
OF ORDER N BY N BY Q+2.
EACH OF THE Q+1 SQUARE N BY N MATRICES, INDICES ZERO TO Q,
IS EXPECTED TO CONTAIN AN INPUT MATRIX  D.

THE FIRST OF THESE MATRICES (INDEX ZERO) REPRESENTS THE
QUADRATIC COMPONENT OF THE OBJECTIVE FUNCTION,
MAXIMISE  W' X  +  0.5  X' D X.
THE OTHERS (INDICES 1 TO Q), REPRESENT THE QUADRATIC
COMPONENTS OF THE QUADRATIC RESTRICIONS
B  +  C' X  +  0.5  X' D X  >  OR = 0
THE LAST OF THE MATRICES D, INDEX Q+1, IS RESERVED FOR
INTERNAL USE BY THE QURO-PROCEDURE.
```

```
THE N DIMENTIONAL ARRAY U IS ASSUMED TO CONTAIN THE VECTOR OF
UPPERBOUNDS ON X.
THE ARRAYS X AND P ARE SPACE-RESERVATIONS FOR THE VECTORS
OF PRIMAL AND DUAL VARIABLES.
OF THESE, X MUST BE PRE-FILLED BY THE USER, P IS INI-
TIALIZED AT ZERO BY THE PROCEDURE.
;

DUAL REDUCED := 'FALSE';
FANCYHIGH:=10000; EPS:=0.000001;
DUAL R I := 0; OLDSUM:=0;

STAGE 0:
INITIATE AT ZERO:
'FOR' I:=1 'STEP' 1 'UNTIL' N+M+2 'DO' ROWLST[I]:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N+M+2 'DO' COLLST[J]:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' P[I]:=0;
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' S[I]:=0;
ADJUST := 'FALSE'; PR:=PC:=0;
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M+2 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N+3
'DO' OLDAPPR[I,J]:=APPR[I,J]:=0;

STAGE 1:
APPROXIMATE RESTRICTIONS:
IN IND:=0;
DUAL R I := CUTN := 0;
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  'IF' COLLST[N+PC+R] = 1000+R
  'THEN' ADJUST := 'TRUE'  'ELSE' ADJUST := 'FALSE';
  'IF' ADJUST 'THEN' APPROXIMATE(R,R,'FALSE');
  APPROXIMATE(R,R,ADJUST); 'END';

STAGE 2:
SET THE PSEUDO LAGR:
FEASIBLE := 'TRUE';
PC:=PR:=RR:=0;

CLEAN TABLEAU:
'FOR' I:=1 'STEP' 1 'UNTIL' N+M+4 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' M+N+4 'DO' T[I,J]:=0;

ASSEMBLE PSEUDO LAGRANGEAN:

LINEAR PART OF PSEUDO L:
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' T[I,M+N+2]:=AB[M+1,I];
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' T[I,M+N+2]:=
T[I,M+N+2] + P[R]*AB[R,I];

QUADRATIC PART OF PSEUDO L:
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  T[I,J] := D[I,J,0];
  'FOR' K:=1 'STEP' 1 'UNTIL' Q 'DO'
  T[I,J] := T[I,J] + P[K-Q+M] * D[I,J,K]; 'END';
```

```
ASSEMBLE REST OF TABLEAU:

PUT LINEAR RESTR:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M-Q 'DO' T[N+I,J] := AB[I,J];
'FOR' I:=1 'STEP' 1 'UNTIL' M-Q 'DO'
T[N+I,N+M+2] := AB[I,N+1];

PUT QUADRATIC APPROXIMATIONS AND EXTRA RESTR:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M+1 'DO'
T[N+I,J] := APPR[I,J];
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M+1 'DO'
T[N+I,N+M+2] := APPR[I,N+1];

PUT INITIAL VALUE:
T[N+M+2,N+PC+M+2]:=AB[M+1,N+1]*2;

PUT UPPER LIMITS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' T[N+M+3,J]:=U[J];
'FOR' I:=N+1 'STEP' 1 'UNTIL' N+M+1 'DO'
'IF' I<N+M-Q+1 'THEN' T[N+M+3,I]:=FANCYHIGH
'ELSE' T[N+M+3,I]:=SQRT(FANCYHIGH);

REENTRY:=0; LAMBDA:=0;

STAGE 3:
SOLVE:
NNEGD := N+M+100;

'IF' OPTIMAL 'AND' REENTRY > 0 'THEN' 'BEGIN'
  'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[N+PC+R]=1000+R
  'AND' T[N+PC+R,N+PC+M+2]>0 'AND' S[R]<0
  'THEN' 'BEGIN'
    ADJUST := 'FALSE';
    APPROXIMATE(R,R,ADJUST);
    UPDATE(R,N+PC+R); 'END'; 'END';


OPTIMAL := 'TRUE';

RECALL QUAG:
'IF' REENTRY=2 'THEN' NNEGD := 0;
QUAG(T,M+PR+1,N+PC,NEQ,NAV,NNEGD,ROWLST,COLLST,0,PC,
M-Q,REENTRY,LAMBDA);

'IF' REENTRY=100 'THEN' 'BEGIN'
  'COMMENT'
  A NON-CONVEX SUBSIDIARY PROBLEM HAS BEEN ENCOUNTERED;
  NEWLINE(1);
  WRITETEXT('('CARRY%ON%WITH%SOLUTION%OF%A%
  NON%CONVEX%PROBLEM')');
  'END';

'IF' ROWLST[N+PC+M+1] = M+1001
'AND' 'NOT' (PC=1 'AND' ROWLST[N+1]=N+1)
'AND' T[N+PC+M+1,N+PC+M+2] < 0 'THEN' 'BEGIN'
  REENTRY := 1;
  QUAG(T,M+PR+1,N+PC,NEQ,NAV,NNEGD,ROWLST,COLLST,0,PC,
  M-Q,REENTRY,LAMBDA); 'END';
```

```
COPY NEXT X:
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' CV[I]:=S[I];
RECORD;

'IF' PC=1 'THEN' 'BEGIN'
  'IF' ABS(S[K]) < 0.000001 'THEN' 'BEGIN'
    'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' K-1,
    K+1 'STEP' 1 'UNTIL' M 'DO'
    'IF' S[R]<0 'AND' CV[R]<0
    'AND' ROWLST[N+1+R] = -1001-R 'THEN'
    P[R] := P[R] + 0.5*T[N+R+1,N+M+3]*CORRECTION(R,R+1);
    'IF' ROWLST[N+1+K]=-1000-K 'THEN'
    P[K] := P[K] + 0.25*T[N+1+K,N+M+3]*CORRECTION(K,K+1);
    'END'; 'GOTO' STAGE 1; 'END';



STAGE 4:
CHECK ON OVERTIGHTNESS AND EMPTINESS:

CHECK EMPTINESS ON LINEAR RESTR:
'IF' REENTRY = -1 'THEN' 'BEGIN'
  'COMMENT'
  EMPTINES ESTABLISHED BY ORDINARY QP ALGORITHM
  IN THE NON-CONVEX MODE. THIS RELATES SPECIFICALLY TO
  LINEAR RESTRICTIONS.;
  EMPTY := 'TRUE';
  'GOTO' FINAL END OF QURO; 'END';
EMPTY := 'FALSE';
'FOR' I:=1 'STEP' 1 'UNTIL' M-Q 'DO'
'IF' ROWLST[N+I]=3000+I 'THEN' EMPTY:='TRUE';
'IF' EMPTY 'THEN' 'GOTO' END OF QURO;

CHECK ON OVERTIGHTNESS OF BINDING APPR:
MOST POSITIVE := -1;
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  'IF' ROWLST[N+I] = 3000+I 'AND' S[I]>EPS
  'THEN' 'GOTO' STAGE 1;
  'IF' ROWLST[N+I] = -1000-I
  'AND' S[I]>EPS 'AND' S[I]*(1+P[I])>MOST POSITIVE
  'THEN' 'BEGIN'
    K:=I; MOST POSITIVE := S[K]*(1+P[K]); 'END'; 'END';

'IF' MOST POSITIVE > 0 'THEN' 'BEGIN'
  AB[K,N+2] := 1;
  PREPARE ADJUSTMENT(N+1,K);
  T[N+M+4,N+1] := 1 000 000 000;
  ADJUST := 'FALSE';
  APPROXIMATE(K,K,ADJUST);
  PC:=1; LIMIT(N+1,K,1);
  REENTRY := 2;
  'GOTO' RECALL QUAG; 'END';

CHECK EMPTINESS ON QUADRATIC RESTR:
EMPTY := 'FALSE';
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' AB[I,N+2] > 0 'AND' ROWLST[N+M+1] = 3001+M
'THEN' 'BEGIN'
  APPR[M+1,N+1] := 1 000 000 000;
  'GOTO' STAGE 1; 'END';
```

```
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+I]=3000+I 'AND' S[I]>0 'THEN' 'GOTO' STAGE 1;
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M+1 'DO'
'IF' ROWLST[N+I] = 3000+I 'THEN' EMPTY := 'TRUE';
'IF' EMPTY 'THEN' 'GOTO' END OF QURO;


STAGE 5:
CHECK ON LOOSE APPR:
'FOR' K:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+K]=1000+K 'AND' S[K] < -EPS
'AND' T[N+K,N+M+2] > EPS
'THEN' 'BEGIN'
  REENTRY := 1; 'GOTO' STAGE 3; 'END';

STAGE 6:
CHECK ON M PLUS 1 RESTR:

'IF' ROWLST[N+M+1] = -1001-M 'AND' IN IND > 0
'THEN' 'BEGIN'
  P[IN IND] := P[IN IND] +
  T[N+M+1,N+M+2]*CORRECTION(IN IND,M+1);
  'GOTO' STAGE 2; 'END';

'IF' ROWLST[N+M+1] = -1001-M 'THEN' 'GOTO' STAGE 8;

'IF' IN IND > 0 'THEN' 'BEGIN'
  UPDATE(M+1,N+M+1);
  IN IND := 0;
  'IF' T[N+M+1,N+M+2] < 0 'THEN' 'BEGIN'
    REENTRY := 1; 'GOTO' RECALL QUAG; 'END'; 'END';

STAGE 7:
SUPERIMPOSE:
NEXT NEGATIVE := -1 000 000 000;
START OF APPROXIMATION SEARCH:
R:=0; MOST NEGATIVE := -10*EPS;

'IF' CUTN < 2*Q 'THEN' 'BEGIN'
  'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    'IF' 'NOT' S[I] < MOST NEGATIVE
    'THEN' 'GOTO' DO NOT IMPROVE;
    'IF' 'NOT' S[I] > NEXT NEGATIVE
    'THEN' 'GOTO' DO NOT IMPROVE;
    'IF' AB[I,N+2] > 0 'THEN' 'GOTO' DO NOT IMPROVE;
    R:=I;
    MOST NEGATIVE := S[R]; 'END';

  SAVE THE OLD ONE:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
  APPR[M+2,J]:=APPR[R,J];

  ADJUST:='TRUE';
  APPROXIMATE(R,R,'FALSE');
  APPROXIMATE(R,R,ADJUST); UPDATE(R,N+M+1);
  'IF' ABS(T[N+M+1,N+M+1]) < EPS*EPS 'THEN' 'BEGIN'
    T[N+M+1,N+M+1] := 0;
    'GOTO' NOW PUT IMPROVEMENT; 'END';
  'IF' T[N+R,N+M+1] < EPS 'THEN' 'GOTO' TRY NEXT;
```

```
  'IF' R = DUAL R I  'THEN' 'GOTO' NOW PUT IMPROVEMENT;
  'IF' 'NOT' T[N+R,N+M+2]/T[N+R,N+M+1] <
  T[N+M+1,N+M+2]/T[N+M+1,N+M+1]
  'THEN' 'GOTO' TRY NEXT;

  NOW PUT IMPROVEMENT:
  CUTN := CUTN+1;  IN IND:=R;
  ROWLST[N+R]:=1000+R; ROWLST[N+M+1]:=-1001-M;
  COLLST[N+R]:=-1000-R; COLLST[N+M+1]:=1001+M;

  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+3 'DO' 'BEGIN'
    NUM:=T[N+R,J]; T[N+R,J]:=T[N+M+1,J];
    T[N+M+1,J]:=NUM; 'END';
  'FOR' J:=1 'STEP' 1 'UNTIL' N+M+3 'DO' 'BEGIN'
    NUM:=T[J,N+R]; T[J,N+R]:=T[J,N+M+1];
    T[J,N+M+1]:=NUM; 'END';

  REENTRY:=1; NNEGD:=N+M+1000;
  'GOTO' RECALL QUAG;

  DO NOT IMPROVE: 'END';

TRY NEXT:
'IF' NEXT NEGATIVE > MOST NEGATIVE 'THEN' 'BEGIN'
  NEXT NEGATIVE := MOST NEGATIVE;
  RESTORE THE OLD ONE:
  'IF' R>0 'THEN' 'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
  APPR[R,J] := APPR[M+2,J];

  'GOTO' START OF APPROXIMATION SEARCH; 'END';


STAGE 8:
VERIFY COMPLEMENTARY SLACKNESS:
OPTIMAL := 'TRUE';
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' P[R]*S[R] > EPS  'THEN' OPTIMAL:='FALSE';

'IF' OPTIMAL 'THEN' 'BEGIN'
  SAVE OLD APPR:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+3 'DO'
  'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
  OLDAPPR[I,J] := APPR[I,J];
  DUAL REDUCED := 'FALSE';
  OLDSUM := 0; 'GOTO' STAGE 9; 'END';

REDUCE DUAL:
NEWSUM:=0;
DUAL R I := 0;

INITIATE CORRECTION RESTR AT ZERO:
'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO' APPR[M+2,J]:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' D[I,J,Q+1]:=0;

ASSEMBLE CORRECTION RESTR:
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  CV[R] := 0;
  'IF' 'NOT' DUAL REDUCED 'THEN' OLDSUM:=OLDSUM+P[R];
  'IF' S[R] > 0 'AND' S[R]*P[R] > 0 'THEN' 'BEGIN'
```

```
    'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO'
    APPR[M+2,J]:=APPR[M+2,J]-AB[R,J]*P[R];
    'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    D[I,J,Q+1]:=D[I,J,Q+1]-D[I,J,Q-M+R]*P[R]; 'END';

  'IF' S[R]<0 'AND' ROWLST[N+R]=-1000-R 'THEN'
  CV[R] := T[N+R,N+M+2]*CORRECTION(R,R); 'END';

PREPARE ADJUSTMENT(N+M+2,0);

PR:=1; APPROXIMATE(M+2,M+2,'FALSE');
UPDATE(M+2,N+M+2);
LEAST POSITIVE := T[N+M+4,N+M+2] := 2;

COMPLETE LAST ELEMENT OF UPDATE:
T[N+M+2,N+M+3] := 0;
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  'IF' S[R] > 0 'AND' S[R]*P[R] > 0 'THEN'
  T[N+M+2,N+M+3] := T[N+M+2,N+M+3] - P[R]*S[R]; 'END';

T[N+M+3,N+M+2] := -T[N+M+2,N+M+3];

'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+R]=1000+R 'AND' S[R]*P[R]>EPS
'THEN' 'BEGIN'
  ADJUST := 'FALSE';
  APPROXIMATE(R,R,ADJUST);
  LIMIT(N+M+2,R,1);

  'IF' T[N+M+4,N+M+2] < LEAST POSITIVE
  'THEN' 'BEGIN'
    LEAST POSITIVE := T[N+M+4,N+M+2];
    DUAL R I := R; 'END'; 'END';

CALL QUAG FOR DUAL ADJUSTMENT:
NNEGD := 1000;
REENTRY:=2; PR:=1;  LAMBDA:=0;
QUAG(T,M+2,N+PC,NEQ,NAV,NNEGD,ROWLST,COLLST,PR,PC,
M-Q,REENTRY,LAMBDA);

'IF' LAMBDA > 1 'THEN' LAMBDA := 1;
LAMBDA := LAMBDA*OLDSUM;
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  'IF' S[R]*P[R]>EPS 'THEN' 'BEGIN'
    'IF' 'NOT' R = DUAL R I
    'THEN' P[R] := P[R] - LAMBDA*P[R]/OLDSUM
    'ELSE' P[R] := P[R] - 0.5*LAMBDA*P[R]/OLDSUM; 'END';
  'IF' CV[R] > 0  'THEN' P[R]:=P[R]+CV[R];
  'IF' P[R] < 0 'THEN' P[R] := 0; 'END';

SELECT BEST APPR:
'IF' DUAL R I > 0 'THEN' 'BEGIN'
  RECORD;
  APPROXIMATE(DUAL R I,DUAL R I,'FALSE'); 'END';
COPY BEST;
CUTN := 0; DUAL REDUCED := 'TRUE';
```

```
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
NEWSUM := NEWSUM+P[R];
'IF' NEWSUM < 0.9*OLDSUM 'THEN' 'BEGIN'
  OLDSUM := NEWSUM; 'GOTO' STAGE 2; 'END';
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
P[R] := 0.9*P[R]*OLDSUM/NEWSUM;
OLDSUM := 0.9*OLDSUM;

'GOTO' STAGE 2;

STAGE 9:
INVESTIGATE FINAL OPTIMALITY:
OPTIMAL:=FEASIBLE:='TRUE'; EMPTY:='FALSE';
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  'IF' S[R] < -EPS 'THEN' FEASIBLE := 'FALSE';
  'IF' ROWLST[N+R]<0 'AND'
  ABS(T[N+R,N+M+2]) > EPS 'THEN' OPTIMAL:='FALSE';
  'END';

'IF' ROWLST[N+M+1] = -1001-M
'AND' ABS(T[N+M+1,N+M+2]) > EPS
'THEN' OPTIMAL := 'FALSE';

'IF' OPTIMAL 'AND' FEASIBLE 'THEN' 'GOTO' END OF QURO;
OPTIMAL := 'TRUE';


STAGE 10:
CHECK APPLICABILITY OF OBJ F LIM:
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+R] = -1000-R 'AND' ABS[R,N+2] > 0
'THEN' 'GOTO' STAGE 11;
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+R] = -1000-R 'AND' ABS[R,N+2] < 0
'AND' ABS(APPR[R,N+2]) > EPS
'THEN' 'GOTO' STAGE 11;

CALCULATE NEW OBJECTIVE FUNCTION LIMIT:
COPY OLD APPR IF LAST ONE OPTIM:
'FOR' J:=1 'STEP' 1 'UNTIL' N+3 'DO'
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
OLDAPPR[I,J]:=APPR[I,J];
ADJUST := 'FALSE';
APPROXIMATE(0,M+1,ADJUST);

STAGE 11:
ADJUST DUAL VARIABLES:

'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'

  'IF' 'NOT' I>M-Q 'THEN' 'BEGIN'
    STATE SHADOWPRICE OF LINEAR RESTR:
    'IF' ROWLST[N+I] = 1000+I 'THEN' P[I]:=0;
    'IF' ROWLST[N+I]=-1000-I 'THEN'
    P[I]:=T[N+I,N+M+2];
    'GOTO' END OF PRICE ADJUSTMENT LOOP; 'END';
```

```
  ATTEND UPWARD ADJUSTMENT FOR A QUADRATIC PRICE:
  'IF' S[I]<EPS 'AND' COLLST[N+I] = 1000+I
  'THEN'  P[I]:=P[I]+T[N+I,N+M+2]*CORRECTION(I,I);
  'IF' P[I]<0 'THEN' P[I]:=0;
  END OF PRICE ADJUSTMENT LOOP: 'END';

'IF' ROWLST[N+M+1]=-1001-M 'AND' IN IND > 0 'THEN' 'BEGIN'
  P[IN IND] := P[IN IND] +
  CORRECTION(IN IND,M+1)*T[N+M+1,N+M+2];
  'IF' P[IN IND] < 0 'THEN' P[IN IND] := 0; 'END';

'GOTO' STAGE 1;

END OF QURO:

'IF' EMPTY 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('PROBLEM%ASSUMED%TO%BE%EMPTY%ON%ACCOUNT%
  OF%OCCURRENCE%OF%ARTIFICIAL%VARIABLES')'); 'END';

REAPPROXIMATE LINEAR SLACKS:
'FOR' R:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[N+R]=1000+R 'THEN' 'BEGIN'
  ADJUST := 'FALSE';
  APPROXIMATE(R,R,ADJUST); UPDATE(R,N+R); 'END';

CORRECT SOLUTION VALUE FOR CONSTANTS:
'FOR' I:=M-Q+1 'STEP' 1 'UNTIL' M 'DO'
T[N+M+2,N+M+2]:=T[N+M+2,N+M+2]+AB[I,N+1]*P[I]*2;

CORRECT SOLUTION VALUE FOR INITIAL VALUE:
T[N+M+2,N+M+2] := T[N+M+2,N+M+2] + 2*AB[M+1,N];

FINAL END OF QURO:   'END';
```

To help the reader of this book to read back the text-listing of the QURO procedure, it may be useful to point out some subordinate internal procedures, whose significance may be summarized as follows:

The procedures approximate and update:

To calculate a linear approximation, and to calculate its current form in the updated QP-tableau.

The procedure prepare adjustment:

To re-order the tableau making room for another activity or restriction.

The procedure limit:

To calculate λ, as defined by (18.3.4) and implement appropriate modifications of the parametric adjustment technique.

Answer to exercise 18.9

Stages 0 - 3

Our first subsidiary problem is:

Maximise $x_1 + x_2$

Subject to $x_1 + 4\,x_2 \leq 9$ (the linear approximation)

$(0 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3)$

The objective function limit is at this stage trivial $(0 \leq 0)$.

Stages 4 - 6

The optimum solution of the first subsidiary problem (See tableau 18 exc 1) x_1 at this upper limit of 2; $x_2 = 1.75$.

There are no overtightness, no loose approximation, and no binding extra-restriction.

Stage 7

The (one) most violated true restriction is re-approximated, with adjustment to $x_1 = 2$, $x_2 = 1.73$, the true quadratic restriction being binding at that point. The new approximation is $x_1 + 3.46\,x_2 \leq 8$.

Tableau 18 exc 2 gives the new approximation as violated restriction in the s_1-slot, the binding old approximation in the p_2-slot. The new approximation is probably superimposable, on account of the zero in the s_1/p_1 cell. Hence we return to stage 3.

Stages 3 - 5

After re-entering the ordinary QP algorithm, the optimum of the modified subsidiary problem is found at the consistent solution vector $x_1 = 2$, $x_2 = 1.73$. (See tableau 18 exc. 3).

There is no overtightness and no loose approximation.

Stage 6

We replace the old approximation $x_1 + 4\,x_2 \leq 9$ which is now amply fulfilled by the objective function limit. As the latter so far has the trivial form $0 \leq 0$, no listing of this modification of the subsidiary problem is given.

Stage 7

No superimposition is called for, the solution-vector being consistent.

Stage 8

We find the condition of optimal form satisfied.

Stage 9

We proceed to stage 10, on account of the linear shadowprice $p_1^* = 0.29$.

Stage 10

We form the objective function limit $x_1 + x_2 \leq 3.73$.

Stage 11

The upper limit on the correction-factor is binding therefore

$$p_1 = 0 + 0.999\ p_1^* = 0 + 0.999 \times 0.29 = 0.29$$

(As the current primal solution is a consistent one, cf = 1 would have let to hitting the true dual variable exactly, as it is, the figure cannot be distinguished within the two decimals accuracy of presentation). We return to stage 1.

Stages 1 - 3

The second subsidiary problem is

$$\text{Maximise} \quad x_1 + x_2 + 0.29(5 - x_1 - x_2^2) = 0.71x_1 + x_2 - 0.58x_1^2 + 1.45$$

$$\text{Subject to} \quad x_1 + 3.46\ x_2 \leq 8 \quad \text{(approximation)}$$

$$x_2 + x_2 \leq 3.73 \quad \text{(objective function limit)}$$

$$(0 \leq x_1 \leq 2; \quad 0 \leq x_2 \leq 3)$$

Stages 4 - 9

The optimum solution vector of the second subsidiary problem (See tableau 18 exc 4) is found to be

x_1 at its upper limit of 2; $x_2 = 1.73$

$p_1^* = 0.00$

No further action is called for until stage 9, when we conclude to final optimality.

(The programmed code, operating with a finer tolerance still finds $p_1^* = 0.00029$ at this point, and makes one round more, but within the two decimals fixed point presentation, the optimal tableau cannot be distinguished from tableau 18 exc.4).

ADDENDUM TO THE ANSWER OF EXCERCISE 18.9

SOME TABLEAUX DEVELOPED IN SOLVING EXCERCISE 18.9
(NON-MEANINGFUL UPPER BOUNDS REPLEACED BY XX)

18 EXC 1
FIRST SUBSIDIARY OPTIMUM.

NA.	B1	D2	S1	P2	VAL.
U1	-	0.25	-	-	0.75
X2	-0.25	-	0.25	-	1.75
P1	-	-0.25	-	-	0.25
S2	-	-	-	-	0.00
2T	0.75	-1.75	0.25	-0.00	3.50
UB	2	XX	XX	XX	XX

18 EXC 2
NR 1, WITH NEW APPR ADDED.

NA.	B1	D2	P2	S2	VAL.
U1	-	0.25	0.13	-	0.75
X2	-0.25	-	-	0.25	1.75
S1	-0.13	-	-	-0.87	-0.06
P2	-	-0.25	0.87	-	0.25
2T	0.75	-1.75	0.06	0.25	3.50
UB	2	XX	XX	XX	XX

18 EXC 3
OPTIMUM OF MODIFIED PROBLEM.

NA.	B1	D2	S1	P2	VAL.
U1	-	0.29	-	-0.15	0.71
X2	-0.29	-	0.29	-	1.73
P1	-	-0.29	-	1.15	0.29
S2	0.15	-	-1.15	-	0.07
2T	0.71	-1.73	0.29	-0.07	3.46
UB	2	XX	XX	XX	XX

18 EXC 4
THE FINAL OPTIMUM.

NA.	B1	D2	S1	P2	VAL.
U1	-0.05	0.29	0.05	0.71	0.71
X2	-0.29	-	0.29	-	1.73
P1	0.05	-0.29	-0.05	0.29	0.00
S2	-0.71	-	-0.29	-	0.00
2T	0.71	-1.73	0.00	-0.00	7.46
UB	2	XX	XX	XX	XX

Part V

INTEGER PROGRAMMING

CHAPTER XIX

INTEGER PROGRAMMING AND SOME OF ITS APPLICATIONS

19.1 Some integer programming terms

The term integer programming is normally applied in the context of linear programming. An integer programming problem differs from the corresponding "normal" linear programming problem, by the additional condition that some (or all) variables must have integer values in any admissable solution.

For example, we may have the following problem

Maximise

$$\tau = x_1 + 1.5x_2 + 0.5x_3$$

Subject to

$$x_1 + x_2 + x_3 \leq 20$$

$$x_1 + 0.5x_2 \leq 5$$

$$0.5x_2 \leq 5$$

$$(x_1, x_2, x_3 \geq 0,\ x_1 \text{ integer-valued})$$

This is a mixed integer programming problem

If the requirement that variables should attain integer values applies to all variables, we speak of an all integer programming problem.

In the mixed integer programming problem the integer requirement applies to some, but not necessarily to all variables, (i.e. to x_1 but not to x_2 and x_3 in the example).

Obviously the mixed integer problem is the general case. Any algorithm which is effective with respect to the mixed integer problem would cope with "ordinary" linear programming problems, and also with all-integer problems as special cases of mixed integer problems.

In a mixed integer problem the variables may be separated into two groups, the integer-restricted and the continuous variables.

A special type of integer-restricted variable is the zero-one variable. As the name indicates, a zero-one variable is a variable for which only two values are admitted, zero and one.

A linear programming problem, which differs from a (mixed) integer programming problem, only by the variables not being required to attain integer values, is called the corresponding continuous problem. The optimal and feasible solution of such a corresponding continuous problem (if there is an optimal solution) is then called the (corresponding) continuous optimum.

It should obviously be understood that, in a corresponding continuous problem, zero-one variables are restricted to the interval from zero to one. The restrictions which are written explicitly with coefficients, e.g. $x_1 + x_2 + x_3 \leq 20$, $x_1 + 0.5x_2 \leq 5$ etc., are indicated as the specified restrictions, to distinguish them from the integer requirements, i.e. the requirements that integer-restricted variables attain integer values and of course, also from the tacit non-negativity requirements.

A solution-vector which satisfies the specified restrictions (and any non-negativity requirements) is then a feasible solution of the continuous problem or for short a solution of the continuous problem.

A genuinely feasible solution which satisfies not only the specified restrictions but also the integer requirements is then called an integer solution.

We will use the term "integer solution" irrespective of the fact that in a mixed integer programming problem, some variables are not restricted to integer values and may attain (in an "integer solution" fractional values.

The true optimum may then also be indicated as the integer optimum, to distinguish it from the continuous optimum.

19.2 Fixed costs and decision variables

Mixed integer problems in which the integer-restricted variables are zero-one variables, are the most frequently applied kind of integer programming problem.

They arise mainly from two types of problems which the non-specialist will not immediately recognise as integer programming problems.

The case of fixed costs ("overheads") is relatively straightforward.

In the fixed-cost problem, the integer variable is a decision variable which indicates whether or not a particular activity, e.g. to build a factory, to operate a machine, etc., is at all engaged. Once the factory is built, the machine bought, etc., certain fixed costs are incurred. These fixed costs are independent of the activity level. (See also Westphal [39])

For example, a firm may be required to transport 15,000 tonnes of coal per month.

This will involve costs of transport and the method of transportation will have to be decided. Obviously, one will wish to minimize the cost of transport. If the coal is sent by road, the firm can use the services of road hauliers and no fixed costs for the firm arise. For bulk-transport the use of a railway or a ship may be cheaper, but would require that a rail-connection to the factory, or a loading berth be built first. Once these investment decisions were made, regularly recurring costs of interest and amortization would arise, irrespective of the amount of coal actually transported. Total transport costs are then the sum of the cost of road-transport (per tonne only), the fixed cost of building the railway (if built), the fixed cost of building the loading berth for maritime transport (if the firm chooses to use this mode of transport at all), and the variable costs of both rail and maritime transport. The problem of minimizing transport cost might now, for example be

Minimise

$$2.5x_1 + 600x_2 + 1.5x_3 + 3500x_4 + 0.4x_5$$

Subject to

$$9{,}000x_2 \geq x_3$$

$$70{,}000x_4 \geq x_5$$

$$x_1 + x_3 + x_5 \geq 15{,}000$$

$$(x_1, x_2, x_3, x_4, x_5 \geq 0, \ x_2 \text{ and } x_4 \text{ zero or one})$$

The interpretation would be (for example) as follows: x_1, x_3 and x_5 are the quantities transported by road (x_1), by rail (x_3), and by sea (x_5). These involve variable costs of 2.5, 1.5 and 0.4 units of money per tonne. The decision variable x_2 is then the decision to build the railhead. If

x_2 is zero the 600 units of fixed costs are not involved, but then no rail transport is possible. If the railhead is built ($x_2 = 1$), the 600 units of fixed costs are incurred, enabling transport by rail up to the railhead's maximum capacity of 9000 tonnes. The same principle applies to maritime transport. If the loading berth is not built ($x_4 = 0$) the restriction that x_5 be not greater than 70,000 x_4 prevents a non-zero value of x_5. The above illustrated use of decision variables represented as zero-one variables can obviously be combined with restrictions of a different nature, e.g. market limits, relations with other production processes, etc.

The one additional point which is also useful to mention here is the treatment of general increasing returns to scale. Suppose the relationship between production capacity and cost of investment is verified for certain realistically considered sizes of a particular installation

	Cost of Investment	Capacity
Size No.1	500	1000
Size No.2	700	2100
Size No.3	900	3500

This example differs from the slightly simpler case of "set-up costs" which are independent from the scale of operations.

If the fixed costs were set up cost, plus a linear function of the scale, the tabulation could, for example be as given in the "proportional capacity" version of the tabulation below

	Initial Set-up Cost	Proportional Additional Cost	Total Cost of Investment	Proportional Capacity	Actual Capacity
Size No.1	340	160	500	1000	1000
Size No.2	340	360	700	2250	2100
Size No.3	340	560	900	3500	3500

i.e. the cost of investment is 340 units plus 16 per 100 units of capacity.

As the example was actually put, there are increasing returns to scale at all investigated scales, and size No.2 only has 2100 units of capacity for 700 units of cost. With generally increasing returns to scale, we need a decision variable for every considered size. One simply writes the appropriate relations three times, as if it were seriously considered to build all three installations of sizes No. 1, 2 and 3. The three decision

variables will be indicated as, for example x_1, x_2, x_3. If it is simply uneconomic to build two or three similar installations that is all, only one will figure in the optimum solution. If it is physically impossible to build two similar installations on the same site, one may add that restriction to the model, i.e., $x_1 + x_2 + x_3 \leq 1$.

19.3 Non-convex restrictions and dummy variables

One may also use dummy variables to represent a general non-convex restriction.

Example

Maximise

$$\tau = 2y - x$$

Subject to

$$y = 1 + 4x - 4x^2 + x^3$$

$$x \leq 3 \quad (x \geq 0,\ y \geq 0)$$

The true shape of the restriction $y \leq 4x - 4x^2 + x^3 + 1$ has the typical S-shape of a cubic restriction and is clearly non-convex.

Provided the true non-linear restriction does not have any near-vertical segments, we may approximate it to any degree of required precision by means of linear segments of predetermined length. The more precise approximation we require, the more segments will be needed. For example, with intervals of 0.5 for the value of x, we have the following tabulation (see also the associated graph).

$x = 0.00 \quad y = 1.00$

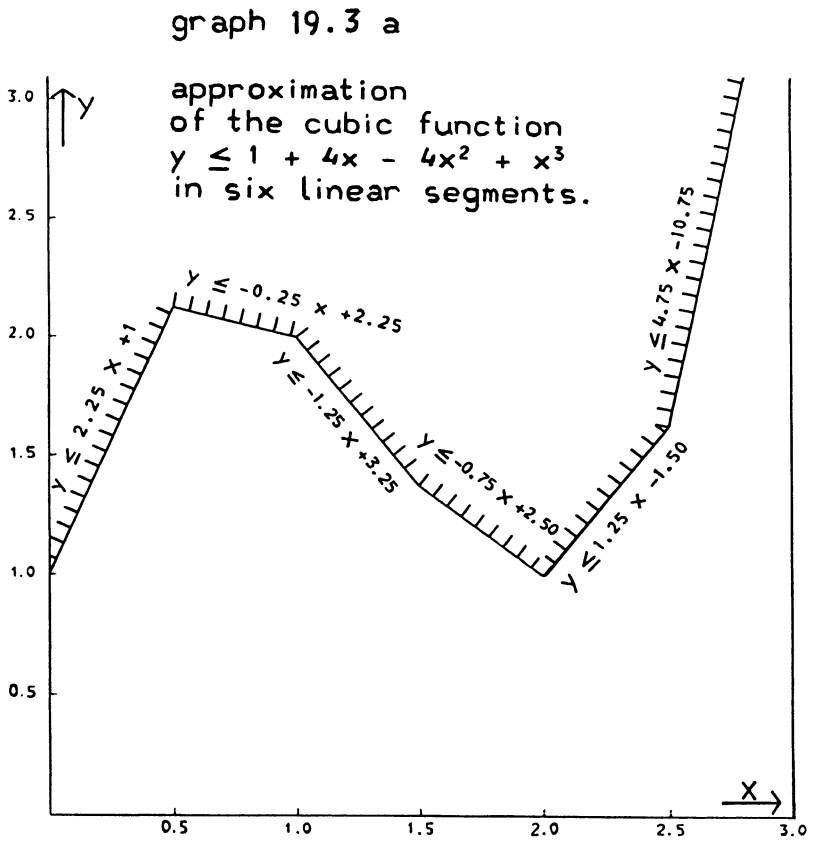
$y_1 \leq 1 + 2.25\ x_1 \qquad (0 \leq x_1 \leq 0.50)$

$x = 0.50 \quad y = 2.13$

$y_2 \leq 2.25 - 0.25\ x_2 \qquad (0.50 \leq x_2 \leq 1.00)$

$x = 1.00 \quad y = 2.00$

$y_3 \leq 3.25 - 1.25\ x_2 \qquad (1.00 \leq x_3 \leq 1.50)$


graph 19.3 a
approximation
of the cubic function
y ≤ 1 + 4x - 4x² + x³
in six linear segments.
Y
X
3.0
2.5
2.0
1.5
1.0
0.5
0.5
1.0
1.5
2.0
2.5
3.0
y ≤ 2.25 x +1
y ≤ -0.25 x +2.25
y ≤ -1.25 x +3.25
y ≤ -0.75 x +2.50
y ≤ 1.25 x -1.50
y ≤ 4.75 x -10.75

$x = 1.50 \quad y = 1.37$

$y_4 \leq 2.50 - 0.75\, x_4 \qquad (1.50 \leq x_4 \leq 2.00)$

$x = 2.00 \quad y = 1.00$

$y_5 - 1.50 + 1.25\, x_5 \qquad (2.00 \leq x_5 \leq 2.50)$

$x = 2.50 \quad y = 1.63$

$y_6 \leq -10.25 + 4.75\, x_6 \qquad (2.50 \leq x_6 \leq 3.00)$

$x = 3.00 \quad y = 4.00$

There remains the problem of actually linking the "true" values of x and y to these segments. We will discuss this issue at first, only for x. We introduce a series of zero-one restricted variables, $d_1, d_2 \ldots\ldots d_6$, and a series of requirements

$$x_1 - 99\, d_1 \leq x \leq x_1 + 99\, d_1$$

$$x_2 - 99\, d_2 \leq x \leq x_2 + 99\, d_1 \qquad \text{etc.}$$

To identify x with just one segment we add the equation

$$d_1 + d_2 + d_3 + d_4 + d_5 + d_6 = 5,$$

i.e. five of the dummy variables are equal to one, and one of them is zero.

The restriction is formally written in terms of the dummy-representations of x i.e.

$$y \leq 2.25\, x_1 + 1$$

$$y \leq -0.25\, x_2 + 2.25$$

$$y \leq -1.25\, x_3 + 3.25$$

$$y \leq -0.75\, x_4 + 2.50$$

$$y \leq 1.25\, x_5 - 1.50$$

$$y \leq 4.75\, x_6 - 10.25$$

The dummy representations of x are defined by the intervals, e.g.

$$0 \leq x_1 \leq 0.5,\ 0.5 \leq x_2 \leq 1.0,\ 1.0 \leq x_3 \leq 1.5 \text{ etc.}$$

The variable x has to be equal to one of them, i.e. if the corresponding zero-one variable d_2 is zero the pair of restrictions

$$\left.\begin{aligned} -99\ d_2 - x_2 + x \leq 0 \\ -99\ d_2 + x_2 - x \leq 0 \end{aligned}\right)$$

requires that x is equal to x_2, when d_2 is zero but when d_2 is one, this pair of restrictions requires virtually nothing.

The number 99 has been taken to represent a "very high number" and could very well be taken higher, except that for typographical reasons, the 99 was more convenient.

For a convex restriction we would not need the integer programming technique at all. This only arises because the linear segments corresponding to the non-convex (upwardly curved) parts of the curve actually cut into the admitted area.

Accordingly, some condensation of the problem is possible, if several consecutive segments together form a convex segment. In the example at hand, this is the case for the first three segments. Algebraically this is recognised, by means of comparing the slope-coefficients.

The first segment ($0 \leq x \leq 0.5$) has a slope-coefficient of 2.25 the slope coefficient of the next segment ($0.5 \leq x \leq 1.0$) is only -0.25, the slope of the third segment ($1.0 \leq x \leq 1.5$) is even less - going down even steeper - i.e. -1.25.

After that the slope increases again, i.e. becomes less negative 0.75. Therefore, the convex segment $0 \leq x \leq 1.5$ requires only one dummy. Thus, the three segments now become

$$y \leq 2.25\ x_1 + 1$$

$$y \leq -0.25\ x_1 + 2.25$$

$$y \leq -0.75\ x_1 + 3.25$$

and x_1 now covers the interval $0 \leq x_1 \leq 1.50$.

The problem is now tabulated.

The programmed procedure listed at the end of this section actually produces an end result which includes the problem-reinterpretation discussed in section 10.4, but the tabulation given in tableau 19.3a contains the original consrants of the various restrictions, with all variables set at zero.

TABLEAU 19.3 A

THE CUBIC INTEGER PROGRAMMING PROBLEM.

INTEGER R.			CONTINUOUS VARIABLES									
D1	D2	D3	X1	X2	X3	X4	X	Y	D4		VALUE	
-1	-1	-1	-	-	-	-	-	-	-1	=	-3	DUMMIES-EQ.
-99	-	-	-2.25	-	-	-	-	1	-	≤	1	CON-
-99	-	-	0.25	-	-	-	-	1	-	≤	2.25	VEX
-99	-	-	1.25	-	-	-	-	1	-	≤	3.25	SEGMENT
-	-99	-	-	0.75	-	-	-	1	-	≤	2.50	SEPERATE
-	-	-99	-	-	-1.25	-	-	1	-	≤	-1.50	NON-CONVEX
-	-	-	-	-	-	-4.75	-	1	-99	≤	-10.25	SEGMENTS
-99	-	-	-1	-	-	-	1	-	-	≤	-	RELATIONS
-	-99	-	-	-1	-	-	1	-	-	≤	-	BETWEEN X
-	-	-99	-	-	-1	-	1	-	-	≤	-	AND THE
-99	-	-	1	-	-	-1	1	-	-99	≤	-	SEGMENTAL
-	-99	-	-	1	-	-	-1	-	-	≤	-	X - DUMMIES
-	-	-99	-	-	1	-	-1	-	-	≤	-	X1 TO X4
-	-	-	-	-	-	1	-1	-	-99	≤	-	
-	-	-	-	-	-	-	1	-2	-		-	OBJECTIVE F
1	1	1	1.50	2	2.50	3	3	4	1		X	UPPER LIMITS
-	-	-	-	1.50	2	2.50	-	-	-		-	LOWER LIMITS

Methods of integer programming as such, i.e. finding solutions to already stated integer programming problems are the subject of the next two chapters. At this point it is, however useful to comment on the possibility to automate the assembly of the tableau, and to eliminate redundant restrictions and variables.

The procedure offered below writes the restrictions which arise from the segmentation of one polynomial restriction of the type.

$$y \leq \sum_{p=0}^{pm} c_p x^p \qquad (19.3.1)$$

It is obviously possible to have several restrictions of the similar type in one problem, and have them written in the same tableau calling the procedure several times. The call to the procedure to be listed below should therefore supply for a polynomial restriction:

the full size of the tableau
the index of the column which refers to y
the index of the column which refers to x
the numbers characterising
the polynomial function, i.e. the index mp (maximum power) and mp + 1 coefficients c_p giving the coefficients associated with x^o (the constant) until x^{mp}.
the number of segments
the index of the column where space is reserved for the first zero one variable
the index of the column where space is reserved for the first column representing x in a particular interval
the upper and lower limits on x

We offer the text of a procedure which assembles the appropriate part of the tableau on the indication of that information.

The listed procedure does more than was indicated in the text so far. The two additional points are the following: Firstly, there is the issue of near-equivalence of two or more adjoining segments. If the number of segments is put fairly large initially, it may easily happen that the approximation would not become noticeably worse, if some segments were merged, i.e. were replaced by only one linear segment.

In the example, segments 3 and 4 ($1.0 \leq x \leq 1.5$ and $1.5 \leq x \leq 2$) came most near to that possibility. The criterion which the procedure applies is that the difference between the slope-coefficients of two adjoining sections, multiplied with their total length, is less than a given tolerance, which is set at

0.01. Thus segments 3 and 4 have slope coefficients of -1.25 and -0.75 respectively, i.e. their difference is 0.50. The total length of the two segments is 0.50 + 0.50 = 1 full unit of x. Hence the tolerance would have to be set at just above 0.50 before these two segments are to be merged.

If the tolerance were set at 0.51 rather than at 0.01, the two segmented restrictions would be replaced by a single one

$$y \leq 3.00 - 1.00x \qquad (1.00 \leq x \leq 2.00)$$

linking the points x = 1.00, y = 2.00 and x = 2.00, y = 1.00 directly. The main reason why this particular loop of the programmed procedure (situated in the programme-text below the label MERGE ALTOGETHER:), did not become operative in the example is that only 6 segments is a rather crude approximation in the first place.

The other point which has not been discussed so far is the redundancy of the integer (zero-one) requirement on the "last" one of any series of dummy variables.

In the example (as it became after condensation) d_1, d_2 and d_3 are only allowed to attain the values zero or one. There also is a requirement $d_1 + d_2 + d_3 + d_4 = 3$. It is therefore sufficient to require that d_4 is in the interval between zero and one, and it is not necessary to verify that d_4 is in fact at one of the two ends of this interval. In this connection, the tableau is re-ordered once more, and the "extra" zero-one variable is put at the end of the tableau rather than with the integer-restricted variables.

TEXT-LISTING OF THE MAIN PROGRAMME USED TO ASSEMBLE AND SOLVE THE CUBIC INTEGER PROGRAMMING PROMLEM.

```
'BEGIN' 'INTEGER' N,NAV,I,J,MM,NN,RN,RBN,NLINRES,NEQ,NPOLYRES,P,
ROWINDEX,COLINDEX,RESTRN,IFZOV,IFDX,IFDXDF,MEQ,NZOV,
EXITTYPE,IX,IY,EQN,MMM,NNN, OUT M, OUT N;
'REAL' ALPHA;

'PROCEDURE' POLY(T,COEFVEC,ALPHA,M,N,MAXPOW,NSE,LOVX,HIVX,
MINUMX,MAXUMX,EQN,RESTRN,IFZOV,IX,IY,IFDX,IFDXDF,RN);
'INTEGER' M,N,MAXPOW,NSE,EQN,RESTRN,IFZOV,
IX,IY,IFDX,IFDXDF,RN;
'REAL' ALPHA,LOVX,HIVX,MINUMX,MAXUMX;
'ARRAY' T,COEFVEC;
'ALGOL';
```

```
'PROCEDURE' ZOLP(INTA,OUTA,OUTM,OUTN,T,M,N,NEQ,NAV,NZOV,
OUTROWLST,ROWLST,COLLST,EXITTYPE,NSL);
'INTEGER' OUTM,OUTN,M,N,NEQ,NAV,NZOV,EXITTYPE,NSL;
'ARRAY' INTA,OUTA,T;
'INTEGER' 'ARRAY' OUTROWLST,ROWLST,COLLST;
'ALGOL';


'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;   'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';


  'COMMENT'
  PROGRAMME TO MAXIMIZE A LINEAR OBJECTIVE FUNCTION,
  SUBJECT TO NLINRES LINEAR RESTRICTIONS, AND NPOLYRES
  POLYNOMIAL RESTRICTIONS.
  EACH POLYNOMIAL RESTRICTION LINKS ONE OF THE VARIABLES
  TO A POLYNOMIAL FUNCTION OF ONE OF THE OTHER VARIABLES.

  OF THE RESTRICTIONS, NEQ ARE ALLOWED TO BE EQUATIONS,
  ALL THE EQUATIONS MUST BE BE LINEAR EQUATIONS.

  OF THE N VARIABLES, NAV ARE ALLOWED TO BE OF ABSOLUTE TYPE,
  I.E. NOT TO BE ASSOCIATED WITH NON NEGATIVITY RESTRICTIONS.

  THE PROGRAMME DOES THIS BY ASSEMBLING AN MM BY NN INTEGER
  LINEAR PROGRAMMING TABLEAU, IN WHICH EACH POLYNOMIAL
  RESTRICTION IS REPRESENTED BY LINEAR SEGMENTS.
  FOR DETAILS OF THIS LINEAR PRESENTATION, SEE THE TEXT OF
  THE PROCEDURE POLY.
  ;

  READ MAIN PROBLEM PARAMETERS:
  NLINRES:=READ; NEQ:=READ; NPOLYRES:=READ; N:=READ; NAV:=READ;

  'BEGIN'
    'ARRAY' LOVX,HIVX,MINUMX,MAXUMX[1:NPOLYRES];
    'INTEGER' 'ARRAY' MAXPOW,NSE[1:NPOLYRES];


    READ MAIN PARAMETERS OF SEPERATE POLYNOMIALS:

    'FOR' RN:=1 'STEP' 1 'UNTIL' NPOLYRES 'DO' 'BEGIN'

      READ THE INDEX FOR THE HIGHEST POWER:
      MAXPOW[RN]:=READ;

      HOW MANY SEGMENTS FOR THIS RESTRICTION:
      NSE[RN]:=READ;

      WHICH INTERVAL:
      LOVX[RN]:=READ;   HIVX[RN]:=READ;
```

```
  WHICH TOTAL INTERVAL:
  MINUMX[RN]:=READ; MAXUMX[RN]:=READ;

  'END';

CALCULATE ORDER PARAMETERS OF MAIN TABLEAU:
MM:=NLINRES; NN:=N;
'FOR' RN:=1 'STEP' 1 'UNTIL' NPOLYRES 'DO' 'BEGIN';
  MM:=MM+1+3*NSE[RN]; NN:=NN+2*NSE[RN]; 'END';

'IF' NEQ > NLINRES 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('TOO%MANY%EQUATIONS')');
  NEWLINE(1);
  WRITETEXT('('THE%NUMBER%OF%EQUATIONS%EXCEEDS%THE%
  NUMBER%OF%LINEAR%RESTRICTIONS')'); 'END'
'ELSE' MEQ:=NEQ;


'BEGIN' 'ARRAY' T[1:MM+3,1:NN+2];

  INITIATE TABLEAU AS ZERO MATRIX:
  'FOR' I:=1 'STEP' 1 'UNTIL' MM+3 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' NN+2 'DO' T[I,J]:=0;

  PUT LINEAR INPUT DATA TEMPORARY IN TOP BLOCK ROW:
  MATI(T,NLINRES+3,N+1,0,0);

  TRANSPORT TO APPROPRIATE BLOCKS:
  'FOR' I:=1+MEQ 'STEP' 1 'UNTIL' NLINRES+3 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO' 'BEGIN'
    T[MM-NLINRES+I,J]:=T[I,J]; T[I,J]:=0; 'END';

  'FOR' I:=1 'STEP' 1 'UNTIL' MEQ,
  MM+MEQ-NLINRES+1 'STEP' 1 'UNTIL' MM+3 'DO'
  'FOR' J:=NAV+1 'STEP' 1 'UNTIL' N+1 'DO' 'BEGIN'
    T[I,NN-N+J]:=T[I,J]; T[I,J]:=0; 'END';


  ATTEND THE POLYNOMIAL RESTRICTIONS:

  INITIATE INDICES FIRST BLOCK:
  ROWINDEX:=MEQ+NPOLYRES; COLINDEX:=NAV; NZOV:=0;
  EQN:=MEQ; IFZOV:=NAV;

  'FOR' RN:=1 'STEP' 1 'UNTIL' NPOLYRES 'DO' 'BEGIN'
    EQN:=EQN+1; MEQ:=MEQ+1;
    RESTRN:=ROWINDEX; ROWINDEX:=ROWINDEX+NSE[RN];
    IFDXDF:=ROWINDEX; ROWINDEX:=ROWINDEX+NSE[RN];
    COLINDEX:=COLINDEX+NSE[RN]; NZOV:=NZOV+NSE[RN];
    IFDX:=COLINDEX; COLINDEX:=COLINDEX+NSE[RN];

    'BEGIN' 'ARRAY' COEFVEC[0:MAXPOW[RN]];
      READ AND ASS INFORM PARTICULAR RESTR:

      ALPHA:=READ;
```

```
    IX:=READ;
    'IF' IX>NAV 'THEN'
    'FOR' RBN:=1 'STEP' 1 'UNTIL' NPOLYRES 'DO'
    IX:=IX+2*NSE[RBN];

    IY:=READ;
    'IF' IY>NAV 'THEN'
    'FOR' RBN:=1 'STEP' 1 'UNTIL' NPOLYRES 'DO'
    IY:=IY+2*NSE[RBN];

    'FOR' P:=0 'STEP' 1 'UNTIL' MAXPOW[RN] 'DO'
    COEFVEC[P]:=READ;

    NOW ASSEMBLE THE RN RESTR:
    MMM:=MM; NNN:=NN;
    POLY(T,COEFVEC,ALPHA,MM,NN,MAXPOW[RN],NSE[RN],
    LOVX[RN],HIVX[RN],MINUMX[RN],MAXUMX[RN],EQN,RESTRN,
    IFZOV,IX,IY,IFDX,IFDXDF,RN);

    RESET INDICES:
    ROWINDEX:=ROWINDEX+MMM-MM; COLINDEX:=COLINDEX+NNN-NN;
    NZOV:=NZOV-ENTIER(0.5*(NNN-NN+1))-1;
    'END'; 'END';


'BEGIN' 'ARRAY' TA[1:MM+3,1:NN+2],
  OUTA[1:MM+NN+NZOV,1:1];
  'INTEGER' 'ARRAY' ROWLST[1:MM],COLLST[1:NN],
  OUTROWLST[1:NZOV+MM+NN];

  PUT ZEROS TO PREVENT OVERFLOW:
  'FOR' I:=1 'STEP' 1 'UNTIL' MM+NN+NZOV 'DO'
  OUTA[I,1]:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' MM 'DO' ROWLST[I]:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' NN 'DO' COLLST[I]:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' NZOV+MM+NN 'DO'
  OUTROWLST[I]:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' MM+2 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' NN+1 'DO' TA[I,J]:=0;

  ZOLP(T,OUTA,OUTM,OUTN,TA,MM,NN,MEQ,NAV,NZOV,
  OUTROWLST,ROWLST,COLLST,EXITTYPE,100);

  'IF' EXITTYPE=0 'THEN' TABO(OUTA,OUT M,0,0,0,1,0,
  OUTROWLST,COLLST)
  'ELSE' TABO(TA,OUT M,OUT N,0,0,1,1,ROWLST,COLLST);
  'END'; 'END'; 'END'; 'END'
```

TEXT-LISTING OF THE PROCEDURE WHICH SEGMENTS ONE SEPERATE POLYNOMIAL RESTRICTION.

```
'PROCEDURE' POLY(T,COEFVEC,ALPHA,M,N,MAXPOW,NSE,LOVX,HIVX,
MINUMX,MAXUMX,EQN,RESTRN,IFZOV,IX,IY,IFDX,IFDXDF,RN);
'INTEGER' M,N,MAXPOW,NSE,EQN,RESTRN,IFZOV,
IX,IY,IFDX,IFDXDF,RN;
'REAL' ALPHA,LOVX,HIVX,MINUMX,MAXUMX;
'ARRAY' T,COEFVEC;

'BEGIN' 'INTEGER' I,J,II,JJ,NSEC,SE,P;
  'REAL' LOWV,HIGV,X,Y,XDIF,XP,DYDX,FANCYHIGH,COP,SLDIF,MAXUMY;

  'COMMENT'
  PROCEDURE TO ASSEMBLE A SEGMENTED APPROXIMATION OF THE POLY-
  NOMIAL RESTRICTION

  ALPHA * Y   < OR =   POLINOMIAL FUNCTION OF X.

  THE FUNCTION IS EXPECTED TO BE NON-CONVEX.
  IN EACH SEGMENT, X IS REPRESENTED BY ASSOCIATED DUMMY VARIABLES.
  THESE DUMMY VARIABLES ARE RESTRICTED TO A PARTICULAR
  INTERVAL.
  ALSO ASSOCIATED ARE ANOTHER SERIES OF DUMMY VARIABLES.
  THOSE DUMMY VARIABLES ARE ZERO-ONE VARIABLES, ONE OF THEM
  WILL BE ZERO AND IDENTIFY AND MAKE EFFECTIVE A PARTICULAR LINEAR
  SEGMENT OF THE FUNCTION, WHILE THE OTHERS ARE AT THE VALUE ONE.

  THE SIGNIFICANCE OF THE PARAMETERS IS AS FOLLOWS:

  T  IS AN M BY N  LINEAR PROGRAMMING TYPE TABLEAU, OR M+2 BY
  N+2,  IF THE MARGINS ARE INCLUDED.

  COEFVEC  IS THE COEFFICIENTS VECTOR WHICH CHARACTERIZES THE
  POLYNOMIAL FUNCTION.
  THE TERMS BY WHICH X POWER ZERO (THE CONSTANT), X POWER ONE,
  X POWER TWO, UNTIL X POWER MAXPOW,
  ARE TO BE MULTIPLIED, ARE STORED IN THIS VECTOR.
  THEIR PLACES IN THE COEFFICIENTS VECTOR ARE THE EXPONENTS, I.E.
  THE CONSTANT IN CELL ZERO, AND THE COEFFICIENT WHICH IS ASSO-
  CIATED WITH THE HIGHEST (MAXIMUM) POWER, IS STORED IN THE CELL
  WITH INDEX MAXPOW.

  NSE IS THE NUMBER OF (LINEAR) SEGMENTS.

  LOVX IS THE LOWER VALUE OF X, I.E. THE LOWER LIMIT OF SEGMENT L
  PRESENTATION OF X.
  HIVX IS THE HIGHER VALUE OF X, I.E. THE HIGHER LIMIT OF
  SEGMENTAL PRESENTATION OF X.

  THE FIRST AND THE LAST SEGMENT ARE, HOWEVER,
  EXTENDED ON THE INDICATION OF THE VARIABLES MINUMX AND MAXUMX.

  THE FIRST SEGMENT OF X STRETCHES FROM MINUMX (THE MINIMUM
  VALUE OF X), TO LOVX +1/NSE * (HIVX-LOVX).
  SIMILARLY, THE LAST SEGMENT OF X STRETCHES FROM
  HIVX - 1/NSE * (HIVX-LOVX), TO MAXUMX.
```

```
OTHERWISE THE JTH SEGMENT OF APPROXIMATION COVERS THE INTERVAL
OF X BETWEEN LOVX + (J-1)/NSE * (HIVX-LOVX),
AND LOVX + J/NSE * (HIVX-LOVX).

EQN IS THE INDEX OF THE ROW OF T, WHERE THE EQUATION-RESTRICTION
ON THE TOTAL VALUE OF THE ZERO-ONE VARIABLES IS TO BE STORED.

RESTRN IS THE INDEX OF THE ROW OF T, WHERE THE REPRESENTATION
OF THE RESTRICTION ITSELF IS TO BE STORED, I.E.
RESTRN+I IS THE ROW-INDEX OF THE I TH SEGMENT.

IFZOV+J IS THE INDEX OF THE COLUMN OF T WHICH IS TO
REPRESENT THE JTH ZERO-ONE VARIABLE.

IX IS THE INDEX OF THE COLUMN OF T, WHICH IS TO REPRESENT X.
IY IS THE INDEX OF THE COLUMN OF T, WHICH IS TO REPRESENT Y.

IFDX+J IS THE INDEX OF THE JTH DUMMY-VARIABLE, REPRESENTING X.

IFDXDF+I IS THE INDEX OF THE ROW OF T,
WHERE THE ITH DEFINITIONAL RESTRICTION,
LINKING X TO ITS DUMMY REPRESENTATIVE
IN THE ITH (OR I-NSE)TH SEGMENT, IS TO BE STORED.
;

PUT FANCYHIGH FANCYHIGH:
FANCYHIGH:=99;

INITIATE MAXIMUM Y LOW:
MAXUMY:=-1000000;


WRITE EQUATION RESTRICTION ON THE INT VAR:
'FOR' J:=1 'STEP' 1 'UNTIL' NSE 'DO' T[EQN,IFZOV+J]:=-1;
T[EQN,N+1]:=-NSE+1;

PUT LIMITS ON X:
'IF' T[M+2,IX]=0 'OR' MAXUMX<T[M+2,IX]
'THEN' T[M+2,IX]:=MAXUMX;
'IF' MINUMX > T[M+3,IX] 'THEN' T[M+3,IX]:=MINUMX;

RELATE X TO SEGMENTED X AND DUMMIES:
'FOR' I:=1 'STEP' 1 'UNTIL' NSE 'DO' 'BEGIN'
  T[IFDXDF+I,IFZOV+I] := -FANCYHIGH;
  T[IFDXDF+I,IX] := 1;
  T[IFDXDF+I,IFDX+I] := -1;
  T[IFDXDF+NSE+I,IFZOV+I] := -FANCYHIGH;
  T[IFDXDF+NSE+I,IX] := -1;
  T[IFDXDF+NSE+I,IFDX+I] := 1;  'END';

PUT LIMITS ON X DUMMIES:
XDIF := (HIVX-LOVX)/NSE;
'FOR' I:=1 'STEP' 1 'UNTIL' NSE 'DO' 'BEGIN'
  T[M+2,IFDX+I] := LOVX+I*XDIF;
  T[M+3,IFDX+I] := LOVX+(I-1)*XDIF; 'END';
```

```
T[IFDXDF+NSE+NSE+1,N+1] := -MINUMX;
T[M+2,IFDX+NSE]:= MAXUMX;

RELATE Y TO X PER SEGMENT:

X:=LOVX;

'FOR' I:=1 'STEP' 1 'UNTIL' NSE 'DO' 'BEGIN'

  'IF' I>1 'THEN' LOWV:=HIGV 'ELSE' 'BEGIN'
    LOWV:=0; XP:=1;
    'FOR' P:=0 'STEP' 1 'UNTIL' MAXPOW 'DO' 'BEGIN'
      LOWV:=LOWV+COEFVEC[P]*XP;
      XP:=XP*X; 'END'; 'END';

  X:=X+XDIF;

  HIGV:=0; XP:=1;
  'FOR' P:=0 'STEP' 1 'UNTIL' MAXPOW 'DO' 'BEGIN'
    HIGV:=HIGV+COEFVEC[P]*XP;
    XP:=XP*X; 'END';

  DYDX := (HIGV-LOWV)/XDIF;

  T[RESTRN+I,IFDX+I]:=-DYDX;
  T[RESTRN+I,IY]:=ALPHA;
  T[RESTRN+I,IFZOV+I]:=-FANCYHIGH;
  T[RESTRN+I,N+1]:=HIGV-X*DYDX;

  PUT LIMIT ON Y:
  'IF' LOWV>MAXUMY 'THEN' MAXUMY:=LOWV;
  'IF' HIGV>MAXUMY 'THEN' MAXUMY:=HIGV; 'END';

SET UPPER LIMIT ON Y AS FOUND:
'IF' T[M+2,IY]=0 'OR' MAXUMY<T[M+2,IY]
'THEN' T[M+2,IY]:=MAXUMY;

PUT LIMITS ON ZERO ONE VAR:
'FOR' J:=1 'STEP' 1 'UNTIL' NSE 'DO' 'BEGIN'
  T[M+2,IFZOV+J]:=1.00000001; T[M+3,IFZOV+J]:=0; 'END';

REDUCE SIZE IF CONVEX OR NEAR EQUIVALENT:
II:=1; JJ:=2; NSEC:=NSE;

'FOR' SE:=2 'STEP' 1 'UNTIL' NSE 'DO' 'BEGIN'
  II:=II+1;

  HIGV:=T[M+2,IFDX+JJ]; LOWV:=-T[IFDXDF+2*NSEC+JJ-1,N+1];
  XDIF:=HIGV-LOWV;
  SLDIF:=(T[RESTRN+II-1,IFDX+JJ-1]-T[RESTRN+II,IFDX+JJ])*XDIF;

  'IF' SLDIF < 0.001 'THEN' 'BEGIN'

    'IF' SLDIF < -0.001 'THEN' 'GOTO' MERGE DUMMIES;

    MERGE SEGMENT:
    DYDX:=(-T[RESTRN+II,IFDX+JJ]*HIGV+T[RESTRN+II,N+1] +
    T[RESTRN+II-1,IFDX+JJ-1]*LOWV-T[RESTRN+II-1,N+1])/XDIF;
    T[RESTRN+II-1,IFDX+JJ-1]:=-DYDX;
    T[RESTRN+II-1,N+1]:=
    -T[RESTRN+II,IFDX+JJ]*HIGV+T[RESTRN+II,N+1]-DYDX*HIGV;
    'FOR' I:=RESTRN+II 'STEP' 1 'UNTIL' M+2 'DO'
    'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO' T[I-1,J]:=T[I,J];
```

```
    SE:=SE-1; NSE:=NSE-1; M:=M-1; IFDXDF:=IFDXDF-1;
    II:=II-1;

    MERGE DUMMIES:
    ADJUST NUMBER OF NON ZERO DUMMIES:
    T[EQN,N+1]:=T[EQN,N+1]+1;

    MERGE COLUMNS:
    'FOR' I:=RESTRN+II, IFDXDF+JJ, IFDXDF+NSEC+JJ,
    IFDXDF+2*NSEC+JJ, M+2 'DO' 'BEGIN'
      T[I,IFZOV+JJ-1]:=T[I,IFZOV+JJ];
      T[I,IFDX+JJ-1]:=T[I,IFDX+JJ]; 'END';


    SHIFT COLUMNS:
    'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' 'BEGIN'

      SHIFT FOR MERGED D:
      'FOR' J:=IFZOV+JJ+1 'STEP' 1 'UNTIL' N+1
      'DO' T[I,J-1]:=T[I,J];

      SHIFT FOR MERGED X:
      'FOR' J:=IFDX+JJ 'STEP' 1 'UNTIL' N
      'DO' T[I,J-1]:=T[I,J];
      'END';

    ADJUST COLUMN INDICES:
    N:=N-2; IFDX:=IFDX-1;
    'IF' IX > IFZOV 'THEN' IX:=IX-1;
    'IF' IY > IFZOV 'THEN' IY:=IY-1;
    'IF' IX > IFDX 'THEN' IX:=IX-1;
    'IF' IY > IFDX 'THEN' IY:=IY-1;

    SHIFT ROWS:
    'FOR' J:=1 'STEP' 1 'UNTIL' N+1 'DO' 'BEGIN'

      SHIFT UPW FOR MERGED UPPER XDF:
      'FOR' I:=IFDXDF+JJ 'STEP' 1 'UNTIL' M+3
      'DO' T[I-1,J]:=T[I,J];

      SHIFT UPW FOR MERGED LOWER XDF:
      'FOR' I:=IFDXDF+NSEC-1+JJ 'STEP' 1 'UNTIL' M+3
      'DO' T[I-1,J]:=T[I,J]; 'END';


    ADJUST ROW INDICES:
    M:=M-2; NSEC:=NSEC-1; 'END'
  'ELSE' JJ:=JJ+1; 'END';


MOVE EXTRA ZERO ONE VAR:
'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' 'BEGIN'
  COP:=T[I,IFZOV+NSEC];
  'FOR' J:=IFZOV+NSEC 'STEP' 1 'UNTIL' N-RN 'DO'
  T[I,J]:=T[I,J+1];
  T[I,N-RN+1]:=COP; 'END';
```

```
END OF POLY:

REINTERPRET TO LOWER LIMIT DISTANCES:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  T[M+2,J] := T[M+2,J]-T[M+3,J];
  'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
  T[I,N+1] := T[I,N+1]-T[M+3,J]*T[I,J]; 'END';

FINAL END OF POLY:  'END';
```

CHAPTER XX

BRANCHING METHODS

20.1 Branching in the order of increasing indices

The algorithm offered in this section is more primitive than the ones normally recommended in the literature.

It is specifically concerned with zero-one variables. It serves the purpose of introducing the idea of branching and it has the further attraction of a relatively simple programme-code.

It solves the (zero one variables) mixed integer programming problem by means of a series of ordinary linear programming problems. In this particular method, these linear programming problems refer to tableaux from which columns which relate to zero-one variables are progressively removed.

A branch is a consecutively solved series of "ordinary" L.P. problems which arise from the introduction of additional requirements, requiring successive variables to attain specified integer values. If the variable is zero we remove the column altogether, if the variable is one, we subtract it from the value column. It seems natural to assume that a branch is developed in the order of the indices of the variables. Thus we would begin with specifying a value for the integer variable with the lowest index. In the example from section 19.3 we could begin with requiring that d_1 should be zero. In that example, there is an equation requirement $d_1 + d_2 + d_3 + d_4 = 3$. The requirement that d_1 should be zero, combined with upper limits on d_2, d_3 and d_4, therefore automatically implies $d_2 = d_3 = d_4 = 1$ This is not necessarily so in the general mixed integer programming problem, and one may have to impose further restrictions on other integer-restricted variables (e.g. d_2, d_3, etc), until an admissable solution to the specified integer programming problem is reached, or until a particular branch is terminated (abandoned) for other reasons. We will obviously need some sort of system to ensure that all appropriate combinations are investigated. We do not start with requiring that all integer-restricted variables actually attain an integer value.

The two main reasons for this practice are a) It is systematic rather than accidental that additional integer requirements are sometimes met without imposing a requirement to that effect on all variables in question. b) Non-integer solutions may reveal the fact that there is no prospect of finding the optimum in a particular branch.

We will come back to this problem, when we have discussed the coding system. There is no inherent reason why problems should be coded in a particular way and no other, provided some basic desiderata are met. Code-numbers (declared as integer variables) should enable us to identify integer restrictions in a particular problem. They should also enable us to relate the code-number to the order in which the various problems are tackled. The formula

$$\text{Code} = \sum_{k=1}^{niv} (1 + rv_k) \quad 3^{niv-k} \qquad (20.1.1)$$

meets these desiderata, and will be used here.

In (20.1.1) rv is the restriction value, i.e. $rv_k = 0$ indicates a restriction which says that the k^{th} zero-one variable is required to be zero, and $rv_k=1$ indicates that the k^{th} zero-one variable is required to be one. It is assumed that variables to which no integer restriction is actually applied are not included in the summation, i.e. the effective value of rv_k is -1 in that case, $1+rv_k$ attaining the value zero.

That this coding formula meets the two basic requirements of a coding system is illustrated below by tabulating its effective significance for the case of three zero-one variables.

The various LP problems are in fact tackled in the order of increasing numerical value of the code-number (see tabulation 20.1).

In the case of three zero-one variables we may (at most) have to investigate 15 LP problems. There is however, a possibility that certain branches may not need to be investigated, at least not beyond the problem at the head of a particular branch.

In fact, the problem "at the head" defines the branch, by a particular series of requirements.

$$d_k = 0,1 \quad (k = 1, 2, \ldots \text{index}, \text{index} \leq niv) \qquad (20.1.2)$$

In (20.1.2), niv is the number of integer restricted variables, i.e. 3 in the tabulated example.

Note that we do not investigate problems where the integer restrictions on variables with higher indices are implemented, while the similar restrictions on variables with lower indices are not implemented. (Problems 1-8 , 10, 11, 19 and 20).

A branch, or the search for a solution of the problem as a whole

Tabulation 20.1

Coding summary for 3 zero-one variables

Inv. order	R. on d_1	R. on d_2	R. on d_3	Code rv_1 as	Code rv_2 as	Code rv_3 as	Value of Code		
1	?	?	?	-	-	-	0		
Never	?	?	0	-	-	$1\times3^0=1$	1		
Never	?	?	1	-	-	$2\times3^0=2$	2		
Never	?	0	?	-	$1\times3=3$	-	3		
Never	?	0	0	-	$1\times3=3$	$1\times3^0=1$	4		
Never	?	0	1	-	$1\times3=3$	$2\times3^0=1$	5		
Never	?	1	?	-	$2\times3=6$	-	6		
Never	?	1	0	-	$2\times3=6$	$1\times3^0=1$	7		T
Never	?	1	1	-	$2\times3=6$	$2\times3^0=2$	8		
2	0	?	?	$1\times3^2=9$	-	-	9		
Never	0	?	0	$1\times3^2=9$	-	$1\times3^0=1$	10		
Never	0	?	1	$1\times3^2=9$	-	$2\times3^0=2$	11		
3	0	0	?	$1\times3^2=9$	$1\times3=3$	-	12	$d_1=d_2=0$ Branch	
4	0	0	0	$1\times3^2=9$	$1\times3=3$	$1\times3^0=1$	13		
5	0	0	1	$1\times3^2=9$	$1\times3=3$	$2\times3^0=2$	14		
6	0	1	?	$1\times3^2=9$	$2\times3=6$	-	15	$d_1=0$ $d_2=1$ Branch	
7	0	1	0	$1\times3^2=9$	$2\times3=6$	$1\times3^0=1$	16		
8	0	1	1	$1\times3^2=9$	$2\times3=6$	$2\times3^0=2$	17		
9	1	?	?	$2\times3^2=18$	-	-	18		T
Never	1	?	0	$2\times3^2=18$	-	$1\times3^0=1$	19		
Never	1	?	1	$2\times3^2=18$	-	$2\times3^0=2$	20		
10	1	0	?	$2\times3^2=18$	$1\times3=3$	-	21	$d_1=1$ $d_2=0$ Branch	
11	1	0	0	$2\times3^2=18$	$1\times3=3$	$1\times3^0=1$	22		
12	1	0	1	$2\times3^2=18$	$1\times3=3$	$2\times3^0=2$	23		
13	1	1	?	$2\times3^2=18$	$2\times3=6$	-	24	$d_1=d_2=1$ Branch	
14	1	1	0	$2\times3^2=18$	$2\times3=6$	$1\times3^0=1$	25		
15	1	1	1	$2\times3^2=18$	$2\times3=6$	$2\times3^0=2$	26		

ends, when one of the following four conditions is met:

a) Full Exhaustion

All relevant problems have been solved, i.e. integer restrictions and all possible combinations of zeros and ones) have been imposed on all variables.

b) Optimality without Exhaustion

A (sub)problem in which integer restrictions have not so far been imposed on all variables (only on those below a certain index) happens to have an optimal solution which satisfies all the remaining integer restrictions as well. Thus, in the example from section 19.3, there are dummy-variables which are subject to an integer restriction of zero-one type. The requirement $d_1 = 0$, combined with $d_1 + d_2 + d_3 + d_4 = 3$ and the upper limits of $d_2 \leq 1$, etc., implies $d_2 = d_3 = d_4 = 1$. Obviously it is then unnecessary to investigate the combination $d_1 = 0$, $d_2 = 0$. Finding an all-integer solution as optimum of the problem at the head of a branch terminates the branch.

c) An empty problem is met

If the problem at the head of a branch is empty, the whole branch is abandoned.

d) An Unbounded problem is met

This is in fact only possible with the initial continuous problem. When the continuous problem has been solved, i.e. is found to have a finite optimal solution, subsequent branching may still lead to empty problems, but not to unbounded problems.

e) Sub-optimality

The optimal solution of the problem at the head of the branch (irrespective of further integer requirements), indicates a lower solution value than a previously found integer solution.

Conditions b), c), d) and e) all cause a reduction in the number of problems to be solved, they are premature terminations.

The most extreme form of premature termination is obviously to find the continuous problem empty, unbounded or all-integer in its optimum.

Similarly a branch is terminated, as soon as the problem at the head of the branch is either found to be wanting in some

respect, i.e. empty or sub-optimal, or to have an all-integer optimum. Recall the tabulation of the three integer restricted variables case:

If problem 18 ($d_1 = 1$, d_2 and d_3 so far not integer restricted) is empty, problems 21 to 26, whose solutions would be feasible solutions of problem 18, need no investigation.

Likewise, if we already found a feasible integer solution, there is a possibility that one or more remaining branches may be found sub-optimal by way of investigating the problems at their heads. Suppose for example that we already recorded an optimal and feasible solution of the $d_1 = 0$ branch (for example) a solution value of 53 for problem 14. ($d_1 = d_2 = 0$, $d_3 = 1$).

We will need to investigate the rest of the $d_1 = 1$ branch, (problem 21 to 26), only if problem 18 is found to have an optimal and feasible solution, with a solution value of more than 53. But even then we may find the sub-branch $d_1 = 1$, $d_2 = 0$ sub-optimal. If problem 21 ($d_1 = 1$, $d_2 = 0$, d_3 not so far integer restricted), has a solution value of 49, we know that the optimum is not in the $d_1 = 1$, $d_2 = 0$ branch. All feasible solutions of problems 22 and 23, including the optima, are also feasible solutions of problem 21, hence their optimum solution values cannot exceed the optimum of problem 21. The other way in which the number of actually investigated problems may be reduced, arises in connection with non-required part-integer solutions.

For reasons which tend to be somewhat more complicated* than in the all-integer case - but not basically different -, this possibility is to some extent systematic rather than incidental. We do not need to record all partly integer solutions. Thus, the optimum of problem 0, may turn out to satisfy the requirements of problem 5. (d_1 and d_2 are not integer-restricted, $d_3 = 1$), for example $d_1 = 0.22$, $d_2 = 0.73$, $d_3 = 1$. Because we do not intend to investigate problem 5 in any case, we may as well classify the solution of problem zero as being non-integer as soon as d_1 is found to have a non-integer value. But if the optimum of problem zero turns out to be $d_1 = 1$, $d_2 = 0$, $d_3 = 0.43$, we may record the fact that problems 18 and 21 have already been solved. It is in fact sufficient to record the attainment of an optimum solution to problem 21.

---

* See Weingartner [38]

A prematurely obtained solution of problem 21 implies that it also is the optimum of problem 18. We do not tackle any sub-problems of the $d_1 = 1$ branch before we enter it at the head. Thus we will record the code 21 and a corresponding solution value in a solutions list.

When we come to tackle problem 18, we can identify 21 as being a sub-problem of the $d_1 = 1$ branch.

The relevant numerical condition of the code is

$$0 \leq \text{recorded code} - \text{code} < 3^{\text{niv-index}} \qquad (20.1.3)$$

Above, recorded code (obviously) is the code of the previously recorded solution, and code is the code of the problem which is currently being tackled.

If the previously recorded code is less than the code of the currently tackled problem, the previously recorded code refers to a problem which belongs to an already exhausted branch. (Problem 17 does not belong to the $d_1 = 1$ branch). The problem at the head of the branch has the lowest code of all the problems in the branch. Since codes of lower values than the code of the currently tackled one, are codes of problems already dealt with the programme developed in association with this section, will wipe out such codes, whenever they are met, just to save list-space. On the other hand, problems with a code which is at least $3^{\text{niv-index}}$ greater than the problem which is being tackled, belong to the next branch, or possible to one which is still further ahead.

For example, we may find the optimal solution of the $d_1 = 1$ branch (whether obtained by fresh calculation or from earlier recording) to be an improvement relative to anything found in the $d_1 = 0$ branch. We therefore enter the $d_1 = 1$ branch further, and tackle problem 21. The index up to which the variables are integer-restricted is then 2, niv-index is 3-2=1, and (20.1.3) reveals that problems whose codes are at least $3^1=3$ higher than 21 (i.e. 24, 25, 26), do not belong to the $d_1 = 1$, $d_2 = 0$ branch.

The computational implementation of the branching algorithm depends to some extent on whether one is interested in the shadow prices of the integer restrictions. In this section it is assumed that the user is not interested in those shadow prices. Each sub-problem which is tackled, is started by partly copying the initial tableau of the continuous problem. Columns which refer to variables on which integer restrictions are actually imposed are not copied. If the integer value is zero, the corresponding column is simply suppressed, if the integer value is one, the corresponding column is subtracted

from the value column, i.e. the variable is brought over to the right-hand side.

The integer programming procedure is now listed as follows:

TEXT-LISTING OF THE ZERO-ONE VARIABLES INTEGER PROGRAMMING PROCEDURE.

```
'PROCEDURE' ZOLP(INTA,OUTA,OUTM,OUTN,T,M,N,NEQ,NAV,NZOV,
OUTROWLST,ROWLST,COLLST,EXITTYPE,NSL);
'INTEGER' OUTM,OUTN,M,N,NEQ,NAV,NZOV,EXITTYPE,NSL;
'ARRAY' INTA,OUTA,T;
'INTEGER' 'ARRAY' OUTROWLST,ROWLST,COLLST;

'BEGIN' 'INTEGER' I,J,P,NEWCODE,NAME,II,JJ,R,K,ROWN,COLN;
  'REAL' SOLUTION VALUE,NUM,QUO,ZERODIS,ONEDIS;
  'BOOLEAN' INTG;
  'ARRAY' SOLUTION VALUES LIST[1:NSL];
  'INTEGER' 'ARRAY' CODELIST[1:NSL],D[NAV+1:NAV+NZOV];

  'PROCEDURE' LINP(T,M,N,NEQ,NAV,ROWLST,COLLST,REENTRY);
  'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,REENTRY;
  'INTEGER' 'ARRAY' ROWLST,COLLST;
  'ALGOL';

  'PROCEDURE' INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,
  IROWLST,R,K,ROWN,COLN,REENTRY);
  'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,R,K,ROWN,COLN,REENTRY;
  'INTEGER' 'ARRAY' ROWLST,COLLST,IROWLST;
  'ALGOL';

  'PROCEDURE' BRANCH(CODE,RE VALUE,INDEX,
  TO BE SOLVED FROM NEW);
  'VALUE' CODE,RE VALUE,INDEX,TO BE SOLVED FROM NEW;
  'INTEGER' CODE,RE VALUE,INDEX;
  'BOOLEAN' TO BE SOLVED FROM NEW;
  'BEGIN'

    CHECK IF INVESTIGATED BEFORE:
    'FOR' P:=1 'STEP' 1 'UNTIL' NSL 'DO' 'BEGIN'
      'IF' CODELIST[P] < CODE 'THEN' CODELIST[P]:=-10;
      'IF' CODELIST[P]-CODE < 3*(NZOV-INDEX) -0.5
      'AND' 'NOT' SOLUTION VALUES LIST[P] > SOLUTION VALUE
      'THEN'  'BEGIN' CODELIST[P]:=0;
        'GOTO' END OF BRANCH; 'END'; 'END';

    'IF' 'NOT' TO BE SOLVED FROM NEW
    'THEN' 'GOTO' RESTRICT ADDITIONALLY;

    PREPARE AN LP PROBLEM:

    PUT INITIAL RHS:
    'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
    T[I,N+1-INDEX]:=INTA[I,N+1];

    PUT HIGH UPPERLIMITDISTANCES FOR SLACKS:
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
    T[I,N-INDEX+2] := 1000000000000;
```

```
ATTEND TO REST OF TABLEAU:
JJ:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  'IF' J > NAV 'AND' 'NOT' J > NAV+INDEX 'THEN' 'BEGIN'
    'IF' D[J]=1 'THEN' 'GOTO' BRING TO RHS
    'ELSE' 'GOTO' NEXT COLUMN; 'END';

  COPY JTH COLUMN:
  JJ:=JJ+1;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+2 'DO'
  T[I,JJ]:=INTA[I,J];
  'GOTO' NEXT COLUMN;

  BRING TO RHS:
  'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
  T[I,N+1-INDEX]:=T[I,N+1-INDEX]-T[I,J];

  NEXT COLUMN:
  'END';

EXITTYPE:=0;
LINP(T,M,N-INDEX,NEQ,NAV,ROWLST,COLLST,EXITTYPE);
TO BE SOLVED FROM NEW := 'FALSE';

'GOTO' CHECK FOR NO SOLUTION OF PARTICULAR PROBLEM;

RESTRICT ADDITIONALLY:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[I]=NAV+1 'THEN' 'GOTO' PUT THE CUT;

CHECK AND ATTEND NON BASIC VAR:
'FOR' K:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
'IF' COLLST[K]=NAV-INDEX 'THEN' 'BEGIN'
  'IF' D[NAV+INDEX]=0 'THEN' 'GOTO' REDUCE
  'ELSE' 'GOTO' PREPARE AN LP PROBLEM; 'END';

CHECK AND ATTEND BINDING UPPER LIMIT:
'FOR' K:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
'IF' COLLST[K]=10000+NAV+INDEX 'THEN' 'BEGIN'
  'IF' D[NAV+INDEX]=1 'THEN' 'GOTO' REDUCE
  'ELSE' 'GOTO' PREPARE AN LP PROBLEM; 'END';

PUT THE CUT:
'IF' D[NAV+INDEX]=0 'THEN'
T[I,N-INDEX+3] := 0.0000001-T[I,N-INDEX+2]
'ELSE' T[I,N-INDEX+2] := T[I,N-INDEX+2]-0.999999;

PUT FANCYHIGH UPPER LIMITS ON SLACKS:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[I] > 1000 'THEN' T[I,N+2] := 1000000000000;
'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
'IF' COLLST[J] > 1000 'AND' COLLST[J] < 1001 + M
'THEN' T[M+2,J] := 1000000000000;
```

```
CALL INTP;
EXITTYPE:=1; R:=K:=ROWN:=COLN:=0;
INTP(T,M,N-INDEX+1,NEQ,NAV,ROWLST,COLLST,ROWLST,
R,K,ROWN,COLN,EXITTYPE);
'IF' EXITTYPE = -10 'THEN' 'GOTO' CALL INTP;
'IF' 'NOT' EXITTYPE=0 'THEN' 'GOTO' END OF BRANCH;

'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO
'IF' (COLLST[J]<1000 'OR' COLLST[J]>NEQ+1000)
'AND' T[M+1,J]<0 'THEN' 'GOTO' CALL INTP;

'FOR' K:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
'IF' COLLST[K]=NAV+INDEX 'OR' COLLST[K]=10000+NAV+INDEX
'THEN' 'GOTO' REDUCE;

REDUCE:

REMOVE KTH COLUMN:
'FOR' I:=1 'STEP' 1 'UNTIL' M+2 'DO'
'FOR' J:=K+1 'STEP' 1 'UNTIL' N-INDEX+3 'DO'
T[I,J-1]:=T[I,J];

REMOVE KTH NAME FROM LIST OF COLUMNAMES:
'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
'IF' COLLST[J] > NAV 'AND' 'NOT'
(COLLST[J] > 1000 'AND' COLLST[J] < 1001 + M)
'THEN' COLLST[J] := COLLST[J] - 1;
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[I] < N 'THEN' ROWLST[I]:=ROWLST[I]-1;
'FOR' J:=K+1 'STEP' 1 'UNTIL' N-INDEX+1 'DO'
COLLST[J-1]:=COLLST[J];

CHECK FOR NO SOLUTION OF PARTICULAR PROBLEM:
'IF' EXITTYPE#0 'THEN'
'GOTO' END OF BRANCH;

CHECK WHETHER WORTHWHILE TO PERSUE:
'IF' 'NOT' T[M+1,N+1-INDEX] > SOLUTION VALUE
'THEN' 'GOTO' END OF BRANCH;

CHECK IF ADDITIONAL INTEGER RESTRICTIONS ARE MET:
INTG:='TRUE';

'FOR' P:=INDEX+1 'STEP' 1 'UNTIL' NZOV 'DO' 'BEGIN'
  NUM:=0;
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[I]=NAV+P-INDEX 'THEN' NUM:=T[I,N+1-INDEX];
  'FOR' J:=1 'STEP' 1 'UNTIL' N-NZOV-INDEX 'DO'
  'IF' COLLST[J]=NAV+P-INDEX+10000 'THEN'
  NUM:=1;
  'IF' NUM < 0.000001 'THEN' NUM:=0;
  'IF' NUM > 0.99999  'THEN' NUM:=1;
  'IF' 'NOT' (NUM=0 'OR' NUM=1) 'THEN' INTG:='FALSE';
  D[NAV+P]:=NUM;
  'END';
```

```
'IF' 'NOT' INTG 'THEN' 'GOTO' MARK NON INTEGER SOL;

MARK NEW IMPROVED INTEGER SOLUTION:
SOLUTION VALUE:=T[M+1,N+1-INDEX];

PUT BRANCHED ZO VAR:
'FOR' I:=1 'STEP' 1 'UNTIL' INDEX 'DO'  'BEGIN'
  OUTROWLST[I]:=NAV+I;
  OUTA[I,1]:=D[NAV+I]; 'END';

INITIATE EXIT TABLEAU PARAMETERS:
OUT N := N-INDEX;  II:=INDEX;

PUT OTHER ZO VAR:
'FOR' NAME:=NAV+INDEX+1 'STEP' 1 'UNTIL' NAV+NZOV 'DO'
'BEGIN'
  II:=II+1;
  OUTROWLST[II]:=NAME;
  'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[I]=NAME-INDEX
  'THEN' OUTA[II,1]:=T[I,N-INDEX+1];
  'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX 'DO'
  'IF' COLLST[J]=NAME 'THEN' OUTA[II,1]:=0;
  'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX 'DO'
  'IF' COLLST[J]=10000+NAME -INDEX
  'THEN' OUTA[II,1]:=INTA[M+2,NAME];
  'END';

PUT VAR OF TYPE ABS:
'FOR' NAME:=1 'STEP' 1 'UNTIL' NAV 'DO' 'BEGIN'
  OUTROWLST[NZOV+NAME] := NAME;
  OUTA[NZOV+NAME,1]:=T[NAME,N+1-INDEX]; 'END';

II:=NZOV+NAV;

PUT ORDINARY VARIABLES WITH COLUMN NAMES:
'FOR' NAME:=NZOV+NAV+1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'

  'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[I]=NAME-INDEX 'THEN' 'BEGIN'
    II:=II+1;
    OUTROWLST[II]:=NAME;
    OUTA[II,1]:=T[I,N+1-INDEX]+INTA[M+3,NAME]; 'END';
  'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX 'DO'
  'IF' COLLST[J]=NAME+10000-INDEX 'THEN' 'BEGIN'
    II:=II+1;
    OUTROWLST[II]:=NAME;
    OUTA[II,1]:=INTA[M+2,NAME]+INTA[M+3,NAME]; 'END';
  'END';

PUT SLACKS:
'FOR' NAME:=1000+1 'STEP' 1 'UNTIL' 1000+M 'DO' 'BEGIN'
  'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
  'IF' ROWLST[I]=NAME 'THEN' 'BEGIN'
    II:=II+1;
    OUTROWLST[II]:=NAME;
    OUTA[II,1]:=T[I,N+1-INDEX]; 'END'; 'END';
```

```
OUT M := II;

'GOTO' END OF BRANCH;


  MARK NON INTEGER SOL:
  NEWCODE:=CODE;
  'FOR' P:=INDEX+1 'STEP' 1 'UNTIL' NZOV 'DO'
  NEWCODE:=NEWCODE+(1+D[NAV+P])*3↑(INDEX-P);
  'FOR' J:=1 'STEP' 1 'UNTIL' NSL 'DO' 'BEGIN'
    'IF' CODELIST[J]=0 'THEN' CODELIST[P]:=NEWCODE;
    'GOTO' BRANCH AGAIN; 'END';

  WARNING FOR EXHAUSTING SOLUTIONLIST:
  'IF' P > NSL 'THEN' 'BEGIN'
    NEWLINE(1);
    WRITETEXT('('SOLUTIONS%LIST%EXHAUSTED')'); 'END';

  BRANCH AGAIN:

    'IF' INDEX=NZOV 'THEN' 'GOTO' END OF BRANCH;

  'BEGIN' 'REAL' SAVED ONE;

    CALCULATE MAXIMUM ONEBRANCH:
    ONEDIS:=1-T[NAV+1,N-INDEX+1];
    QUO:=10000000000;
    'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX 'DO'
    'IF' 'NOT' (COLLST[J]>1000 'AND' COLLST[J] < 1000+NEQ+1)
    'AND' T[NAV+1,J] < 0 'THEN' 'BEGIN'
      'IF' -T[M+1,J]/T[NAV+1,J] < QUO 'THEN'
      QUO:=-T[M+1,J]/T[NAV+1,J]; 'END';
    SAVED ONE := T[M+1,N-INDEX+1] - QUO*ONEDIS;

    CALCULATE MAXIMUM ZEROBRANCH:
    ZERODIS:=T[NAV+1,N+1];
    QUO:=10000000000;
    'FOR' J:=1 'STEP' 1 'UNTIL' N-INDEX 'DO'
    'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1000+NEQ+1)
    'AND' T[NAV+1,J] > 0 'THEN' 'BEGIN'
      'IF' T[M+1,J]/T[NAV+1,J] < QUO 'THEN'
      QUO:=T[M+1,J]/T[NAV+1,J]; 'END';

    ZEROBRANCH:
    RE VALUE := 0;
    'IF' 'NOT' T[M+1,N-INDEX+1] - QUO*ZERODIS >
    SOLUTION VALUE 'THEN' 'GOTO' ONEBRANCH;

    FORK:
    D[NAV+INDEX+1]:=RE VALUE;
    NEWCODE:=CODE;
    NEWCODE:=NEWCODE+(1+RE VALUE)*3↑(NZOV-INDEX);
    BRANCH(NEWCODE,RE VALUE,INDEX+1,TO BE SOLVED FROM NEW);
    TO BE SOLVED FROM NEW := 'TRUE';
```

```
    ONEBRANCH:
    'IF' RE VALUE=0 'THEN' 'BEGIN'
      'IF' 'NOT' SAVED ONE > SOLUTION VALUE
      'THEN' 'GOTO' END OF BRANCH;
      RE VALUE := 1; 'GOTO' FORK; 'END'; 'END';


  END OF BRANCH: 'END';



PARTLY FILL EXIT NAMELIST:
'FOR' I:=1 'STEP' 1 'UNTIL' NZOV 'DO' OUTROWLST[I]:=NAV+I;
'FOR' I:=1 'STEP' 1 'UNTIL' NAV 'DO' OUTROWLST[NZOV+I]:=I;

FILL CODELIST AND SOL LIST WITH DUMMIES:
'FOR' P:=1 'STEP' 1 'UNTIL' NSL 'DO' 'BEGIN'
  SOLUTION VALUES LIST[P]:=0; CODELIST[P]:=-10; 'END';

INITIATE SOLUTION VALUE LOW:
SOLUTION VALUE := -1000000000;

  INITIATE TABLEAU ORDER:
  OUT M := M;  OUT N := N-NZOV;

BRANCH(0,0,0,'TRUE');

REGISTER WHETHER SOLVED:
'IF' SOLUTION VALUE > -99999999 'THEN' EXITTYPE:=0
'ELSE' EXITTYPE:=-1;

END OF ZOLP: 'END';
```

20.2 Branching in the general mixed integer problem

The general mixed integer problem is here defined as follows

Maximise $\tau = \underline{w}' \underline{x}$

Subject to $\underline{A}\underline{x} \leq \underline{b}$ (7.2.2)

$$l_j \leq x_j \leq h_j \quad (j = n_1, n_1 + 1, \ldots, n) \tag{20.2.1}$$

and

$$\left.\begin{array}{l} x_j = l_j + 1, l_j + 2 \ldots\ldots h_j \\ (j = n_1 + 1, n_1 + 2 \ldots\ldots, n_1 + \text{nirv}) \end{array}\right) \tag{20.2.2}$$

Here n is the total number of elements of $\underline{x}$, n_1 is the number of "free" variables which are not subject to any lower or upper bound, an nirv is the number of integer-restricted variables.

It is more or less conventional to restrict integer-restricted variables to non-negative valued integers ($x_j = 0, 1, 2 \ldots.$), but this convention has no meaning in connection with the algorithm offered here, because the upper and lower limit facilities of sections 10.3 and 10.4 will be employed in any case. Note, however, that while upper limits may be put at some fancyhigh figure if no meaningful upper limit is intended, one should try to supply a realistic lower limit, for the reasons explained in section 10.4.

The notion of the zero-branch and the one-branch, as discussed in section 20.1 may now be replaced by that of the lower branch and the higher branch.

Branching consists of adding additional restrictions to the problem, the branching restrictions, or to be more precise in redefining upper and lower limits. We may branch by inequalities or by equations. If we branch by means of inequalities, we branch (split) the problem for some variable x_j into two rump-problems.

In the x_j lower branch problem, x_j is restricted to

$$x_j \leq \text{int}\,(x_j^*) \tag{20.2.3}$$

where x_j^* is the current (fractional) value of x_j, $\text{int}(x_j^*)$ is the next lower integer number.

The x_j higher branch problem, then contains the branching restriction

$$x_j \geq \text{int}(x_j^*) + 1 \qquad (20.2.4)$$

As we may find need to subsequently investigate different integer values of x_j, we also refer to the problems restricted by (20.2.3) or (20.2.4), as x_j higher main branch and the x_j lower main branch problems.

There is no general rule which says that either (20.2.3) or (20.2.4) is binding on the integer optimum.

Example

Maximise $\tau = 9\,x_1 - 3\,x_2 + 2\,x_3$

Subject to $3\,x_2 \geq 7\,x_1$, $10\,x_1 + 11\,x_3 \leq 25$, $0 \leq x_1 \leq 100$,

$0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$,

x_1 and x_2 integer-restricted, x_3 continuous.

The continuous optimum of this problem is $x_1 = 2\frac{1}{2}$, $x_2 = 5\ 5/6$, $x_3 = 0$, with a solution value of $\tau = 5$.

The restriction $3\,x_2 \geq 7x_1$ is binding in the continuous optimum, and it effectively reduces the amount of solution value to be obtained from a unit of x_1, from 9 to 2.

The integer requirements on x_1 and x_2 imply however, more than 3 units of x_2 per 7 units of x_2, except for exact multiples of $x_1 = 3$, $x_2 = 7$. The result is that the continuous variable x_3 becomes a more economical way of utilizing the 25 units of b_2, and the integer optimum is $x_1 = x_2 = 0$, $x_3 = 2\ 1/11$, with $\tau = 4\ 2/11$.

A point in the algorithm where a previously existing LP problem is split into one or more sub-problems which differ only in the branching restriction on one variable, is called a node.

There is no uniform convention with respect to equations or inequalities. Land and Doig [26] are generally credited with

having pioneered the branching method, and they use equations. Hence in our example, branching on x_2 would give rise to the $x_2 = 0$, $x_2 = 1$, and $x_2 = 2$ branches $\frac{2}{=}$ thereafter, for $x_2 > 2\frac{1}{2}$, the problem becomes infeasible. Dakin [7] on the other hand uses inequalities.

Partly for this reason the term "branch and bound method" which is otherwise usual is not used here. There is an element of ambiguity in this term because the word bound could refer either to a bound on the objective function or to a bound on a variable; we will simply speak of a branching method.

A major problem in all branching algorithms is how to keep record of the deleniation between those parts of the continuous feasible space area that have already been explored, and the areas that still need investigating and which may conceivably contain the optimum.

Dakin solves this record-keeping problem by keeping a list of so-far unexplored problems. Every time a higher branch problem is chosen, the corresponding lower branch problem is added to the list of unexplored problems and vice versa. Land and Doig on the other hand, solve sub-problems strictly in order of declining solution value; as a result of this procedure their algorithm tends to involve a substantial number of sub-problems, which are in the process of further branching at the same time.

The algorithm which is offered here, solves the record-keeping problem by postulating a re-ordering of the variables, using just two sub-problems per variable at the same time. It is agreed by all those who have tried their hands on integer programming that it is computationally efficient to branch on a variable for which the integer restriction is in some sense more binding than for other variables.

The criterion opted for here is to branch on the variable which is furthest away from any integer value, that criterion also appears to be the one used in most other algorithms. The ordering system is as follows:

The number of variables, which are already subject, or being subjected to branching restrictions, is called the node-index,

or the node level. The variables to which branching restrictions apply, are then called the variables at or below the node index. The variables at or below the node index are considered as ordered according to the time of branching. The variables at or below the node index may further be distinguished as follows:

The variable on which branching took place in the continuous problem before there were any branching restrictions on other variables is indicated as the leading variable, the variable on which branching is now taking place or has just taken place is indicated as the variable at the node index.

The variables on which branching took place previously and on which branching restrictions are still in force in the current problem are together indicated as the variables below the node-index. There is no ordering of the variables to which no branching restrictions apply, and these (unordered) integer-restricted variables on which no branching restrictions apply in the current problem are called the variables above the node-index.

The ordering of the variables below tne node-index permits us to develop additional sub-problems with branching restrictions which differ only in the constant, in a systematic way.

Suppose, for example, that we are dealing with a problem in which there are 4 integer restricted variables. We will indicate them as x_1, x_2, x_3 and x_4, but it should be borne in mind that the first integer restricted variable may be preceeded by any number of free variables and that the last integer restricted variable may be followed by any number of bounded continuous variables. (We impose the convention that free variables come before integer-restricted variables, and that bounded continuous variables come after integer restricted variables.)

Variable	lower branch	higher branch	current branch
x_3	3	7	higher
x_2	5	exhausted	lower
x_4	unexplored	4	higher
x_1	unexplored	unexplored	?

node-index = 3.

This record sheet gives us the following information:

The variable on which branching took place for the first time, was x_3, i.e. x_3 is the leading variable. Sub-problems which contained the integer requirements $x_3 = 4$, $x_3 = 5$, and $x_3 = 6$, have already been fully explored; if there was an indication that the integer optimum could be characterized by $x_3 = 4$, $x_3 = 5$, or $x_3 = 6$, a best so far attained integer solution with $4 \leq x_3 \leq 6$ is already noted separately, to be available for output if it still is the best solution at the end of the algorithm. The $x_3 = 3$ sub-problem has been found feasible (otherwise the x_3 lower branch would be indicated as being exhausted), but this sub problem has not so far been explored further, because the solution value of the $x_3 = 7$ sub problem was found to be higher. The problem which is currently being investigated, contains the branching restrictions $x_3 = 7$, $x_2 = 5$, $x_4 = 4$, while no branching restriction applies as yet to x_1.

The $x_3 = 7$, $x_2 = 5$, $x_4 = 4$ sub problem has been developed, by treating the $x_3 = 7$ sub-problem as an integer programming problem in its own right, with x_2, x_4 and x_1 being the inter-restricted variables. We distinguish branching in a sub problem and thereby developing branching restrictions on more variables, from branching on the leading variable by using the term <u>further branching</u>. All branching takes place on the variable which is furthest away from any integer value. If therefore follows that x_2 was the variable furthest away from any integer value in the continuous optimum of the $x_3 = 7$ sub problem, and x_4 was found as the variable furthest away from any integer value in the continuous optimum of the $x_3 = 7$ $x_2 = 5$ sub problem.

The value of x_4 in the optimum solution of the $x_3 = 7$, $x_2 = 5$, sub-problem must have been in the interval $3 < x_4 < 4$, otherwise the $x_3 = 7$, $x_2 = 5$, $x_4 = 4$ sub problem would not be the first problem to be developed when branching on x_4. The variables x_3, x_2 and x_4 (ordered in that way) are the variables at or below the node-index, x_1 is the one variable above the node-index; x_3 and x_2 are the variables below the node-index, x_4 is the variable at the node-index.

To state the algorithm somewhat more formally, it may be useful to introduce some additional notation and terminology.

$x_j(k)$ is the j^{th} element of $\underline{x}$, on which branching took place at node level k.

The sub-problem which contains the branching restriction

$$x_{j(k)} = \text{int}\ (x^*_{j(k)} + f + 1) \qquad (20.2.5)$$

(and no branching restrictions above node level k) is called the f^{th} further out sub-branch problem on the $x_{j(k)}$ higher branch.

The similar problem for f = 0, is called the opening problem of the $x_{j(k)}$ higher main branch. Similarly, for the lower branch, the opening problem contains the equation-equivalent of (20.2.3), and the f^{th} further out problem on the lower branch contains the restriction

$$x_{j(k)} = \text{int}(x^*_{j(k)} - f) \qquad (20.2.6)$$

A sub branch problem which differs from a previously developed one on the same main branch, only by the fact that f has been increased by one, is called the next further out problem on the main branch in question.

The exploration of both main branch problems and of sub branch problems takes place under the side condition that the value of the objective function must exceed a previously set figure. That figure is initially $-\infty$ (in computational implementation: -10^{24}) but as feasible integer solutions are found the required minimum is increased each time an improved solution is identified.

A sub branch problem which, though possessing an optimal and feasible solution, does not attain the required solution value is called a sub-optimal sub-branch problem.

A sub branch problem is said to be fully explored if we have found the integer optimum of the sub branch problem, but also if it is found to be empty, empty of integer solutions, sub-optimal or empty of non sub-optimal integer solutions. Since sub-branch problems are solved by treating them as integer programming problems in their own right, we refer to them in that capacity as head problems, i.e. a head problem is a problem which is split into sub-problems by branching.

A main branch is said to end if we establish that the integer optimum (if it exists) does not obey the restrictions of any of its remaining unexplored further-our problems.

This may occur on the following indications:

a) A sub branch problem is found to be empty

We may interpret the branching restriction on the variable at

the node index as an inequality even where it is transformed from a binding inequality into an equation, when the sub branch problem is redefined as a head problem in order to branch further on the remaining variables above node-index.

Further out problems may then be said to be developed from each other by parametrically displacing the constants in (20.2.3) and (20.2.4). Once such a displacement results in the generation of an empty problem, this automatically implies the emptiness of all remaining further out problems. (See also Chapter XIII).

b) A sub-branch problem is found to be sub-optimal

The same logic as explained under a) for an empty problem applies here. If the required value of the objective function is added as a restriction, cases a) and b) become formally undistinguishable.

c) The (unbranched) solution of the sub-branch problem is found to be integer

This condition is verified only if the solution is not sub-optimal; therefore a newly identified integer solution is by implication an improved integer solution, and identifies a new lower limit of all acceptable solution values.

Sub-optimality of the remaining further-out problems is then implied.

Note that all three of these end of main branch indications relate to the unbranched solution of a sub branch problem of the main branch itself, not to sub-problems of these sub-branch problems. For example an integer programming problem may contain an equation restriction let us say $3\ x_2 = 7\ x_5$, and we may branch on x_2 at $x_2 = 4.5$. As $x_2 = 5$ implies $x_5 = 2\ 1/7$, the $x_2 = 5$ sub problem is empty of integer solutions, this fact does not contradict the possibility that further out problems, e.g. the $x_2 = 7$ sub-problem may yet yield an integer solution!

One computational point is useful to mention here, before we outline the algorithm in more detail. This relates to re-entry of the LP algorithm with an infeasible solution. We can specify both upper and lower limits simply by putting negative entries in the appropriate tableau-cells, and in the case of a lower limit also altering the non-updated limit.

The following tableau - which is in fact simply a set of figures rather than the solution of a specific problem may illustrate the point.

TABLEAU 20.2 A

RECAPITULATION OF LIMITS.

NAME	!!	S 1	S 2	!!	VALUE	DIST.
X 1	!!	-1.10	0.70	!!	2.15	97.85
X 2	!!	-0.10	0.50	!!	7.65	92.35
T	!!	3.50	1.17	!!	5.40	X
UB	!!	X	X	!!	X	X
LB	!!	0	0	!!	X	X

It will be recalled (see section 10.3 and 10.4) that lower limits are not part of the LP algorithm as such, and that they are - in the computational implementation offered in this book - conventionally stored in the $m + 3^d$ row of the tableau, and stay in their original place. Hence the two zeros of the lower bound row indicate that x_1 and x_2 as reported in the tableau itself are the true values of the variables their lower limits being zero.

The upper limits on the other hand go with the variables and are re-formed when a variable is eliminated as the sum of its old value and the distance from its upper limit. The computational implementation of the branching method offered here uses an integer programming adaptation of the basic LP algorithm, called INTP which is discussed in more detail later on in Section 21.4. (The same procedure INTP is also employed by the cutting algorithm.)

It is assumed that the upper limits on x_1 and x_2 are so far both at 100. If we branch on x_2, the $x_2 \geq 8$ restriction is put by the simple device of replacing the entry of 7.65 in the x_2-row/value column by 7.65 - 8 = -0.35 while at the same time setting the un-updated lower limit at 8.

The tableau now becomes:

TABLEAU 20.2 B

LOWER LIMIT REENTRY TABLEAU.

NAME	!!	S 1	S 2	!!	VALUE	DIST.
X 1	!!	-1.10	0.70	!!	2.15	97.85
Y 2	!!	-0.10	0.50	!!	-0.35	92.35
T	!!	3.50	1.17	!!	5.40	X
UB	!!	X	X	!!	X	X
LB	!!	0	8	!!	X	X

where y_2 is (see section 10.4) the distance between x_2 and its lower limit. As far as the lower limit is concerned, the basic LP algorithm as discussed in Part I and written down as the LINP procedure would suffice. (The phase I substitute objective function does not distinguish between slack-variables and elements of $\underline{x}$, it just picks negative valued variables). The one further bit of programming done to serve this aspect of the lower limit is an adaptation of the reporting procedure of section 10.4 which distinguished between branched limits in an exit-tableau, and initial limits in the separately stored set-up tableau.

The upper limit is dealt with in an analogeous way, the restriction $x_2 \leq 7$ is put in the tableau, simply by reporting its upper limit distance to be 7 - 7.65 = - 0.65, and the opening problem of the x_2 lower branch is written as follows:

TABLEAU 20.2 C

UPPER LIMIT REENTRY TABLEAU.

NAME	!!	S 1	S 2	!!	VALUE	DIST.
X 1	!!	-1.10	0.70	!!	2.15	97.85
X 2	!!	-0.10	0.50	!!	7.65	-0.65
T	!!	3.50	1.17	!!	5.40	X
UB	!!	X	X	!!	X	X
LB	!!	0	0	!!	X	X

i.e. x_2 is in excess of its upper limit, the excess being 0.65. The integer programming adaptation of the LP algorithm INTP as distinct from LINP) will react to this tableau by generating the appropriate substitute objective function, flicking the x_2-row round as a b_2 row and make the required step.

TABLEAUX 20.2 D AND E

ELEMINATION OF AN UPPER LIMIT.

NAME	!!	S 1	S 2	!!	VALUE	DIST.
X 1	!!	-1.10	0.70	!!	2.15	97.85
B 2	!!	0.10	(-0.50)	!!	-0.65	7.65
T	!!	3.50	1.17	!!	5.40	X
UB	!!	X	X	!!	X	X
LB	!!	0	0	!!	X	X

NAME	!!	S 1	B 2	!!	VALUE	DIST.
X 1	!!	-0.96	1.40	!!	1.24	98.76
S 2	!!	-0.20	-2	!!	1.30	X
T	!!	3.73	2.34	!!	3.88	X
UB	!!	X	7	!!	X	X
LB	!!	0	0	!!	X	X

The same facility also makes it possible to generate the next further out problem, by subtracting the branching-restriction column from the value column and adding it to the upper limit distance column (and changing the separately stored constant of the upper limit restriction).

Hence a re-entry tableau of the $x_2 = 6$ sub-problem is written as follows:

TABLEAU 20.2 F

A FURTHER-OUT REENTRY TABLEAU.

NAME	!!	S 1	B 2	!!	VALUE	DIST.
X 1	!!	-0.96	1.40	!!	-0.16	100.16
S 2	!!	-0.20	-2	!!	3.30	X
T	!!	3.73	2.34	!!	0.54	X
UB	!!	X	6	!!	X	X
LB	!!	0	0	!!	X	X

and the same method would always produce a tableau which can be handled by the Phase I of INTP, even if the result would include negative distances from upper limits.

We are now in a position to summarise the algorithm in some detail. The branching procedure may be called at any node-level. It performs the following operations

. it reserves memory space for an upperbranch tableau and a lower branch tableau, which are released when returning to the point of call
. it branches on one variable, and for the appropriate values of that variable generates a call to itself, one node-level higher
. it writes any integer solutions of which the solution value exceeds the previously attained one, to an output-buffer tableau.
. it makes appropriate adjustments to the branching record.

Since each call generates further calls until the problem is exhausted, the effect of a single call is to ask for solution of the integer programming problem in which the variables below and at the node-index are set at specific integer values, whereas the variables above the node-index are specified as integer restricted variables, but will generally be fractionally valued in the solution offered at the moment of calling.

We now proceed as follows:

Preliminary

Apply the Simplex Algorithm to the continuous problem (If an empty or unbounded problem is met at this stage, terminate the

algorithm).

Set the minimum required solution value at -10^{24}.

Call the branching procedure at node-level 1, offering the continuous problem as head problem.

Once inside the branching procedure, our operations-summary is as follows:

Stage 1

Search for a variable to branch on

Search for the integer restricted variable $x_{j(k)}$, which is furthest away from any integer value. If none is found, all integer-restricted variables being integer valued, declare the current head problem the integer optimum and exit to the point of call. (This can only happen at this stage if the head problem is the continuous problem).

Stage 2 high

Open the higher branch

Mark the higher branch as being opened, and the lower branch as unexplored. Copy the current head problem into the higher branch tableau and make the adjustments needed to convert it into a re-entry tableau of the opening problem of the higher main branch.

Stage 2a high

Re-enter the Simplex Method (higher branch)

If the current higher branch sub-problem is found empty or its solution sub-optimal, mark the higher branch as exhausted, and go to stage 3, (otherwise proceed to stage 2b high).

Stage 2b high

Check for integer solution (higher branch)

If the current solution is integer in all integer-restricted variables, write the current tableau to the output-buffer, set the required solution value at its new higher figure and mark the higher branch as exhausted. (This is of necessity the case at maximum node-level, hence the branching process does not carry on indefinitely). Go to Stage 3.

Stage 2 low

Open the lower branch

Mark the lower branch as being opened.

Copy the current head problem into the lower branch tableau and make the adjustment needed to convert it into a re-entry tableau of the opening problem of the lower main branch.

Stage 2a low

Re-enter the Simplex Method (lower branch)

If the current lower branch sub-problem is empty, or its solution sub-optimal, mark the lower branch as exhausted, and go to stage 3, otherwise proceed to stage 2b low.

Stage 2b low

Check for integer solution (lower branch)

If the current subproblem is integer in all integer-restricted variables, write the current tableau to the output-buffer, set the required solution value at its now higher figure and mark the lower branch as exhausted.

Stage 3

Choose what to do next

If both branches are exhausted, exit to the point of call, i.e. to the end of the algorithm in the main programme if the node-level is one, and otherwise to the end of one of the two sides of stage 4, one node-level lower.

If the lower branch is unexplored, go to stage 2 low to open the lower branch.

If the higher branch is exhausted go to stage 4 low; if the lower branch is exhausted, go to stage 4 high.

If the solution value of the current lower branch sub-problem exceeds that of the current higher branch sub-problem, go to stage 4 low, otherwise proceed to stage 4 high.

Stage 4 high

Branch further (high)

Offer the current higher branch problem as head-problem, with

all branching restrictions (at or below the node-index, there are no other), coded as equations, as head-problem to the branching procedure itself, one node-level higher, thereby again entering the branching procedure at stage 1.

Stage 4a high

(When getting at this point the last-formulated higher branch problem has been fully explored)

Push upwards

Adapt the value-column, the upper limit distances column and the lower limit on $x_{h(k)}$, to form a re-entry tableau for the next further-out problem on the higher branch.

Go to stage 2a high, to solve the new further-out problem.

Stage 4 low

Branch further (low)

Often the current lower branch problem as head-problem, with all branching restrictions coded as equations, as head problem to the branching procedure itself, one node-level higher than the current one.

Stage 4a low

Push downwards

Adapt the value column, the upper limit distances column and the upper limit on $x_{j(k)}$, to form a re-entry tableau for the next further-out problem on the lower branch.

Go to stage 2a low, to solve the new further-out problem.

End of branching procedure.

We now illustrate the algorithm, by reference to the small 3-variable problem which was stated earlier in this section but which is repeated here.

Maximise $\tau = 9x_1 - 3x_2 + 2x_3$

Subject to $7x_1 - 3x_3 \leq 0$

$10x_1 + 11x_3 \leq 25$

$(0 \leq x_1 \leq 100,\ 0 \leq x_2 \leq 100,\ 0 \leq x_3 \leq 100,\ x_1, x_2$ integer)

The following operations are involved

Preliminary

Solve the continuous problem, finding an optimal and feasible solution, $x_1 = 2.50$, $x_2 = 5.83$, $x_3 = 0$, $\tau = 5$.

Enter the branching procedure with node-index 1

Stage 1

Find x_1 as $x_{j(k)}$, to be branched on.

Stage 2 high

Form the re-entry tableau as follows

TABLEAU 20.2 G

OPENING TABLEAU OF THE X1 HIGHER BRANCH (EMPTY)

NAME	!!	S 1	S 2	X 3	!!	VALUE	DIST.
Y 1	!!	-	0.10	1.10	!!	-0.50	97.50
X 2	!!	-0.33	0.23	2.57	!!	5.83	94.17
T	!!	1	0.20	0.20	!!	5	X
UB	!!	X	X	100	!!	X	X
LB	!!	3	-	-	!!	X	X

(Conform section 10.4 lower bounds are stored in the cells which correspond to the variable's original position, the difference between the zero in the set up tableau and the 3 in the node 1 tableau, permits us to distinguish y, as a branched variable from the original x_1 variable even where their numerical codes are identical).

Stage 2a high

On re-entering the Simplex Algorithm the problem is found to be empty. (in the small example this is obvious without reference to a tableau, $x_1 \geq 3$ contradicts $10x_1 + 11x_3 \leq 25$, $x_3 \geq 0$).

We mark the x_1 higher branch as exhausted.

Stage 3

Since the lower branch is unexplored, we go to stage 2 low.

Stage 2 low

We form the re-entry tableau

TABLEAU 20.2 H

OPENING TABLEAU OF THE X1 LOWER BRANCH

NAME	!!	S 1	S 2	X 3	!!	VALUE	DIST.
X 1	!!	-	0.10	1.10	!!	2.50	-0.50
X 2	!!	-0.33	0.23	2.57	!!	5.83	94.17
T	!!	1	0.20	0.20	!!	5	X
UB	!!	X	X	100	!!	X	X
LB	!!	-	-	-	!!	X	X

Stage 2a low

On re-entering the Simplex Method, we find a new optimal tableau, which is

TABLEAU 20.2 I

CONTINUOUS OPTIMUM OF X1 LOWER BRANCH HEADPROBLEM

NAME	!!	S 1	S 2	B 1	!!	VALUE	DIST.
X 3	!!	-	0.09	-0.91	!!	0.45	99.55
X 2	!!	-0.33	-	2.33	!!	4.67	95.33
T	!!	1	0.18	0.18	!!	4.91	X
UB	!!	X	X	2	!!	X	X
LB	!!	-	-	-	!!	X	X

This solution is not suboptimal, we proceed to stage 2b low.

Stage 2b low

We find that the solution is not integer in x_2, therefore we go to stage 3.

Stage 3

Since the x_1 higher branch is empty, (has been exhausted) further branching now takes place on the last developed problem on the x_1 lower branch.

Stage 4 low

Offer the $x_1 = 2$ sub-branch problem as head problem, for further branching at node-level 2.

Stage 1

We find x_2 as the next variable to branch on.

Stage 2 high

We form the re-entry tableau

TABLEAU 20.2 J

REENTRY TABLEAU OF THE X1 = 2, X2 ≥ 5 PROBLEM

NAME	!!	S 1	S 2	B 1*	!!	VALUE	DIST.
X 3	!!	-	0.09	-0.91	!!	0.45	99.55
Y 2	!!	(-0.33)	-	2.33	!!	-0.33	93.33
T	!!	1	0.18	0.18	!!	4.91	X
UB	!!	X	X	2	!!	X	X
LB	!!	-	5	-	!!	X	X

* B1 NOT TO ENTER THE BASIS

N.B. In actual computational implementation x_1 would be temporarily recoded as s_1, and s_1 and s_2 as s_2 and s_3, to conform with the coding of equations discussed in section 10.2.

Stage 2a high

We find a new optimum tableau as follows:

TABLEAU 20.2 K

OPTIMUM OF THE X1 = 2, X2 ≥ 5 PROBLEM

NAME	!!	Y 2	S 2	B 1*	!!	VALUE	DIST.
X 3	!!	-	0.09	-0.91	!!	0.45	99.55
S 2	!!	-3	0	-7	!!	1	X
T	!!	3	0.18	7.18	!!	3.91	
UB	!!	95	X	2	!!	X	X
LB	!!	-	5	-	!!	X	X

* B1 NOT TO ENTER THE BASIS

This solution is not sub-optimal, hence we proceed to stage 2b high.

Stage 2b high

We find the solution integer, therefore $x_1 = 2$, $x_2 = 5$, $\tau = 3.91$ is listed as the best so far obtained integer solution. We mark the x_2 higher branch (of the $x_1 = 2$ subproblem) as exhausted, and require all future solutions to satisfy the condition $\tau > 3.91$.

Stage 3

Since the x_2 lower branch is so far unexplored, we go to stage 2 low.

Stage 2 low

We form the re-entry tableau as follows

TABLEAU 20.2 L

REENTRY TABLEAU OF THE X1 = 2, X2 ≤ 4 PROBLEM

NAME	!!	S 1	S 2	B 1*	!!	VALUE	DIST.
X 3	!!	-	0.09	-0.91	!!	0.45	99.55
X 2	!!	-0.33	-	2.33	!!	4.67	-0.67
T	!!	1	0.18	0.18	!!	4.91	X
UB	!!	X	X	2	!!	X	X
LB	!!	-	-	-	!!	X	X

This is an empty problem, the substitute objective function is in this case:
sum of infeasibilities = $b_2 = -0.33\ s_1 + 2.33\ b_1^*$ and since b_1 is an equation slack, no permitted incoming variable is found. We mark the x_2 lower branch (of the $x_1 = 2$ subproblem) as exhausted.

Stage 3

Since both main branches at node-level 2 are exhausted we return to the point of call which is the end of stage 4 low, having found the integer optimum of the $x_1 = 2$ sub-problem.

Stage 4a low (node-level 1)

We develop the lower branch re-entry tableau as follows

TABLEAU 20.2 M

REENTRY TABLEAU OF THE X1 =1 FURTHER OUT PROBLEM

NAME	!!	S 1	S 2	B 1*	!!	VALUE	DIST.
X 3	!!	-	0.09	-0.91	!!	1.36	98.64
X 2	!!	-0.33	-	2.33	!!	2.33	97.67
T	!!	1	0.18	0.18	!!	4.73	X
UB	!!	X	X	1	!!	X	X
LB	!!	-	-	-	!!	X	X

* B1 NOT TO ENTER THE BASIS

Stage 2a low

We find the re-entry tableau already optimal and feasible without any steps, and the solution is not sub-optimal $\tau = 4.73 > 3.91$.

Stage 3

Since the x_1 higher branch is exhausted, we go to stage 4 low.

Stage 4 low

We offer the $x_1 = 1$ sub problem as head-problem, entering the branching procedure at node-level 2.

Stage 1

We find x_2 as the next variable to branch on.

Stage 2 high

The higher branch re-entry problem re-entry tableau is as follows:

TABLEAU 20.2 N

REENTRY TABLEAU OF THE X1 =1, X2 ≥ 3 PROBLEM.

NAME	S 1	S 2	B 1*	VALUE	DIST.
X 3	-	0.09	-0.91	1.36	98.64
Y 2	-0.33	-	2.33	-0.67	97.67
T	1	0.18	0.18	4.73	X
UB	X	X	1	X	X
LB	-	3	-	X	X

* B1 NOT TO ENTER THE BASIS

Stage 2a high

$$x_1 = 1,\ x_2 = 3,\ (y_2 = 0),\ \tau = 2.73$$

is found as optimal and feasible solution of the specified sub-problem. The solution of this problem is, however, sub-optimal, in view of the earlier found solution $x_1 = 2$, $x_2 = 5$, $\tau = 3.91$.

We mark the x_2 higher branch as exhausted, and go to stage 3.

Stage 3

Since the x_2 lower branch (of the $x_1 = 1$ sub-branch problem) is still unexplored, we go to stage 2 low.

Stage 2 low

The re-entry tableau for the $x_1 = 1$, $x_2 \leq 2$ sub problem is developed, but in the interest of avoiding tedious repetition it is not copied here.

Stage 2a low

The $x_1 = 1$, $x_2 \leq 2$ subproblem is found empty (contradicts the restriction $3x_2 \geq 7$).

We mark the x_2 lower branch of the $x_1 = 1$ subbranch problem as exhausted, and go to stage 3.

Stage 3

Having exhausted both the higher and the lower x_2 main branch problems, we return to the point of call i.e. the end of stage 4 low of node-level 1.

Stage 4a low

The next further-out problem on the x_1 lower branch is the $x_1 = 0$ subbranch problem, the re-entry tableau of this problem is as follows

TABLEAU 20.2 0

REENTRY TABLEAU OF THE X1 = 0 SUB-PROBLEM.

NAME	S 1	S 2	B 1*	VALUE	DIST.
X 3	-	0.09	-0.91	2.27	97.73
X 2	-0.33	-	2.33	-	100
T	1	0.18	0.18	4.55	X
UB	X	X	-	X	X
LB	-	-	-	X	X

* B1 NOT TO ENTER THE BASIS

The zero values of x_2 arises in this case because, with x_1 at the value $x_1=0$, the binding restriction $7x_1 \leq 3x_2$ determines the corresponding value of x_2.

In fact the tableau print was given as -0.00 and a further step was needed.

Stage 2a low

We find $x_1 = 0$, $x_2 = 0$ as solution, this solution is not sub-optimal. It was not actually the same vertex, the value of x_2 was exchanged against the slack of the restriction $7x_1 - 3x_2 \leq 0$, the latter restriction now being classified as not binding with the slack-variable entering the basis of the value $s_1 = 0$.

This solution is not sub-optimal, we therefore proceed to stage 2b low.

Stage 2b low

The current sub-problem is found integer. We record the new solution $x_1 = 0$, $x_2 = 0$ as the best so far found one and set the required solution value as $\tau > 4.55$, and mark the x_1 lower branch as exhausted.

Stage 3

Having exhausted both the x_1 higher branch and the x_1 lower branch we exit from the branching procedure.

For the leading variable x_1 that is also the exit from the algorithm, $x_1 = 0$, $x_2 = 0$ with $x_3 = 2.27$ is the integer optimum.

20.3 Branching methods developed by other authors

We now briefly discuss the two main published branching algorithms (Dakin's [7] and Land and Doig's [26].

We take Dakin's first.

As already indicated, Dakin uses inequalities. Accordingly, in the example-problem discussed above, the first sub-problem is the $x_1 \geq 3$ sub problem. This is empty, and further branching takes place on the basis of the solution of the $x_1 \leq 2$ sub problem, for the same reasons as listed above for selecting the $x_1 = 2$ subproblem for further branching on other variables i.e. x_2. The notion of developing further-out problems i.e. exploring other integer values of variables which have already been branched upon is dropped, in favour of that of the alternative problem.

This feature of Dakin's algorithm is activated when the $x_1 \leq 2$, $x_2 \geq 5$ subproblem has been solved.

This problem has a solution and it is in fact an integer solution, i.e. $x_1 = 2$, $x_2 = 5$. Despite the fact that this is an integer optimum of a sub-problem, the alternative problem $x_1 \leq 2$, $x_2 \leq 4$ is tackled - as was done above with $x_1 = 2$, $x_2 = 4$ subproblem, the difference as yet still being solely the use of inequalities.

The equivalent subproblem when using equations is empty but when inequalities are used, the solution $x_1 = 1\ 5/7$, $x_2 = 4$, $x_3 = 5/8$, with $\tau = 4\ 6/7$, is found, a solution which satisfies the $x_1 \leq 2$ restriction, but not the $x_1 = 2$ restriction. Again branching on x_1 in the $x_1 \leq 2$, $x_2 \geq 4$ sub-problem makes the development of further out problems with different values of x_1 superfluous. The $x_1 \leq 1$, $x_2 \geq 4$

subproblems is in due course developed as a sub-problem of the $x_1 \leq 2$, $x_1 \geq 4$ subproblem, rather than as a further out problem of the x_1 lower branch.

The obvious drawback of this is that it involves the need to store information about the optima of a largish and generally unpredictable number of alternative problems.

This is in fact the very point on which Dakin criticised the Land and Doig procedure where the same problem arises in a different way.

Land and Doig develop, in effect further out problems in much the same way as is done by the algorithm offered in this book.

They do not, however, exhaust a subproblem before they develop the next subproblem at the same mode-level.

This makes it necessary to preserve information about solutions. If all other subproblems are subsequently found to be either empty of integer solutions or alternatively to possess lower solution values it may be necessary to use the solution of a particular subproblem to the purpose of splitting it into new ones by further branching.

Exhaustion of a particular subproblem as practiced by the algorithm offered in this book avoid this problem: either the sub-problem is empty of non-suboptimal integer solutions or we record a new best-so-far attained inter solution.

20.4 Text-listing of a recursive branching procedure

We now give the usual text-listing of a procedure-code.

It follows the outline given earlier in section 20.2 fairly closely, a feature which was enhanced by editing the text accordingly and inserting the labels stage 1, etc., even when they are not actually used by the code.

```
'PROCEDURE' BRANCH(T,M,N,NEQ,NAV,NIRV,MINV,ROWL,COLL,
EXT,IROWL,EXROWL,EXCOLL,NODE,EXNODE,RECORD);
'ARRAY' T,EXT;
'INTEGER' M,N,NEQ,NAV,NIRV,NODE,EXNODE;
'INTEGER' 'ARRAY' ROWL,COLL,IROWL,EXROWL,EXCOLL,RECORD;
'REAL' MINV;

'BEGIN'
  'INTEGER' I,J,R,K,NEXTJ,ROWN,COLN,II,REENTRY;
  'REAL' DIST,NUM;

  'PROCEDURE' INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,
  IROWLST,R,K,ROWN,COLN,REENTRY);
  'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,R,K,ROWN,COLN,REENTRY;
  'INTEGER' 'ARRAY' ROWLST,COLLST,IROWLST;
  'ALGOL';

  'BEGIN'

    'ARRAY' LT,HT[1:M+3,1:N+2];
    'INTEGER' 'ARRAY' LROWL,HROWL[1:M],LCOLL,HCOLL[1:N];

    'COMMENT'  CODING CONVENTION:
    THE BRANCHING RECORD IS KEPT IN THE ARRAY RECORD.

    ROW 0 OF THIS ARRAY IS RESERVED FOR THE NAME-CODES OF
    THE VARIABLES THAT HAVE BEEN BRANCHED ON.

    ROWS 1 AND 2 OF THE ARRAY RECORD CONTAIN THE RECORD OF
    THE STATUS OF THE VARIOUS BRANCHES, AS FOLLOWS:

    0  =  UNEXPLORED,
    1  =  PRELIMINARY EXPLORED, I.E. BEING OPENED,
    2  =  BEING EXPLORED FURTHER OUTWARDS, I.E. FULLY OPEN.
    3  =  EXHAUSTED.

    ROW 3 IS RESERVED FOR THE EQUATION NAME-CODES OF THE
    BRANCHED VARIABLES ASSOCIATED WITH ANY BEST-SO-FAR-FOUND
    INTEGER SOLUTION, IN ASSOCIATION WITH THE OUTPUT-BUFFER
    TABLEAU, WHERE THESE UPPER AND LOWER LIMITS ARE CODED AS
    EQUATIONS.

    THIS REFERS TO THE MOMENT OF BRANCHING, OR OF WRITING TO
    THE EXIT-TABLEAU.

    THE ACTUAL VALUES OF THE LIMITS AT WHICH THE BRANCHING
    RESTRICTIONS ARE SET, ARE STORED AS NORMAL UPPER AND
    LOWER LIMITS ON THE VARIABLES IN QUESTION.
    ;

    INITIATE AS UNEXPLORED:
    'FOR' I:=0,1,2 'DO' RECORD[I,NODE]:=0;

    STAGE 1:
    FIND A VARIABLE TO BRANCH ON:
    NEXTJ:=0;  DIST:=0;
```

```
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWL[I] < NAV+NIRV+1 'THEN' 'BEGIN'
  'COMMENT'
  INTEGER RESTRICTED VARIABLES MUST BE PLACED DIRECTLY
  AFTER THE FREE VARIABLES, THEREFORE WE RECOGNIZE THEM
  BY THIS CONDITION ON THEIR NAME-CODES. ;

  NUM := T[I,N+1]-ENTIER(T[I,N+1]);
  'IF' NUM>0.999 'THEN' T[I,N+1]:=ENTIER(T[I,N+1]+0.002);
  'IF' NUM<0.001 'THEN' T[I,N+1]:=ENTIER(T[I,N+1]);
  NUM := T[I,N+1]-ENTIER(T[I,N+1]);
  'IF' NUM > 0.5 'THEN' NUM:=1-NUM;
  'IF' ABS(NUM) > 0.00001 'AND' NUM > DIST 'THEN' 'BEGIN'
    II:=I; NEXTJ:=ROWL[I]; DIST:=NUM; 'END'; 'END';

'IF' NEXTJ=0 'THEN' 'BEGIN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' EXT[I,J]:=T[I,J];
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' EXROWL[I]:=ROWL[I];
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' EXCOLL[J]:=COLL[J];
  EXNODE := 0;  MINV := T[M+1,N+1];
  'GOTO' END OF BRANCHING; 'END';

RECORD[0,NODE] := NEXTJ;

STAGE 2 HIGH:
COPY HEAD PR TO HIGHER BR:
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' HT[I,J]:=T[I,J];
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' HROWL[I]:=ROWL[I];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' HCOLL[J]:=COLL[J];

OPEN HIGHER BRANCH:
RECORD[1,NODE] := 1;  RECORD[2,NODE] := 0;
HT[M+3,NEXTJ]:=HT[M+3,NEXTJ]+1;
HT[II,N+1]:=HT[II,N+1]-ENTIER(HT[II,N+1])-1;

STAGE 2A HIGH:
REENTER HIGHER BR:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  NUM:=HT[I,N+1]+HT[I,N+2];
  'IF' ABS(NUM) < 0.0000001
  'THEN' HT[I,N+2]:=0.0000001-HT[I,N+1]; 'END';
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' (HCOLL[J]>NAV 'AND' HCOLL[J]<NIRV+1)
'OR' (HCOLL[J]>10000+NAV 'AND' HCOLL[J]<10001+NIRV)
'THEN' 'BEGIN'
  NUM := HT[M+2,J]-ENTIER(HT[M+2,J]);
  'IF' NUM>0.999
  'THEN' HT[M+2,J]:=ENTIER(HT[M+2,J]+0.002);
  'IF' NUM<0.001
  'THEN' HT[M+2,J]:=ENTIER(HT[M+2,J]); 'END';

REENTRY:=1; R:=K:=0; ROWN:=COLN:=0;
INTP(HT,M,N,NEQ+NODE-1,NAV,HROWL,HCOLL,IROWL,R,K,ROWN,
COLN,REENTRY);
```

```
CHECK ON EMPTYNESS OF HB:
'IF' REENTRY = -1 'THEN' 'BEGIN'
  'COMMENT'
  EMPTY PROBLEM ON HIGHER BRANCH;

  'IF' MINV < -993 000 000 'AND' NODE > EXNODE
  'THEN' 'BEGIN'
    'COMMENT'
    NO INTEGER SOLUTION HAS BEEN FOUND SO FAR,
    THIS IS SO FAR THE HIGHEST BRANCHED SOLUTION.
    JUST IN CASE NOTHING BETTER IS FOUND,
    RECORD THIS SOLUTION FOR DIAGNOSTIC PURPOSES;
    'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
    'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' EXT[I,J]:=HT[I,J];
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' EXROWL[I]:=HROWL[I];
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' EXCOLL[J]:=HCOLL[J];
    'FOR' J:=1 'STEP' 1 'UNTIL' NODE 'DO'
    RECORD[3,J] := RECORD[0,J];
    EXNODE := NODE;  MINV := -995 000 000;  'END';

  RECORD[1,NODE]:=3;
  'GOTO' CHOOSE; 'END';

HB MAY NEED ANOTHER STEP:
'IF' 'NOT' K=0 'THEN' 'GOTO' REENTER HIGHER BR;

HIGHER BR NON EMPTY:
'IF' 'NOT' HT[M+1,N+1] > MINV 'THEN' 'BEGIN'
  'COMMENT'
  SUBOPTIMAL SOLUTION, TREAT AS EMPTY;
  REENTRY := -1;
  'GOTO' CHECK ON EMPTYNESS OF HB; 'END';

'IF' RECORD[1,NODE] = 2
'THEN' 'GOTO' FIND WHETHER HB INTEGER;

PUT EQUATION CODE IN HB:
K:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' HCOLL[J]=NEXTJ 'THEN' 'BEGIN'
  K:=J; HCOLL[J]:=1000+NODE; 'END';
'IF' K=0 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('EQUATION%FOR%NEW%HIGHER%BRANCH%
  ON%X')'); PRINT(NEXTJ,5,0);
  WRITETEXT('('%%NOT%FOUND')'); 'END';

'FOR' J:=1 'STEP' 1 'UNTIL' K-1,
K+1 'STEP' 1 'UNTIL' N 'DO'
'IF' HCOLL[J]>999+NODE 'AND' HCOLL[J]<1001+NODE+M
'THEN' HCOLL[J]:=HCOLL[J]+1;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' HROWL[I]>999+NODE 'THEN' HROWL[I]:=HROWL[I]+1;

STAGE 2B HIGH:
FIND WHETHER HB INTEGER:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' HROWL[I] < NAV+NIRV+1 'THEN' 'BEGIN'
  NUM := HT[I,N+1]-ENTIER(HT[I,N+1]);
```

```
  'IF' NUM>0.999
  'THEN' HT[I,N+1]:=ENTIER(HT[I,N+1]+0.002);
  'IF' NUM<0.001 'THEN' HT[I,N+1]:=ENTIER(HT[I,N+1]);
  NUM := HT[I,N+1]-ENTIER(HT[I,N+1]);
  'IF' ABS(NUM) > 0.00001
  'THEN' 'GOTO' CHOOSE; 'END';

RECORD HB OUTCOME:
MINV := HT[M+1,N+1];  EXNODE := NODE;
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' EXT[I,J]:=HT[I,J];
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' EXROWL[I]:=HROWL[I];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' EXCOLL[J]:=HCOLL[J];
'FOR' J:=1 'STEP' 1 'UNTIL' NODE 'DO'
RECORD[3,J] := RECORD[0,J];

RECORD[1,NODE]:=3;

'GOTO' CHOOSE;

STAGE 2 LOW:
COPY HEAD PR TO LOWER BR:
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' LT[I,J]:=T[I,J];
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' LROWL[I]:=ROWL[I];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' LCOLL[J]:=COLL[J];

OPEN LOWER BRANCH:
RECORD[2,NODE]:=1;
LT[II,N+2]:=LT[II,N+2]-ENTIER(LT[II,N+2])-1;

STAGE 2A LOW:
REENTER LOWER BR:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  NUM:=LT[I,N+1]+LT[I,N+2];
  'IF' ABS(NUM) < 0.0000001
  'THEN' LT[I,N+2]:=0.0000001-LT[I,N+1]; 'END';
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' (LCOLL[J]>NAV 'AND' LCOLL[J]<NIRV+1)
'OR' (LCOLL[J]>10000+NAV 'AND' LCOLL[J]<10001+NIRV)
'THEN' 'BEGIN'
  NUM := LT[M+2,J]-ENTIER(LT[M+2,J]);
  'IF' NUM>0.999
  'THEN' LT[M+2,J]:=ENTIER(LT[M+2,J]+0.002);
  'IF' NUM<0.001
  'THEN' LT[M+2,J]:=ENTIER(LT[M+2,J]); 'END';

REENTRY:=1; R:=K:=0; ROWN:=COLN:=0;
INTP(LT,M,N,NEQ+NODE-1,NAV,LROWL,LCOLL,IROWL,R,K,ROWN,
COLN,REENTRY);

CHECK ON EMPTYNESS OF LB:
'IF' REENTRY = -1 'THEN' 'BEGIN'
  'COMMENT'
  EMPTY PROBLEM ON LOWER BRANCH;
```

```
  'IF' MINV < -993 000 000 'AND' NODE > EXNODE
  'THEN' 'BEGIN'
    'COMMENT'
    NO INTEGER SOLUTION HAS BEEN FOUND SO FAR,
    THIS IS SO FAR THE HIGHEST BRANCHED SOLUTION.
    JUST IN CASE NOTHING BETTER IS FOUND,
    RECORD THIS SOLUTION FOR DIAGNOSTIC PURPOSES;
    'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
    'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' EXT[I,J]:=LT[I,J];
    'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' EXROWL[I]:=LROWL[I];
    'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' EXCOLL[J]:=LCOLL[J];

    'FOR' J:=1 'STEP' 1 'UNTIL' NODE 'DO'
    RECORD[3,J] := RECORD[0,J];
    EXNODE := NODE;  MINV := -995 000 000;  'END';

  RECORD[2,NODE]:=3;
  'GOTO' CHOOSE; 'END';

LB MAY NEED ANOTHER STEP:
'IF' 'NOT' K=0 'THEN' 'GOTO' REENTER LOWER BR;

LOWER BR NON EMPTY:
'IF' 'NOT' LT[M+1,N+1] > MINV 'THEN' 'BEGIN'
  'COMMENT'
  SUBOPTIMAL SOLUTION, TREAT AS EMPTY;
  REENTRY := -1;
  'GOTO' CHECK ON EMPTYNESS OF LB; 'END';

'IF' RECORD[2,NODE] = 2
'THEN' 'GOTO' FIND WHETHER LB INTEGER;

PUT EQUATION CODE IN LB:
K:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' LCOLL[J]=NEXTJ+10000 'THEN' 'BEGIN'
  K:=J; LCOLL[J]:=1000+NODE; 'END';

'IF' K=0 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('EQUATION%FOR%NEW%LOWER%BRANCH%
  ON%X')'); PRINT(NEXTJ,5,0);'
  WRITETEXT('('%%%NOT%FOUND')'); 'END';

'FOR' J:=1 'STEP' 1 'UNTIL' K-1,
K+1 'STEP' 1 'UNTIL' N 'DO'
'IF' LCOLL[J]>999+NODE 'AND' LCOLL[J]<1001+NODE+M
'THEN' LCOLL[J]:=LCOLL[J]+1;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' LROWL[I]>999+NODE 'THEN' LROWL[I]:=LROWL[I]+1;

STAGE 2B LOW:
FIND WHETHER LB INTEGER:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' LROWL[I] < NAV+NIRV+1 'THEN' 'BEGIN'
  NUM := LT[I,N+1]-ENTIER(LT[I,N+1]);
```

```
   'IF' NUM>0.999
   'THEN' LT[I,N+1]:=ENTIER(LT[I,N+1]+0.002);
   'IF' NUM<0.001 'THEN' LT[I,N+1]:=ENTIER(LT[I,N+1]);
   NUM := LT[I,N+1]-ENTIER(LT[I,N+1]);
   'IF' NUM > 0.00001
   'THEN' 'GOTO' CHOOSE; 'END';

RECORD LB OUTCOME:
MINV := LT[M+1,N+1];  EXNODE := NODE;
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' EXT[I,J]:=LT[I,J];
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' EXROWL[I]:=LROWL[I];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' EXCOLL[J]:=LCOLL[J];
'FOR' J:=1 'STEP' 1 'UNTIL' NODE 'DO'
RECORD[3,J] := RECORD[0,J];

RECORD[2,NODE]:=3;


STAGE 3:
CHOOSE:

'IF' (RECORD[1,NODE]=3 'AND' RECORD[2,NODE]=3)
'THEN' 'BEGIN'
   'COMMENT' HEAD PROBLEM EXHAUSTED;
   'GOTO' END OF BRANCHING; 'END';

'IF' RECORD[2,NODE] = 0
'THEN' 'GOTO' COPY HEAD PR TO LOWER BR;

'IF' RECORD[1,NODE]=3 'THEN' 'BEGIN'
   'COMMENT'
   CHOOSE THE LOWER BRANCH BY DEFAULT;
   'GOTO' BRANCH FURTHER IN LB; 'END';

'IF' HT[M+1,N+1] > LT[M+1,N+1] 'OR' RECORD[2,NODE]=3
'THEN' 'BEGIN'
   'COMMENT'
   CHOOSE THE HIGHER BRANCH;
   'GOTO' BRANCH FURTHER IN HB; 'END';

'GOTO' BRANCH FURTHER IN LB;

STAGE 4 HIGH:
BRANCH FURTHER IN HB:
RECORD[1,NODE]:=2;
'IF' NODE = NIRV 'THEN' 'GOTO' RECORD HB OUTCOME;
BRANCH(HT,M,N,NEQ,NAV,NIRV,MINV,HROWL,HCOLL,
EXT,IROWL,EXROWL,EXCOLL,NODE+1,EXNODE,RECORD);

STAGE 4A HIGH:
PUSH UPWARDS:
RECORD[1,NODE] := 2;
K:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' HCOLL[J]=1000+NODE 'THEN' K:=J;
```

```
'IF' K=0 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('%INTEGER%REQUIREMENT%ON%X')');
  PRINT(NEXTJ,5,0);
  WRITETEXT('('%%%IN%THE%HIGHER%BRANCH%%%
  NOT%FOUND')'); 'END';

'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO' 'BEGIN'
  HT[I,N+1]:=HT[I,N+1]-HT[I,K];
  HT[I,N+2]:=HT[I,N+2]+HT[I,K]; 'END';

HT[M+3,NEXTJ]:=HT[M+3,NEXTJ]+1;
NUM := HT[M+3,NEXTJ]-ENTIER(HT[M+3,NEXTJ]);
'IF' NUM>0.999
'THEN' HT[M+3,NEXTJ]:=ENTIER(HT[M+3,NEXTJ]+0.002);
'IF' NUM<0.001 'THEN' HT[M+3,NEXTJ]:=ENTIER(HT[M+3,NEXTJ]);
'GOTO' REENTER HIGHER BR;


STAGE 4 LOW:
BRANCH FURTHER IN LB:
RECORD[2,NODE]:=2;
'IF' NODE = NIRV 'THEN' 'GOTO' RECORD LB OUTCOME;
BRANCH(LT,M,N,NEQ,NAV,NIRV,MINV,LROWL,LCOLL,
EXT,IROWL,EXROWL,EXCOLL,NODE+1,EXNODE,RECORD);

STAGE 4A LOW:
PUSH DOWNWARDS:
RECORD[2,NODE]:=2;
K:=0;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' LCOLL[J]=1000+NODE 'THEN' K:=J;

'IF' K=0 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('%INTEGER%REQUIREMENT%ON%X')');
  PRINT(NEXTJ,5,0);
  WRITETEXT('('%%%IN%THE%LOWER%BRANCH%%%
  NOT%FOUND')'); 'END';

'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO' 'BEGIN'
  LT[I,N+1]:=LT[I,N+1]-LT[I,K];
  LT[I,N+2]:=LT[I,N+2]+LT[I,K]; 'END';

LT[M+2,K]:=LT[M+2,K]-1;
'GOTO' REENTER LOWER BR;


END OF BRANCHING:
END OF BRANCHING PROCEDURE:

'END'; 'END';
```

```
TEXT-LISTING OF THE MAIN PROGRAMME, WHICH SOLVES THE CONTINUOUS
HEADPROBLEM, AND SETS THE CALL TO THE BRANCHING PROCEDURE.

'BEGIN' 'INTEGER' M,N,NAV,NEQ,REENTRY,
NIRV,EXNODE,I,J,JJ,R,K,ROWN,COLN;
'REAL' MINV;

'PROCEDURE' BRANCH(T,M,N,NEQ,NAV,NIRV,MINV,ROWL,COLL,
EXT,IROWL,EXROWL,EXCOLL,NODE,EXNODE,RECORD);
'ARRAY' T,EXT;
'VALUE' NODE;
'INTEGER' M,N,NEQ,NAV,NIRV,NODE,EXNODE;
'INTEGER' 'ARRAY' ROWL,COLL,
IROWL,EXROWL,EXCOLL,RECORD;
'REAL' MINV;
'ALGOL';

'PROCEDURE' INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,
IROWLST,R,K,ROWN,COLN,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,R,K,ROWN,COLN,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST,IROWLST;
'ALGOL';

'PROCEDURE' REPO(T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL;
'ALGOL';

'PROCEDURE' IREP(TRIVT,T,M,N,NEQ,NAV,ROWL,COLL);
'ARRAY' TRIVT,T; 'INTEGER' M,N,NEQ,NAV;
'INTEGER' 'ARRAY' ROWL,COLL;
'ALGOL';

'PROCEDURE' MATI(MATR,MB,NB,FR,FC);
'ARRAY' MATR; 'INTEGER' MB,NB,FR,FC;  'ALGOL';

'PROCEDURE' TABO(MATR,M,N,SR,SC,RH,ER,ROWLST,COLLST);
'ARRAY' MATR; 'INTEGER' M,N,SR,SC,RH,ER;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';

'COMMENT'
  MIXED INTEGER PROGRAMMING, ADMITTING ANY INTEGER VALUES,
  BY BRANCHING.

PRESENTATION OF DATA:
FIRST THE NUMBER OF RESTRICTIONS I.E. M,
THEN THE NUMBER OF VARIABLES, I.E. N,
FOLLOWED BY THE NUMBER OF EQUATIONS,  NEQ,
FOLLOWED BY NAV, THE NUMBER OF VARIABLES
TO WHICH THE TACIT (NON-NEGATIVITY) RESTRICTION DOES   N O T
APPLY.
THEN AS LAST INTEGER PARAMETER, NIRV, THE NUMBER OF
INTEGER RESTRICTED VARIABLES.
```

```
THEREAFTER PUNCH EACH ROW OF THE COMPOSITE MATRIX

   A      B
   W      0
   U      0
   L      0

TO REPRESENT  A.X  < OR = B,
MAXIM  -W.X
    X  < OR = U
  AND X > OR = L

THE PROGRAMME READS ALL THE ELEMENTS OF THE UPPER BOUNDS VECTOR
U,  AND THE LOWER BOUNDS VECTOR L,
DESPITE THE FACT THAT FOR VARIABLES WITHOUT NON-NEGATIVITY
RESTRICTION,  THE MAIN PROCEDURE USES THESE NUMBERS ONLY FOR
BOUNDED VARIABLES.
;

MINV := -1 000 000 000;
M:=READ; N:=READ; NEQ:=READ; NAV:=READ; NIRV:=READ;
'BEGIN'
  'ARRAY' TA[1:M+4,1:N+2],EXT[1:M+3,1:N+2];
  'INTEGER' 'ARRAY' ROWL,EXROWL,IROWL[1:M],COLL,EXCOLL[1:N],
  RECORD[0:3,1:NIRV];

  PREFILL T WITH ZEROS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' TA[I,J]:=0;

  MATI(TA,M+3,N+1,0,0);
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
  TA[I,N+1] := TA[I,N+1]-TA[M+3,J];
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  TA[M+2,J] := TA[M+2,J]-TA[M+3,J];
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  TA[M+4,J] := TA[M+2,J];

  'FOR' I:=1 'STEP' 1 'UNTIL' M+3 'DO' TA[I,N+2]:=1000000;

  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' IROWL[I]:=1;

  REENTRY:=0;

  REENTER:

  R:=0; K:=0; ROWN:=0; COLN:=0;
  INTP(TA,M,N,NEQ,NAV,ROWL,COLL,IROWL,R,K,ROWN,
  COLN,REENTRY);
```

```
CHECK ON EMPTYNESS:
'IF' REENTRY = -1 'THEN' 'BEGIN'
  'COMMENT'
  EMPTY PROBLEM;
  NEWLINE(1);
  WRITETEXT('('MAIN%PROBLEM%EMPTY,%EVEN%WITHOUT%
  INTEGER%RESTRICTIONS')');
  'GOTO' REPORT ON SOLUTION; 'END';

'IF' REENTRY = 1 'THEN' 'BEGIN'
  'COMMENT'
  UNBOUNDED PROBLEM;
  NEWLINE(1);
  WRITETEXT('('UNBOUNDED%PROBLEM')');
  'GOTO' REPORT ON SOLUTION; 'END';

MAY NEED ANOTHER STEP:
'IF' 'NOT' K=0 'THEN' 'BEGIN'
  REENTRY := 1; 'GOTO' REENTER; 'END';

NON EMPTY:
BRANCH NOW:
REENTRY := 0;
BRANCH(TA,M,N,NEQ,NAV,NIRV,MINV,ROWL,COLL,
EXT,IROWL,EXROWL,EXCOLL,1,EXNODE,RECORD);

OUT:

REENTRY:=1;
'IF' MINV > -999 000 000 'THEN' REENTRY := 0;
'IF' REENTRY=0 'THEN' 'GOTO' CONVERT OPTIMUM;
NEWLINE(1);
WRITETEXT('('PROBLEM%EMPTY%OF%INTEGER%SOLUTIONS')');
NEWLINE(1);
'IF' MINV > -990 000 000 'THEN' 'GOTO' CONVERT OPTIMUM;

'IF' N < 14 'OR' M+N < 40
'THEN' TABO(TA,M,N,0,0,1,1,ROWL,COLL)
'ELSE' 'BEGIN'
  TABO(TA,M,0,0,N,0,1,ROWL,COLL);
  TABO(TA,0,N,M,0,1,0,ROWL,COLL); 'END';

'GOTO' REPORT ON SOLUTION;

CONVERT OPTIMUM:

'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' EXCOLL[J]>1000 'AND' EXCOLL[J]<1001+EXNODE
'THEN' 'BEGIN'
  JJ := RECORD[3,EXCOLL[J]-1000];
  EXCOLL[J] := JJ;
  'IF' EXT[M+3,JJ]=TA[M+3,JJ]
  'THEN' EXCOLL[J]:=EXCOLL[J]+10000; 'END';
```

```
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' EXCOLL[J]>1000+EXNODE 'AND' EXCOLL[J]<10000
  'THEN' EXCOLL[J]:=EXCOLL[J]-1;

  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
  'IF' EXROWL[I] > 1000 'THEN' EXROWL[I]:=EXROWL[I]-EXNODE;

  REPORT ON SOLUTION:
  'IF' ABS(REENTRY)=1 'THEN' REPO(TA,M,N,NEQ,NAV,ROWL,COLL)
  'ELSE' IREP(TA,EXT,M,N,NEQ,NAV,EXROWL,EXCOLL);

  END OF PROGRAMME:

'END'; 'END'
```

CHAPTER XXI

THE USE OF CUTS*

21.1 Elementary cuts, augmented cuts and combined cuts

The solution of integer programming problems by means of cuts has been pioneered by R. Gomory [15], [16] , and has been surveyed more recently at textbook level by Zionts [41] . The treatment of the material in this and the following sections differs from the work of these authors on the following points: Firstly, the interpretation of a combined cut as a cut on a combination of variables is believed to make the logic of "deeper" cuts more easy to appreciate than the algebraic approach followed in these source-texts. Some other points are of a more substantial nature. They concern the anticipation of fractional value of integer-restricted non basic variables, the permanent incorporation of upper limits into the problem itself, and formulation of cuts before the continuous problem has been solved.

A cut is an additional restriction, obtained on the basis of information contained in an updated tableau, which excludes an area in which no integer solution exists, from the feasible space area.

We will discuss the procedure of cutting, at this stage on the assumption that the continuous problem is solved first. If a solution is fractional, one "cuts" successive slices off from the feasible space area, until the integer optimum is found in a corner of the re-defined feasible space area. To understand why we can make cuts it is useful to interpret a vertex, e.g. the continuous optimum as a linear programming problem in its own right.

The relevant coordinate directions are the variables, which are associated with the columns of an updated tableau e.g. the continuous optimum. (See also section 8.11). Slacks of equations are not considered as variables in this context. They might as well be removed from the tableau, as they do not figure in the integer programming problem at all.

---

*A number of example-tableaux in this chapter are typed rather than computer-file listed. It was felt that exact fractions e.g. 1/11 instead of 0.09 are more useful in illustrating cuts than the two-digit rounded figures produced by the tableau-printing procedure.

We may distinguish several types of cuts.

An elementary cut is a restriction which excludes from the feasible space area a sub-area, the corners of which are the origin, and in each coordinate direction a point where a particular variable attains the next nearest integer value.

Example:

Consider the following integer programming problem:

Maximise $\tau = 3\ x_1 + x_2$

Subject to $8\ x_1 + 6\ x_2 \leq 15$

$2\ x_1 - 4\ x_2 \leq -1$

($0 \leq x_1 \leq 100$, $0 \leq x_2 \leq 100$, x_1, x_2 integer restricted)

This small example will be used in this and in some of the succeeding sections to illustrate the various types of cuts.

The "fancyhigh" upper limits have been set at a figure of 100 rather than the million used so far. This is because of the need to contain rounding errors not only relative to the rounded numbers, but also the absolute magnitude of the rounding errors themselves.

The effectiveness of an integer programming algorithm which uses cuts and does calculations in floating point numbers, is critically dependent on the precision of its calculations. If the correct value of some integer-restricted variable at a certain point of calculation is $x_k = 1600$, but because of rounding it is reported as 1599.983 we should either set our tolerance for accepting a number as integer at a difference in excess of 0.01 or we would risk to miss the correct solution altogether, as the next cut might well require $x_k \leq 1599$. To set the tolerance at a fairly narrow margin, and then not dealing with excessively high values of variables at all, is simpler in terms of programming, than to have a special loop to calculate the tolerance.

The continuous optimum of the demonstration problem is

$x_1 = 1\ 5/22$ and $x_2\ 19/22$.

We can map the non-negativity of x_1 and x_2, as well as some

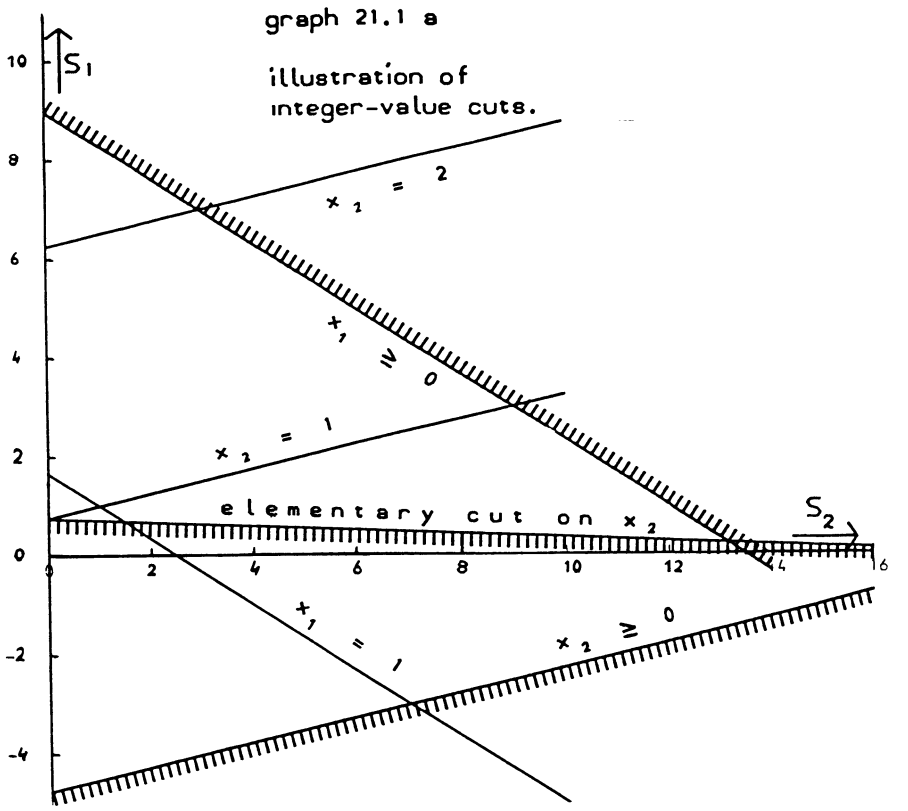
TABLEAU 21.1 A

CONTINUOUS OPTIMUM OF THE EXAMPLE

NAME	S 1	S 2			VALUE
X 1	0.09	0.14	! ≤	!	1.23
X 2	0.05	-0.18	! ≤	!	0.86
T	0.32	0.23	!	!	4.55

DITTO, NON-DECIMAL PRESENTATION.

NAME	S 1	S 2			VALUE
X 1	1/11	3/22	! ≤	! 1	5/22
X 2	1/22	-2/11	! ≤	!	19/22
T	7/22	5/22	!	! 4	6/11



other lines e.g. $x_1 = 1$ and $x_2 = 2$ in the s_1, s_2 coordinate plane. This has been done in graph 21.1a. The fancyhigh limits which are not operative in this example, have been left out of the illustrations.

The equations of these lines in the s_1, s_2 coordinate place are

$x_1 = 0$: $1/11\ s_1 + 3/22\ s_2 = 1\ 5/22$

$x_1 = 1$: $1/11\ s_1 + 3/22\ s_2 = 5/22$

$x_2 = 0$: $1/22\ s_1 - 2/11\ s_2 = 19/22$

$x_2 = 1$: $1/22\ s_1 - 2/11\ s_2 = -3/22$

$x_2 = 2$: $1/22\ s_1 - 2/11\ s_2 = -1\ 3/22$

(See also tableau 21.a, as well as graph 21.a).

We normally cut on a variable i.e. a cut is designed to exclude a sub-area in which a particular variable attains only fractional values. If the row of the associated Simplex tableau, which describes a variable, contains only positive entries or zeros the making of a cut is easy.

Thus the x_1-row reads $1/11\ s_1 + 3/22\ s_2 + x_1 = 1\ 5/22$ giving rise to the restriction $1/11\ s_1 + 3/22\ s_2 \leq 1\ 5/22$, the non-negativity of x_1. The slack of this restriction i.e. x_1 cannot exceed the number 1 5/22, hence $x_1 = 2$ (or $x_1 = 3$, 4 etc.) is outside the feasible area.

The next integer value to be investigated is therefore $x_1 = 1$, and a cut on x_1 will require $x_1 \leq 1$.

A somewhat similar situation arises when a row contains only negative elements, except in the value column. For example x_1 is implicitly assumed to have an upper limit which we conventionally put, in the absence of a specified upper limit at 100 . If that upper limit restriction is written explicitly, it reads: $-1/11\ s_1 - 3/22\ s_2 + b_1 = 98\ 17/22$ and quite clearly b_1 is an integer restricted variable which can be restricted to $b_1 \geq 99$.

Conversely, if we had found a restriction which reads (for example) $-2.3\ s_1 - 1.5\ s_2 + x_1 = 5\ 1/3$ we would have known that x_1 would have to satisfy $x_1 \geq 6$ (and the associated upper limit $b_1 \leq 94$.)

It is not always possible to write a cut-restriction which admits for such a simple interpretation. In the example, x_2 can be either greater than, or smaller than its current value of 19/22.

We may therefore distinguish at this stage two classes of cuts e.g. limit cuts which impose a specific limit (to be not greater than or not smaller than a certain integer number) on some variable, and elementary cuts.

Elementary cuts are cuts which we may write, even if we cannot associate a specific limit on a variable with the cut.

We can always calculate the coefficients of a cut-restriction by a method which follows the definition of a cut as given above. In the demonstration-example we can write an elementary cut on x_2. Inspection of the optimum tableau shows that introduction of s_1 in the basis reduces x_2 by 1/22 per unit of s_1. Therefore in the s_1-direction, x_2 will attain an integer value (zero), for $s_1 = 19/22 : 1/22 = 19$.

In the s_2-direction, x_2 will attain an integer value for $s_2 = 3/22 : 2/11 = 3/4$. In both cases, we need the absolute value of the coefficient, but for a negative coefficient the numerator is the difference with the next integer value of the variable which is being cut on.

The figure 19 and 3/4 indicate the values which s_1 and s_2 may attain before any solutions with integer values for x_2 come in the excluded area. We will therefore indicate them as the limiting value. More precisely, 19 is the limiting value of s_1 with respect to x_2 becoming zero, and 3/4 is the limiting value of s_2 with respect to x_2 becoming unity.

The limiting value of a non-basic variable is the value which that non-basic variable may attain without a particular basic variable (which we cut on) to exceed an adjoining integer value. When the tableau-cell which describes the effect of a non-basic variable on a variable to be cut on, is positive, (e.g. 1/22), the limiting value is the quotient of the fractional part of the current value of the basic variable cut on, divided by the particular tableau-cell.

$$\ell_j = (x_i - \text{entier } (x_i)) / t_{ij} \qquad (21.1.1a)$$

If the column for a non-basic variable contains a negative entry in the appropriate row (e.g. -2/11), the limiting value is the absolute value of the difference between the current value of the basic variable being cut on and the next higher integer, divided by the absolute value of the negative coefficient.

$$\ell_j = (\text{entier } (x_i + 1) - x_i)/(-t_{ij}) \qquad (21.1.1b)$$

The area to be excluded, is the area spanned by the origin (e.g. $s_1 = s_2 = 0$) and the points on the coordinate axes indicated by the limiting values (e.g. $s_2 = 0$, $s_1 = 19$, in the s_1-direction, and $s_1 = 0$, $s_2 = 3/4$ in the s_2-direction.)

Any solution vector which is a convex combination of the current solution and the points on its coordinate axes where the limiting values occur, will be characterized by values of the variable cut on, which either are between two integer values (e.g. x_2 between 1 and zero in the example), or, if only one integer number is involved, between that one integer and the current value.

The exclusion of this convex combination is made effective by requiring

$$\sum_{j=1}^{n} - 1/\ell_j \; v_j \leq - 1 \qquad (21.1.2)$$

where ℓ_j is the limiting value of the j^{th} variable, and ℓ_j the j^{th} variable itself. In the example, the cut on x_2 therefore becomes

$$- 1/19 \; s_1 - 1 \; 1/3 \; s_2 \leq - 1$$

As was already indicated above, a cut according to (21.1.2) with limiting values according to the definition given above, may be indicated as an elementary cut. If an elementary cut can be interpreted as requiring the variable which is cut on to attain the next lower integer value (e.g. $x_1 \leq 1$) we may speak of an upper limit cut.

The meaning of the analogous term lowerlimit cut will be obvious. It should also be borne in mind that we intend to employ a version of the basic Simplex LP algorithm, in which every variable has an upper and a lower limit. Limits on integer restricted variables will obviously be set at an integer value. Thus upper limit cuts are lower limit cuts on upper limit distances. If no information to the contrary is stated at the same time, it will be assumed that limit cuts are on specified variables.

Rows which refer to specified variables and only contain positive (negative) entries and zeros, give rise to limit cuts on these variables. We may, if we so wish, calculate and present a limit cut in precisely the same way as for an elementary cut which is not a limit cut.

For example the upper limit cut on x_1 could be calculated as follows (see also tableau 21.1a):

The limiting value of s_1, relative to x_1 becoming unity is 5/22: 1/11 = $2\frac{1}{2}$
The limiting value of s_2, relative to x_1 becoming unity is 5/22: 3/22 = 1 2/3

The reciprocals of $2\frac{1}{2}$ and 1 2/3 are 2/5 and 3/5. Therefore the elementary cut on x_1 is

$$- 2/5\ s_1 - 3/5\ s_2 \leq -1.$$

If this restriction is multiplied by 5/22, we obtain

$$- 1/11\ s_1 - 3/22\ s_2 \leq - 5/22$$

Addition of this restriction to the x_1-restriction itself

$$1/11\ s_1 + 3/22\ s_2 + x_1 = 1\ 5/22$$

yields:

$$x_1 \leq 1$$

There is, however, one essential point of difference between the two presentations of a limit cut. The slack-variable of

$$- 1/11\ s_1 \quad -3/22\ s_2 \leq - 5/22$$

is the difference between the value of x_1 which we now aim for, and any possible later value, and this slack is an integer-restricted variable. The slack-variable of

$- 2/5\ s_1 - 3/5\ s_2 \leq - 1$ is <u>not</u> integer-restricted. We shall come back to this point at a later stage.

We shall now pay some attention to the effectiveness of different types of cuts. A cut is more <u>effective</u> than another cut if it if it excludes all the fractional solutions which the other cut excludes, and some more as well. This definition of effectiveness if relative i.e. we cannot really say <u>how</u> effective a cut is. We cannot even always say that one cut is more effective than another. We will however also use the term in a wider, vaguer sense i.e. we may indicate that we <u>expect</u> a cut to lead quickly to the solution. Elementary cuts which are not also limit cuts are not a particular effective device for solving integer programming problems. They will be used only if other more effective types of cuts cannot be made. This is

apparent from the graph where the elementary cut on x_2 does not touch the integer optimum.

There are several things which can be done to increase the effectiveness of cuts as a means to quickly obtaining an integer solution.

The two devices which are the subject of this section are the augmentation of the limiting value of an integer-restricted non-basic variable (to unity), and the combined limit cut. The issue of augmentation is best discussed with reference to the limiting value of s_2 relative to x_2 becoming unity. In the original LP-tableau, the s_2-restriction contains only integer coefficients, all of which refer to integer-restricted variables, s_2 is itself an integer-restricted variable.

All points in the area excluded by the elementary cut on x_2 conform to the restriction $0 \leq s_2 \leq 3/4$.

If we push the limiting value of s_2 outwards to become $s_2 = 1$ this may bring some points with integer values for x_2 in the excluded area. But that will all be points with fractional values of s_2, hence those points will not include the integer optimum, where s_2 will be integer.

Whenever a limiting value of an integer restricted non-basic variable is found to be less than unity, we will substitute unity for it.

Instead of the elementary cut on x_2 we should have written - if nothing more effective appeared to be possible -

$$-1/19\ s_1 - s_2 \leq -1$$

A cut of this type, which is based on the augumentation of limiting values of integer restricted variables to unity, will be indicated as an augmented cut. Augmentation alone will not remedy the basic weakness of an elementary cut which is not also a limit cut, neither in general, nor in the particular example. (The true solution is $s_1 = s_2 = 1$ and this is still inside the amply fulfilled cut $-1/19\ s_1 - s_2 \leq -1$). If we cannot make a limit cut on a specified variable, the next best possibility is a limit cut on an integer combination of integer restricted variables. In an all integer problem, which the example is, this is always possible, but in the general mixed integer case this may, or may not be possible.

To relate the appropriate combination to a specified variable we will use an asterisk. Thus, x^*_2 is the auxiliary variable developed in combination with x_2. There are several possible combinations in the example, which would enable us to make a limit cut on a combination of integer restricted variables. For example, we might define an auxiliary variable

$$x^*_2 = x_2 + 2\ x_1$$

and the x^*_2 - row of the tableau would be found by adding twice the x_1-row to the x_2-row. The significance of the auxiliary variable row would in that case be

$$5/22\ s_1 + 1/11\ s_2 + x^*_2 = 3\ 7/22$$

From the definition of x^*_2 as $x_2 + 2\ x_1$ i.e. an integer combination of integer restricted variables it follows that x^*_2 is also an integer restricted variable. And we can write an upper limit cut, the effective significance of which is

$$x^*_2 \leq 3.$$

For program-technical reasons we shall restrict the use of this device to combination of a specified variable (or its upper limit distance) with non-basic variables (or their upper limit distances).

To make an effective use of the combination-device in connection with basic variables we would have to perform a systematic search for the appropriate combination. For non-basic variables the equivalent of the updated row is a (negative) unit vector i.e. the non-negativity of s_1 would be written explicitly as $-s_1 \leq 0$. This effectively means that we can, for the purpose of developing a cut, add or subtract integer numbers to or from the coefficients which refer to integer-restricted non-basic variables. This includes integer-restricted slacks. (See also section 21.4 for the treatment of upper limits on slack-variables. Thus, from the x_2-row

$$1/22\ s_1 - 2/11\ s_2 + x_2 = 19/22$$

we infer

$$-21/22\ s_1 - 2/11\ s_2 + x^*_2 = 19/22 \quad (x^*_2 = x_2 + s_1)$$

and we may make a lower limit cut on x^*_2 ($x^*_2 \geq 1$).

We may also develop auxiliary variables which are integer-restricted without being sign-restricted. Consider, for example, the difference between x_2 and s_2. We subtract the unwritten

s_2-row from the x_2-row, i.e. we add unity to the s_2- coefficient (the s_2-row reads $- s_2 \leq 0$).

The auxiliary variable equation then is

$$1/22\ s_1 + 9/11\ s_2 + x^*_2 = 19/22 \quad (x^*_2 = x_2 - s_2)$$

and we can write an upper limit cut on $x^*_2 \leq 0$. We will see later in this chapter that a systematic use of upper limits allows us to interpret even such rows as being associated with sign-restricted variables. Cuts of this type, i.e. cuts on a specified basic variable and an integer combination of integer restricted non-basic variables will be indicated as <u>combined cuts</u>.

In some cases a combined cut may be useful for a different reason than solely for being able to make a limit cut, or even in a situation where it would not be needed for that reason.

For example, a restriction (the x_3-row) might read:

$$1\ 1/10\ s_1 + 1/5\ s_2 + x_3 = 1\ 2/5.$$

We can in that case immediately formulate an upper limit cut on x_3. Assuming that the slack of the s_1-restriction is integer-restricted, the combined restriction

$$1/10\ s_1 + 1/5\ s_2 + x^*_3 = 1\ 2/5 \quad (x^*_3 = x_2 + s_1)$$

gives rise to a more effective cut, as may be seen by comparing the two cut-restrictions.

We perform this comparison, for the moment with both cuts in the elementary form

$$-2\ 3/4\ s_1 - -\tfrac{1}{2}\ s_2 \leq -1 \qquad (x_3 \leq 1)$$

or

$$-\ 3/4\ s_1 - \tfrac{1}{2}\ s_2 \leq -1 \qquad (x^*_3 = x_3 + s_1 \leq 1)$$

The point is simple that any set of figures which will satisfy the cut on x_3 itself, also satisifes the cut on $x^*_3 = x_3 + s_1$, but not the other way round. The systematic fact is that the limiting value of s_1 with respect to x^*_3 is greater than the similar figure with respect to x_3. Reduction of the coefficient for s_1 in the original restriction from 1 1/10 increases the limiting value by the inverse ratio from 4/11 to 4. Although the limiting value can be augmented to 1.00 the

augmented cut

$$- s_1 - \tfrac{1}{2} s_2 \leq - 1$$

is still less effective than the combined cut.

The choice of the most appropriate combination is to some extent influenced by the form in which cut-restrictions are presented. Cuts should if possible (which, in an all-integer problem means always) be written in a form in which the slack-variable of a cut-restriction is itself integer-restricted. Thus, in the example above, we would develop an x^*_3-cut

$$- 1/10\ s_1 - 1/5\ s_2 \leq - 2/5 \qquad (x^*_2 - \text{cut})$$

The restriction is equivalent with

$$- 1/4\ s_1 - \tfrac{1}{2}\ s_2 \leq - 1 \qquad (x^{**}_2 - \text{cut})$$

except for the integer form. i.e. the integer requirement on the slack-variable.

Throughout the rest of this chapter, cuts with integer-restricted slacks will be indicated with single asterisks, cuts with continuous slacks by letters with double asterisks.

There is in general a variety of combinations to be considered. In practice we can sort out the most effective combination implicitly by means of the following rule concerning the limiting values of integer restricted non-basic variables.

The limiting value of the x_j non-basic variable, relative to the x_i basic variable or an associated combination integer, is the highest of the following numbers:

a) The fractional part of the basic variable x_i divided by the fractional part of the tableau element (coefficient) c_{ij} if c_{ij} is positive, (x_i = 2 2/3, c_{ij} = 3 3/4 limiting value = 2/3: 3/4 = 8/9), or the fractional part of the basic variable x_i, divided by the absolute value of the difference between c_{ij} and the next lower integer number, if c_{ij} is negative (x_i = 2 2/3, c_{ij} = $-2\frac{1}{4}$ limiting value = 2/3: $(3 - 2\frac{1}{4})$ = 2/3: 3/4 = 8/9.

The implied combination contains in these two examples for a) the terms 3 x_i and - 3 x_i respectively, leaving in both cases a fractional term 3/4 x_j in the auxiliary expression.

b) The absolute value of the difference between the value of the basic variable and the next higher integer, divided by

the absolute value of the difference between the coefficient and the next higher integer, if c_{ij} is positive (x_i = 2 2/3, c_{ij} = 3 3/4 limiting value = (3 - 2/3): (4 - 3 3/4) = 1/3: $\frac{1}{4}$ = 1 1/3), or the absolute value of the difference between the value of the basic variable and the next higher integer, divided by the fractional part of the absolute value of the coefficient, if the coefficient is negative.

(x_i = 2 2/3, c_{ij} = - 2$\frac{1}{4}$ limiting value = (3 - 2 2/3): $\frac{1}{4}$ = 1/3: $\frac{1}{4}$ = 1 1/3).

The combination of variables in the implied auxiliary variable contains for the two examples under b) the terms 4 x_i and - 2 x_j, leaving in both examples a fractional term $-\frac{1}{4} x_j$ in the auxiliary expression.

c) The number 1.00 = unity, if no higher value has been found under a) or b).

The generalization of the above stated rules to non-basic slack variables with integer requirement (including slacks of limit-cuts) will be obvious. The operative significance of these rules is here illustrated with the following:

Example:

Maximise $\tau = 3x_1 + x_2 - x_3$

$8x_1 + 6x_2 + 3x_3 \leq 15$

$2x_1 - 4x_2 + x_3 \leq -1$

($0 \leq x_1 \leq 100$, $0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$) x_2, x_3 integer restricted).

This is a modification of our earlier demonstration example. We have dropped the integer requirement on x_1, and by implication also on s_1 and s_2, and added the third variable x_3.

The corresponding continuous optimum is given below

Name	s_1	s_2	x_3		Value
x_1	1/11	3/22	9/22	$\leq$	1 5/22
x_2	1/22	-2/11	-1/22	$\leq$	19/22
τ	7/22	5/22	2 2/11		4 6/11

Because x_1 is now a continuous variable, s_1 and s_2 also loose their integer restricted nature.

There are now two cuts in association with x_2.

One is the elementary cut on x_2 itself:

$$- 1/19\ s_1 - 1\ 1/3\ s_2 - 1/3\ x_3 \leq - 1$$

The other cut is on a combination

$$1/22\ s_1 - 2/11\ s_2 + 21/22\ x_3 + x^*_2 = 19/22 \quad (x^*_2 = x_2 - x_3)$$

and gives in first instance rise to

$$-1/19\ s_1 - 1\ 1/3\ s_2 - 1\ 2/19\ x_3 \leq - 1$$

This combination does not look very effective at first sight since 1/22, the coefficient in the x_2 restriction itself is a rather small number, but we cannot be sure on this point without further investigation. However the coefficient for x_3 is based on a limiting value of 19/22: 21/22 = 19/21, which should be augmented, and the second cut becomes

$$- 1/19\ s_1 - 1.1/3\ s_2 - x_3 \leq - 1$$

The cut on x_2 itself is, however found to be the most effective, hence the cut

$$- 1/19\ s_1 - 1.1/3\ s_2 - 1/3\ x_3 \leq - 1$$

is eventually put in the tableau.

We are however, not only interested in the effectiveness of individual cuts, but also in formulating cuts with integer, restricted slacks. This implies that we wish to form combinations on which a limit cut can be written, and certain restrictions on the choice of combinations arise from that desideratum.

21.2 Classes of cuts

Because there are different types of cuts to be made, we must state some priority rules as to which cuts to make in a particular tableau. It would be a waste of tableau-space and of calculations if all possible cuts were made.

At this point in our discussion, we state three main priority-classes of cuts or rather of auxiliary variables.

* Specified variables which permit a limit cut to be written on the specified variable itself.

As was explained in the previous section, limit cuts are more likely to be effective cuts than other types of cuts. We recognize a variable which permits a limit cut by the fact that the non-zero entries in its row (other than in columns which refer to equations slacks), are either all positive (upper limit cut) or all negative except the entry in the value column (lower limit cut). Therefore we will cut on or in association with a variable on which a limit cut can be made, in preference to a variable on which we cannot make a limit cut. Note that this does not always imply that we will write the limit cut. Even for a variable which permits a limit cut, a combination may lead to a more effective cut. For example, suppose (for the sake of argument) that the x_1-row reads:

	s_1	s_2	$\leq$	Value
x_1	1/11	$\frac{1}{2}$		1 1/22

Like the row which is actually there, these figures allow an upper limit cut.

Upper limit cuts on specified variables can in fact be made, simply by entering the appropriate negative entry in the upper limit distances column, i.e.

	s_1	s_2	$\leq$	Value	Distance
x_1	1/11	$\frac{1}{2}$		1 1/22	- 1/22

If s_2 is integer-restricted, we cannot only augment its limiting value from 1/11 to 1, we can also write a combined cut on $x^*_1 = x_1 + s_2$.

The associated auxiliary variable restriction is

$$1/11\ s_1 - \tfrac{1}{2}\ s_2 + x^*_1 = 1\ 1/22$$

and the limiting value relative to x^*_1 becoming 2 is $21/22 : \frac{1}{2} = 1\ 10/11$.

In this case we will not use the combination, because it would lead to loss of integer form (and require an explicit restriction,

rather than simply changing the upper limit). But even if we were to use the combination, the x^*_1 cut belongs to the top priority class. This does, incidentally imply that it would be written as a negative upper limit distance rather than as an explicit restriction.

*Specified variables which permit a limit cut to be written on an auxiliary variable associated with them.

When no limit cut on a specified variable itself can be written, a limit cut on an auxiliary variable has priority over a cut which is not a limit cut at all.

We recognize a variable of this "priority two" class by the lack of sign homogeneity between the coefficients in its row, in so far as these coefficients refer to integer restricted non basic variables, while there is sign-homogeneity between the coefficients which refer to continuous variable.

*Specified variables which do not permit a limit cut of any type to be written on or in association with them.

We recognize a variable of this type by the fact that its associated row contains, among the coefficients which refer to continuous non-basic variables, positive non-zero as well as negative non-zero numbers.

A cut on a variable of this class is made only when no cut or in combination with a variable of one of the two higher priority classes can be made.

There is a fourth, non-priority class of variables. It is not recommended to write cuts on slack variables, even where some slack variables are integer-restricted. This is because integer values for all integer-restricted slack-variable is a necessary rather than a sufficient condition for all integer solution, whereas integer values for all specified variables is sufficient.

In the continuous optimum of the demonstration example, the two slack-variables are in fact integer valued (zero), but x_1 and x_2 are both fractional. If one were to cut on a slack variable one might find that an integer value of the slack-variable is duly attained, while none of the specified variables attain an integer value.

Within each class of variables, we still have to decide which variable to cut on. The algorithm which we shall actually implement, uses for various priority-classes, three methods of resolving this choice, viz:

a) To cut on the variable with the lowest index.
b) To cut, space-reservation in the tableau permitting, on all variables which come in a particular priority-class, and
c) To sort out - for a particular incoming variable - the basic variable for which a cut is likely to lead to the highest limiting value.

21.3 The subsidiary cut

A major problem in integer programming is that a cut on one integer-restricted variable may lead to another integer-restricted variable, which so far had an integer value to become fractionally valued.

This is most obvious in the case of an integer-restricted non-basic variable. Zero is an integer number, and if an integer-restricted variable enters the basis at the fractional value an integer-restricted variable becomes fractionally valued. A related problem is that a single cut generally leads to one integer-restricted variable becoming integer-valued, we really want all integer-restricted variables to become integer-valued.

Our general approach to these problems is to anticipate these undesired possibilities, in particular when low-priority cuts have been used. When we have been forced to use a non-limit cut, we scan adjoining vertices, without fully making the corresponding steps. We perform the search operation for a pivot, and update the value and upper limit columns, but don't so far update the full tableau. The updating of the solution-vector is done outside the main tableau - which is therefore preserved - and we speak of a hypothetical step. A preliminary cut which is made on the basis of information obtained by way of a hypothetical step. That hypothetical step is the one indicated by a search operation in an attempt to find a solution which satisfies a cut or some cuts already made before.

A cut which is made in the main tableau itself, by reference to the solution-vector which that main tableau describes, is then indicated as a main cut. Limit cuts, when found will be made and the corresponding steps will also be actually made. No subsidiary cuts will therefore arise from limit cuts, and we will indicate limit cuts also as preliminary cuts. However, in this example we will treat them as main cuts as otherwise no subsidiary cuts could be illustrated in a small example.

In our demonstration example, the cut on x_1 indicates s_2 as pivot column variable, and the slack of the cut x^*_1 as pivot row. We assume, for the sake of argument, that the x^*_1 - cut is treated as a main cut, rather than as an initial cut

i.e. we will now consider the next vertex as hypothetical rather than making a step forthwith.

The search operation for a "hypothetical" pivot could initially be the same as normal re-entry of Phase I of the L.P. algorithm, but the following special features of the L.P.-search operation in an integer programming context become relevant once the first "hypothetical" step has been indicated:

a) The "efficient" rule of selecting the pivotal row to match a particular pivotal column is applied (see section 9.2). For the standard L.P. problem we compromised this rule, because it results in a bias for small pivots.

b) Once we have established that a particular column contains a negative entry against a violated restriction, and the column could give rise to additional subsidiary cuts, we will re-enter the "Phase I" search operation, on the basis of that particular column as pivotal column.

The significance of these rules is illustrated below, as follows:

Hypothetical step

Name	s_1	s_2	$\leq$	Value
x_1	1/11	3/22		1 5/22
x_2	1/22	-2/11		19/22
x^*_1	-1/11	(-3/22)		-5/22
τ	7/22	5/22		4 6/11

Partial update (column extract)

	x^*_1	$\leq$	Value
x_1	1		1
x_2	-1 1/3		1 1/6
s_2	-7 1/3		1 2/3
τ	1 2/3		4 1/6

The updated column-extract may now serve to indicate a variable which is fractionally valued in the hypothetical vertex e.g. x_2. A subsidiary cut is now developed with the help of a partial row-update. Just as in the case of main cuts we need only to cut on specified variables, i.e. only on x_2.

Partial update (row-extract) with cut on x_2

Name	s_1	x^*_1	$\leq$	Value
x_2	1/6	-1 1/3		1 1/6
s_2	2/3	(-7 1/3)		1 2/3
x^*_2	-1/6	-2/3		-1/6
τ	1/6	1 2/3		4 1/6

The x^*_2 - cut in this example is a limit-cut on the combination $x^*_2 = x_2 - 2\,x^*_1$.

To make the cut comparable with the main vertex we now make the backward step, i.e. we re-introduce x^*_1 as a basic variable, and eliminate the s_2-slack. This backward step can be performed on the row-extract. The operation reverses the initial hypothetical step (implemented on the extracts). Since the extract was initially updated we call this operation unupdating.

Unupdated row-extract with subsidiary cut

Name	s_1	s_2	$\leq$	Value
x_2	1/22	-2/11		19/22
x^*_1	-1/11	-3/22		-5/22
x^*_2	-5/22	-1/11		-7/22
τ	7/22	5/22		4.6/11

The unupdated x^*_2 cut-restriction is now amalgamated with the main tableau, which becomes:

Name	s_1	s_2	$\leq$	Value
x_1	1/11	3/22		1 5/22
x_2	1/22	-2/11		19/22
x^*_1	-1/11	-3/22		-5/22
x^*_2	-5/22	(-1/11)		-7/22
τ	7/22	5/22		4 6/11

When after making a subsidiary cut, we come to scan the hypothetical incoming variable column, once more in search for a Phase I pivot, we speak of a secondary reentry column.

At this point it is essential that the search for pivots uses the "efficient" rule of the largest quotient when selecting between a number or rows representing violated restrictions, and not the "conservative" rule of the smallest quotient. Thus when s_2 is again chosen as pivot column, the newly made x^*_2-cut becomes pivot row, rather than x^*_1 - row, which would lead to the same subsidiary cut as before.

This requires some adaptation of the standard L.P. algorithm where the "efficient" rule of row-selection is not always used.

As in the "Phase I" of the normal L.P. algorithm, a column is not chosen or even considered as pivot column, unless it contains negative elements which correspond with negative entries in the value column.

We therefore know that secondary re-entry columns are always bounded. Of the eligible minus-minus pairs, we will select the one which gives the biggest value of the incoming variable.

The re-entry of the pivotal column may or may not decide which subsidiary cut is going to match a particular column-variable. This will be the case if one of the following circumstances arises

a) The pivot row is a normal basic variable rather than a violated cut
b) The next hypothetical vertex is integer-valued

In the first case there is little point in making a further subsidiary cut in association with that particular column, in the second case the job has been done, as far as that column is concerned.

In order to verify that entry of a particular variable into the list of basic variables avoids (as far as possible) fractional values of previously integer valued variables we will re-enter the same column after making a subsidary cut and if necessary, add a new subsidiary cut which gives rise to a higher limiting value. The one exception to this rule arises in the case of a zero in the extract.

If a column-extract contains a zero in the pivotal column, no subsidiary cut on the variable associated with the corresponding row is developed, even it if is a fractionally valued integer-restricted variable.

Theorem

When we re-enter a column to search for a Phase I pivot after writing a subsidiary cut in association with a hypothetical step (the column being indicated by the previous entry of the same column), the pivotal row is either not a cut, or the last-written subsidiary cut is the restriction which gives rise to the largest value of the variable associated with the column, and hence is indicated as pivotal row.

Example in lieu of a proof:

The sign-arrangement $\begin{matrix} - & + \\ - & - \end{matrix}$

is systematic for the "unupdating" step. e.g.

	x^*_1	Value
s_2	-7 1/3	1 2/3
x^*_2	-2/3	-1/6

The number -7 1/3 in the s_2, x^*_1 cell is negative because it is reciprocal of a negative pivot (the -3/22). The number 1 2/3 is positive because it is the value of s_2 and the entries of - 2/3 and - 1/6 are negative because all entries in a cut-restriction row are negative. In the tableau containing the "main" cut, the slack of the new subsidiary cut is necessarily more negative than in the tableau-extract which was used to formulate it.

Column updating in the backward step, i.e. division by minus the pivot also maintains the negative element in the cut-restriction/pivot column cell, and the sign arrangement

$\begin{matrix} \ominus & + \\ - & - \end{matrix}$ always results in $\begin{matrix} - & - \\ - & - \end{matrix}$

The -1/11 in the x^*_2, s_2 cell means that there is a critical ratio for which the restriction becomes binding.

By definition (the definition of a cut as excluding the current solution), a cut restriction is not satisfied at the (hypothetical) vertex for which the cut was formulated. By assumption, i.e. the "efficient" rule of row selection, all

other cuts (e.g. the main cut and any other subsidiary cuts) are satisfied at a hypothetical vertex. The one exception to this rule other than elimination of a normal variable i.e. not reaching a cut, relates to zero coefficients. In either case, we will refrain from formulating a subsidiary cut.

Therefore the last formulated subsidiary cut is the most binding restriction. (Thus, in the example, the x^*_2 - cut with a critical ratio of 7 is more binding on s_2, than the previously formulated x^*_1 - cut, and this is because the x^*_2 - cut was formulated at a vertex where the x^*_1 - cut was satisfied. In the interest of restricting the total number of cuts we will therefore discard a subsidiary cut, whenever the immediately following re-entry of the same column gives rise to a new subsidiary cut.

We now re-enter the "Phase I" search operation for a hypothetical pivot, but restrict ourselves in first instance to the s_2 - column.

The purpose of such a "restricted re-entry" is to see whether we can perhaps replace the last-formulated subsidiary cut by one which is associated with a higher limiting value for the s_2 - variable. In order to limit the number of re-entries of the same column, we will resort to an unrestricted search for a hypothetical pivot, whenever the value of the incoming variable in the previous hypothetical vertex is at least equal to 10.

This qualification of the column-restriction is not present and we proceed as before.

Column-extract

Name	s_2	$\leq$	Value
x_1	3/22		1.5/22
x_2	-2/11		19/22
x^*_1	-3/22		-5/22
x^*_2	-1/11		-7/22
τ	5/22		4 6/11

Updated column-extract giving hypothetical vertex

Name	x^*_2	$\leq$	Value
x_1	$1\frac{1}{2}$		3/4
x_2	-2		$1\frac{1}{2}$
x^*_1	$-1\frac{1}{2}$		$\frac{1}{4}$
s_2	-11		$3\frac{1}{2}$
	$2\frac{1}{2}$		3 3/4

We have indeed reached the feasible space in one step. If this were not the case we would develop no further subsidiary cuts in association with this column. As it is the hypothetical vertex indicates a cut on x_1, and again we use a row-extract to implement that cut.

Row-extract with pivotal row and x_1-row (to be cut on)

Name	s_1	s_2	$\leq$	Value
x_1	1/11	3/22		1 5/22
x^*_2	-5/22	(-1/11)		1 7/22
τ	7/22	5/22		4 6/11

Name	s_1	x^*_2	$\leq$	Value
x_1	$-\frac{1}{4}$	$1\frac{1}{2}$		3/4
s_2	$2\frac{1}{2}$	-11		$3\frac{1}{2}$
x^*_1	-3/4	$-\frac{1}{2}$		-3/4
τ	$-\frac{1}{4}$	$2\frac{1}{2}$		3 3/4

Updated row-extract with cut

Name	s_1	s_2	$\leq$	Value
x_1	1/11	3/22		1.5/22
x^*_2	-5/22	-1/11		-7/22
x^*_1	-7/11	-1/22		-10/11
τ	7/22	5/22		4 6/11

When, as in this case, we have found a higher limiting value for a particular variable by cutting after "restricted" re-entry, we overwrite the previous subsidiary cut. The previously developed subsidiary cut in association with a hypothetical pivot in the s_2-column was (see tableaux on previous page) the x^*_2 cut, we therefore are left with two x^*_1 cuts, the main cut and a subsidiary cut.

The successor tableaux are:

Main tableau with hypothetical pivot

Name	s_1	s_2	$\leq$	Value
x_1	1/11	(3/22)		1 5/22
x_2	1/22	-2/11		19/22
x^*_1	-1/11	-3/22		-5/22
x^*_1	-7/11	-1/22		-10/11
τ	7/22	5/22		4 6/11

Corresponding updated column-extract

Name	x_1	$\leq$	Value
s_2	7 1/3		9
x_2	1 1/3		$2\frac{1}{2}$
x^*_1	1		1
x^*_1	1/3		$-\frac{1}{2}$
τ	-1 2/3		$2\frac{1}{2}$

The new hypothetical vertex is not feasible, and there is no point in the development of a further subsidiary cut in association with the s_2 - column.

On leaving the choice of the column free, s_1 is indicated as pivotal column, against the first x^*_1 - cut as pivotal row.

Main tableau with hypothetical pivot

Name	s_1	s_2	$\leq$	Value
x_1	1/11	3/22		1 5/22
x_2	1/22	-2/11		19/22
x^*_1	(-1/11)	-3/22		-5/22
x^*_1	-7/11	-1/22		-10/11
τ	7/22	5/22		4 6/11

Corresponding updated column-extract

Name	x^*_1	$\leq$	Value
x_1	1		1
x_2	$\frac{1}{2}$		3/4
s_1	-11		$2\frac{1}{2}$
x^*_1	-7		15/22
τ	$3\frac{1}{2}$		3 3/3

We will now develop a subsidiary cut on (a combination in association with) x_2, which will not be illustrated further here.

21.4 An integer programming adaption of the linear programming procedure

The procedure INTP offered in this section is a modification of the LINP procedure from section 12.3. The modification

concerns the already mentioned "efficient" rule of row selection, the "search only", "column-restricted" and "isolated step" re-entries an extension of the upper limit facilities, and the fact that outside Phase 0, only one step per call is made.

The first of these points largely relates to the internal working of the procedure. Thus, the instructions

```
REENTRY:= 0;
LINP (T, M, N, NEQ, NAV, ROWLST, COLLST, REENTRY);
or alternatively
REENTRY   :=  0;
IN: 'IF' REENTRY = - 10  'THEN' REENTRY:  = 1;
R  :=  K := ROWN := COLN := 0;
INTP (T, M, N, NEQ, ROWLST, COLLST, R, K, ROWN, COLN, REENTRY);
'IF' REENTRY  = - 10 'THEN' 'GOTO' IN;
```

both amount to asking for the solution of an ordinary LP problem. If LINP is employed, a compromise-rule of row selection is applied in doing so, if INTP is applied, the "efficient" rule of row-selection is employed in doing so, and only one step at a time is made. The exit value of - 10 for the re-entry parameter indicates the fact that a step has actually been made. The second point, the "search only" re-entry relates to hypothetical steps.

On exit, R will indicate the index of the last selected pivotal row, K the index of the last-selected pivotal column and ROWN and COLN the names of the pivot row and pivot column variable. On "normal" exit after normal entry, this is not a particularly useful information.

However, if the procedure is entered with the re-entry-parameter set a 2, no step will actually be made.

The instructions

```
REENTRY := 2; R:= K := ROWN := COLN := 0;
INTP (T, M, N, NEQ, NAV, ROWLST, COLLST, R, K, ROWN, COLN, REENTRY);
```

amount to a call for a "hypothetical" step. The rules for pivot-selection are (for K = 0), the rules which this version of the procedure employs normally. If K is initiated at a non-zero number, we ask only for a pivot in that particular column. In short, this is the call for a secondary re-entry. Thirdly, there is the facility of making an isolated prescribed step. If the re-entry-parameter is on re-entry set at 1, and

R, K, COLN AND ROWN are set at non-zero values, the step as indicated by the values of R, K, COLN, and ROWN will be made. This is for upadating tableau-extracts according to a previously indicated step, and for implementing earlier choices.

If the re-entry-parameter is 1, and the other parameters are zero, a single step, as indicated by the usual rules is made. The fourth point, the extension of upper limit facilities relates in particular to upper limit cuts. This version of the procedure handles upper limits on slack-variables and notes negative entries in the upper-limit distances column as violated upper limit restrictions. It accordingly activates Phase I.

Note that this is a basic re-specification of the LP problem. The linear programming problem is now specified as

$$\begin{array}{ll} \text{Maximise} & \underline{w}'\,\underline{x} \\ \text{Subject to} & \left.\begin{array}{l} \underline{d} \leq A\,\underline{x} \leq \underline{b} \\ \\ \underline{\ell} \leq \underline{x} \leq \underline{u} \end{array}\right) \end{array} \qquad (21.4.1)$$

The relation between (21.4.1) and the upper limits on the slack-variables is in fact

$$\underline{s} = -A\,\underline{x} + \underline{b} \leq \underline{b} - \underline{d} \qquad (21.4.2)$$

Just as in the case of upper limits on the specified variables (elements of $\underline{x}$), we would put a fancyhigh number as upperbound, where no meaningful upper limit on a slack-variable is intended.

These facilities make it possible to formulate an upper limit cut, by the simple process of putting the difference between the current fractional value of a variable and the next lower integer value, in the distances column. Thus, following the similar procedure as discussed in section 20.2 for branching restrictions the example in section 21.3 is written as follows:

Name	s_1	s_2	$\leq$	Value	Distance
x_1	1/11	3/22		1 5/22	-5/22
x_2	1/22	-2/11		19/22	99 3/22
τ	7/22	5/22		4 6/11	
Upper b	99	99			
Lower b	0	0			

The upper limits on x_2, s_1 and s_2 are "fancyhigh" limits, but the violated upper limit distance represents a limit cut.

The search operations would then signal the upper limit on x_1, coded as l0001 as pivotal row against s_1 as pivotal column.

A secondary advantage of this way of recording limit cuts is that irrespective of whether they are generated as main cuts or as subsidiary cuts, we can keep any reduced upper limits. (Explicit cuts may have to be discarded when they turn out to be amply fulfilled.)

Because of the relative ease with which one can make a limit cut, we will actually make limit cuts on all integer-restricted variables at each vertex, actual or hypothetical whenever that is possible.

The textual listing of the procedure INTP is now given as follows:

```
'PROCEDURE' INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,
IROWLST,R,K,ROWN,COLN,REENTRY);
'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,R,K,ROWN,COLN,REENTRY;
'INTEGER' 'ARRAY' ROWLST,COLLST,IROWLST;

'BEGIN' 'INTEGER' I,J,TRYR,TRYN,FIRST,LAST,PRESCRIBED ROW,
UNIFORM,Z;
'ARRAY' WORKSPACE[1:2,1:2];
'REAL' ASC,HIG,QUO,PREV POS,TQUO,PIV,COP,NUM,PIWEIGHT,
SAVED ASC;
'BOOLEAN' INVERTED,FEASIBLE,UPPERBOUND,POSITIVE,MEASURED,
SEARCH ONLY,PRESCRIBED COLUMN;

'REAL' 'PROCEDURE' INPROD(I,A,B,X,Y);
'VALUE' A,B; 'INTEGER' I,A,B; 'REAL' X,Y; 'ALGOL';

'PROCEDURE' PRODUCT(A,B,C); 'REAL' A,B,C;
'BEGIN'
  WORKSPACE[1,1]:=A;
  WORKSPACE[1,2]:=B; WORKSPACE[2,2]:=C;
  A:=INPROD(Z,1,2,WORKSPACE[1,Z],WORKSPACE[2,Z]);
  END OF PRODUCT; 'END';

'PROCEDURE' ORDL(T,M,N,ER,RH,ROWLST,COLLST);
'ARRAY' T; 'INTEGER' M,N,ER,RH;
'INTEGER' 'ARRAY' ROWLST,COLLST; 'ALGOL';
```

```
'COMMENT' LINEAR PROGRAMMING PROCEDURE,
ADAPTED FOR USE IN THE CONTEXT OF INTEGER PROGRAMMING.

INPUT SUPPLY AND SIGNIFICANCE OF THE OPERANDS AS IN
LINP, EXCEPT FOR THE FOLLOWING DIFFERENCES:

INTP HAS A THIRD LIST OF NAME-CODES AS PROCEDURE PARAMETER.
THIS IS THE LIST OF INTEGER RESTRICTED ROWNAMES.
THIS LIST IS  N O T  GENERATED BY THE PROCEDURE ITSELF.
THE FUNCTION OF IROWLST IN THE INTP PROCEDURE IS TO INHIBIT
UNDESIRABLE SETTING OF FANCYHIGH UPPER LIMITS.
IF THE I-TH ELEMENT IF IROWLST IS NOT A ZERO,
THE UPPER LIMIT DISTANCE OF THE I-TH SLACK-VARIBLE MAY
LEGITIMABLY BE ZERO, AND NO FANCYHIGH NUMBER IS PUT
INSTEAD.
THIS INFORMATION IS SUPPLIED ACCORDING TO THE FOLLOWING
CONVENTION:
0  =  NOT INTEGER RESTICTED,  #0  =  INTEGER RESTRICTED.

INTP RECOGNIZES UPPER LIMITS ON SLACK-VARIABLES, AND
THE NUMERICAL CONTENT OF THE N+2ND COLUMN,
THE UPPERBOUND DISTANCES COLUMN FOR BASIC VARIABLES
SHOULD ALSO BE SUPPLIED.

TO ACCOMODATE UPPER BOUNDS FOR SLACKS WHERE NO MEANINGFUL
UPPER BOUND IS INTENDED, A ZERO MAY BE SUPPLIED INSTEAD,
AND THE PROGRAMME WILL SUBSTITUTE A MILLION FOR IT.

THE CODING OF UPPER LIMITS ON SLACK-VARIABLES IS SIMILAR
TO THOSE ON ORDINARY VARIABLES, I.E. BY WAY OF AN
ENLARGEMENT OF 10000.
THE UPPER LIMIT ON THE I-TH SLACK-VARIABLE IS THEREFORE
CODED AS   11000+I.
;

'IF' K < 0 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('YOU %ERRONEOUSLY%PRESCRIBED%A
  NEGATIVE%COLUMN')');
  NEWLINE(1);
  WRITETEXT('('K%%=%%%')');
  PRINT(K,5,0); 'END';

'IF' K > N 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('YOU%ERRONEOUSLY%PRESCRIBE%A%COLUMN-INDEX%
  IN%EXCESS%OF%THE%TABLEAU-SIZE.')');
  NEWLINE(1);
  WRITETEXT('('K%%=%%%')');
  PRINT(K,5,0); 'END';

'IF' K > 0 'AND' (R<1 'OR' R>M)
'AND' 'NOT' (R=0 'AND' ABS(ROWN-COLN)=10000) 'THEN' 'BEGIN'
  NEWLINE(1);
  WRITETEXT('('YOU%ERRONEOUSLY%PRESCRIBE%A%ROW-INDEX%
  OUTSIDE%THE%TABLEAU.')');
  NEWLINE(1);
  WRITETEXT('('R%%=%%%')');
  PRINT(R,5,0); 'END';
```

```
WORKSPACE[2,1]:=1;

PRESCRIBED ROW := R; SEARCH ONLY := 'FALSE';
'IF' PRESCRIBED ROW=0 'AND' ROWN#0
'THEN' PRESCRIBED ROW:=ROWN;
INVERTED := 'TRUE'; FIRST:=1; LAST:=N;
PRESCRIBED COLUMN := 'FALSE';

'IF' 'NOT' REENTRY = 0 'THEN' 'BEGIN'
  'COMMENT'
  FOR REENTRY=0, THE NORMAL LP ALGORITHM IS FOLLOWED FROM THE
  START, INCLUDING THE FILLING OF THE NAMELISTS.
  PHASE 0 IS GONE THROUGH, AND  O N E  NORMAL STEP IS MADE.
  OTHERWISE, FOR NON-ZERO VALUES OF REENTRY, THE PROCEDURE
  EXPECTS AN ALREADY UPDATED TABLEAU, WITH NAMELISTS FILLED,
  WHICH NEEDS RE-OPTIMIZING AND/OR THE FINDING OF A NEW
  FEASIBLE SOLUTION.

  RECOGNIZED RE-ENTRY MODES (OTHER THAN THE NORMAL
  ENTRY MODE)
  ARE REENTRY=1 AND REENTRY=2.

  FOR REENTRY=1  JUST  O N E  STEP IS MADE.
  DEPENDING ON THE INITIAL VALUES OF THE ROW AND COLUMN-
  INDICES R AND K, THIS STEP MAY BE EITHER FREELY CHOSEN
  ACCORDING TO THE NORMAL SEARCH CRITERIA, OR RESTRICTED,
  OR EVEN PRESCRIBED.

  FOR R=0, K=0, THE STEP IS FREELY CHOSEN.

  FOR R=0, K#0, THE STEP HAS TO BE FOUND IN THE INDICATED
  COLUMN, OR NONE AT ALL IS MADE.
  THE PROBLEM MAY AT THAT POINT BE FOUND TO BE EMPTY OF
  A FEASIBLE SOLUTION IN THAT PARTICULAR COLUMN.

  FOR R#0, ROWN#0 , K#0, NO SEARCH IS PERFORMED, ONLY A
  PREDETERMINED STEP IS MADE.
  THE NAMECODES ROWN AND COLN SHOULD IN THAT CASE ALSO HAVE
  BEEN INITIALIZED AT THEIR APPROPRIATE VALUES.

  FOR REENTRY=2 NO ACTUAL STEP IS MADE AT ALL, BUT THE VALUES
  OF THE ROW-AND COLUMN INDICES AND THE NAMECODES ARE
  TRANSMITTED TO THE MAIN PROGRAMME.

  THE REENTRY PARAMETER ALSO SERVES AS EXIT PARAMETER.
  ALARM MESSAGES FOR EMPTY AND UNBOUNDED PROBLEMS ARE  N O T
  PRINTED BY THE INTP PROCEDURE ITSELF, INFORMATION TO THE
  MAIN PROGRAMME IS RECORDED VIA THE REENTRY PARAMETER.
  FOR NORMAL, I.E. OPTIMAL AND FEASIBLE EXIT,
  OR EVEN IF EMPTINESS OR UNBOUNDEDNESS, THOUGH PRESENT,
  IS NOT SO FAR REVEALED,
  THE REENTRY PARAMETER BECOMES EITHER ZERO OR MINUS 10,
  EVEN IF ITS VALUE ON ENTRY WAS DIFFERENT.

  AN UNBOUNDED PROBLEM IS INDICATED BY REENTRY = 1,
  AN EMPTY PROBLEM BY REENTRY = -1.
```

```
  REENTRY = 0 INDICATES AN ALREADY FEASIBLE PROBLEM,
  REENTRY = -10 INDICATES A PROBLEM WHICH IS NOT SO FAR
  FEASIBLE, AND REQUIRES MORE STEPS TO BE MADE, QUITE
  APART FROM OPTIMALITY.

  THIS VERSION OF THE PROCEDURE, WHICH IS ADAPTED TO INTEGER
  PROGRAMMING HAS THE FOLLOWING SPECIAL FEATURES:

  NOT ONLY SPECIFIED VARIABLES, BUT ALSO SLACK-VARIABLES,
  HAVE UPPER LIMITS.
  ON FIRST ENTRY, THE UPPERBOUNDS DISTANCES COLUMN SHOULD
  THEREFORE BE PRE-FILLED.

  THE NUMBERS SUPPLIED AS UPPERLIMIT   D I S T A N C E S
  FOR THE SLACK-VARIABLES, INDICATE THE INTERVAL OVER WHICH
  THE SPECIFIED COMBINATION OF VARIABLES IS ALLOWED TO VARY.

  THUS THE RESTRICTION IS
  SIGMA A I,J  LT OR EQ  B I, AND
  SIGMA A I,J  GT OR EQ  B I  -   L I,
  WHERE L I  IS THE LIMIT DISTANCE SPECIFIED FOR THE ITH
  SLACK-VARIABLE.

  GENERALLY, A SINGLE CALL TO THE INTEGER PROGRAMMING ADAPTATION
  OF THE LINEAR PROGRAMMING PROCEDURE, CAUSES THE MAKING OR
  ON CERTAIN TYPES OF CALLS ONLY THE SEARCH AND INDICATION,
  OF   O N E   STEP.
  THE EXCEPTION IS WHEN THE PROCEDURE IS CALLED WITH REENTRY
  SET AT ZERO, I.E. IN THE NON-REENTRY MODE, AND THERE ARE
  VARIABLES OF TYPE ABSOLUTE, WITHOUT NON-NEGATIVIY RESTRICTION.
  PHASE 0 IS ALWAYS COMPLETED, EXCEPT IN THE CASE OF A SYSTEM
  OF INADEQUATE RANK. ONE MORE STEP AFTER PHASE ZERO IS M DE IN
  THAT CASE.
  ;

  'IF' REENTRY=2 'THEN' SEARCH ONLY := 'TRUE';

  'IF' 'NOT' K=0 'THEN' 'BEGIN'
    PRESCRIBED COLUMN:='TRUE'; FIRST:=LAST:=K; 'END';

  'GOTO' PHASE I; 'END';

FILL NAMELISTS:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' COLLST[J] := J;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' ROWLST[I] := 1000+I;

SET UPPER BOUNDS AND FILL DUMMY ENTRIES:
'FOR' J:=NAV+1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[M+2,J]=0
'THEN' T[M+2,J]:=1000000;
T[M+1,N+1]:=T[M+2,N+1]:=T[M+1,N+2]:=T[M+2,N+2]:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' T[I,N+2]=0 'AND' IROWLST[I]=0
'THEN' T[I,N+2]:=1000000000;
'FOR' J:=1 'STEP' 1 'UNTIL' NAV 'DO' T[M+2,J]:=0;

ATTEND PRIMAL DEGENERACY:
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,N+1]=0
'AND' IROWLST[I]=0 'THEN' 'BEGIN'
  T[I,N+1]:=1;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' T[I,J] < 0 'THEN'
  T[I,N+1]:=T[I,N+1]-T[I,J];
  T[I,N+1]:=0.00000000000000001*T[I,N+1];
  'IF' I < NEQ+1 'THEN' T[I,N+1]:=-T[I,N+1]; 'END';
```

```
ATTEND DUAL DEGENERACY:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' T[M+1,J]=0 'THEN' 'BEGIN'
  T[M+1,J]:=1;
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,J] > 0 'THEN'
  T[M+1,J]:=T[M+1,J]+T[I,J];
  T[M+1,J]:=0.000000000000001*T[M+1,J];
  'IF' J < NAV+1 'THEN' T[M+1,J]:=-T[M+1,J]; 'END';

'IF' 'NOT' K=0 'THEN' 'GOTO' PHASE I;

PHASE 0:
ENTER VARIABLES WITHOUT SIGN RESTRICTION:
INVERTED:='FALSE';

RETURN IN INVERSION:
K:=K+1;
'IF' K > NAV 'THEN' 'BEGIN'
  INVERTED:='TRUE'; 'GOTO' ORDER; 'END';

COLN:=K; QUO:=10000000000000000; ROWN:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
'IF' ROWLST[I] > NAV 'AND' 'NOT' T[I,K] = 0 'THEN' 'BEGIN'
  TQUO:=T[I,N+1]/T[I,K];
  'IF' TQUO < 0 'THEN' TQUO:=-TQUO;
  'IF' TQUO < QUO 'THEN' 'BEGIN'
    QUO:=TQUO; R:=I; ROWN:=ROWLST[I]; 'END'; 'END';
'IF' ROWN=0 'THEN' 'GOTO' UNBOUNDED;
QUO := T[R,N+1]/T[R,K]; 'GOTO' MAKE THE STEP;

ORDER:
ORDL(T,M,N,2,2,ROWLST,COLLST);

PHASE I:    FEASIBLE := 'TRUE';
R := PRESCRIBED ROW;
'IF' K#0 'AND' (PRESCRIBED ROW # 0 'OR' ABS(COLN-ROWN)=10000)
'THEN' 'GOTO' RESTORE CORRECT VALUE OF QUO;

CHECK ON EMPTYNESS BY CONTRADICTION:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' T[I,N+1] + T[I,N+2] < 0 'THEN' 'BEGIN'
  'IF' T[I,N+1]+T[I,N+2]>-0.0001 'THEN' 'BEGIN'
    'FOR' J:=1,2 'DO'
    'IF' T[I,N+J]<0 'AND' T[I,N+J]>-0.0001
    'THEN' T[I,N+J]:=0;
    T[I,N+2]:=-T[I,N+1]+0.000001; 'END'
  'ELSE' 'GOTO' EMPTY; 'END';

FIND WHETHER A FEASIBLE SOLUTION EXISTS:
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,N+1] < 0
'OR' T[I,N+2] < 0 'THEN' FEASIBLE:='FALSE';

MAXIMIZE:   HIG:=0;   COLN:=0;
'FOR' J:= FIRST 'STEP' 1 'UNTIL' LAST 'DO'
'IF' COLLST[J] < 1000 'OR' COLLST[J] > NEQ + 1000 'THEN' 'BEGIN'
```

```
INITIALIZE SUBSTITUTE PREFERENCE FUNCTION:
ASC:=0; UNIFORM :=0;

'IF' FEASIBLE 'THEN' ASC:=-T[M+1,J] 'ELSE'
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
  SAVED ASC := ASC;
  'IF' T[I,N+1] < 0 'THEN' ASC:=ASC-T[I,J];
  'IF' T[I,N+2] < 0 'THEN' ASC:=ASC+T[I,J];

  'IF' UNIFORM = 0 'AND' 'NOT' ASC = SAVED ASC
  'THEN' UNIFORM :=1;
  'IF' UNIFORM = 1 'THEN' 'BEGIN'
    'IF' T[I,N+1] < 0 'AND' T[I,J] > 0.000001
    'THEN' UNIFORM := -1;
    'IF' T[I,N+2] < 0 'AND' T[I,J] < -0.000001
    'THEN' UNIFORM := -1; 'END'; 'END';

REFUSE UNDERSIZE PHASE I PIVOTS:
'IF' 'NOT' FEASIBLE 'AND' ASC < 0.0000001
'THEN' 'GOTO' FINISHED WITH THIS COLUMN;

SELECT BETWEEN NON PREFERRED COL BY DUAL RATIO:
'IF' ('NOT' FEASIBLE 'AND'T[M+1,J] > 0 'AND' ASC > 0)
'THEN' ASC := 0.001*ASC*ASC/T[M+1,J];

'IF' 'NOT' ASC > 0 'THEN' 'GOTO' FINISHED WITH THIS COLUMN;

PUT PRFERENCE FOR UNIFORM COLUMNS:
'IF' UNIFORM = 1 'THEN' ASC := 10000000*ASC;

PUT PREFERENCE FOR PREFERRED COLUMNS:
'IF' 'NOT' FEASIBLE 'AND' T[M+1,J] < 0 'THEN'
ASC:=ASC-1000000000000*T[M+1,J];

SEARCH FOR QUO IN JTH COLUMN:
QUO:=PREV POS:=T[M+2,J];  TRYR:=0;
'IF' COLLST[J]<10000 'THEN' TRYN:=COLLST[J]+10000
'ELSE' TRYN:=COLLST[J]-10000;

'IF' UNIFORM =1 'THEN' ASC := ASC*1000;

'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'

  MEASURED := 'FALSE'; POSITIVE := 'FALSE';

  VIOLATED RESTR ON VALUE COL:

  EQN:
  'IF' ROWLST[I]>1000 'AND' ROWLST[I]<1001+NEQ
  'AND' T[I,J]<0 'AND' T[I,N+1]<0
  'THEN' 'BEGIN'
    'IF' T[I,N+1]>-0.0000001 'THEN' T[I,N+1]:=-0.0000001;
    POSITIVE := 'TRUE';
    'GOTO' CHECK QUOTIENT; 'END';

  'IF' T[I,J]<0 'AND' T[I,N+1]<0
  'THEN' 'GOTO' CHECK QUOTIENT;
```

```
VIOLATED UPPERBOUND DISTANCE:
'IF' T[I,J]>0 'AND' T[I,N+2]<0
'THEN' 'GOTO' TRY UPPER BOUND;

POSITIVE RATIO WITH UPPER LIMIT DISTANCE:
'IF' T[I,J] < 0 'AND' 'NOT' T[I,N+2] < 0
'THEN' 'GOTO' TRY UPPER BOUND;
POSITIVE RATIO WITH VALUE COLUMN:
'IF' T[I,J]>0 'AND' 'NOT' T[I,N+1]<0
'THEN' 'GOTO' CHECK QUOTIENT;
'GOTO' FINISHED WITH THIS PIVOT;

REMEASURE:
'IF' POSITIVE 'THEN' 'GOTO' FINISHED WITH THIS PIVOT;
MEASURED := 'TRUE';
VALUE COLUMN AFTER UPPER LIMIT:
'IF' T[I,J] > 0 'AND' 'NOT' T[I,N+1]<0
'THEN' 'GOTO' CHECK QUOTIENT;
UPPER LIMIT AFTER VALUE COL:
'IF' T[I,J] < 0 'AND' 'NOT' T[I,N+2]<0
'THEN' 'GOTO' TRY UPPERBOUND;
'GOTO' FINISHED WITH THIS PIVOT;

CHECK QUOTIENT:
UPPERBOUND:='FALSE'; TQUO:=T[I,N+1]/T[I,J];
'IF' TQUO=0 'THEN' TQUO:=0.000000000001/T[I,J];
'IF' 'NOT' T[I,N+1] < 0 'THEN' POSITIVE:='TRUE'
'ELSE' POSITIVE := 'FALSE';

MEASURE:

'IF' ('NOT' UPPERBOUND 'AND' T[I,J]<0 'AND' T[I,N+1]<0)
'OR' (UPPERBOUND 'AND' T[I,J]>0 'AND' T[I,N+2]<0)
'THEN' POSITIVE:='FALSE';

'IF' ROWLST[I]>1000 'AND' ROWLST[I]<1001+NEQ
'THEN' 'BEGIN'
  POSITIVE:='TRUE'; UPPERBOUND:='FALSE';
  'GOTO' RECORD; 'END';

IN PREFERRED COLUMNS ACCEPT ONLY POSITIVE:
'IF' T[M+1,J] < -0.000001 'AND' 'NOT' POSITIVE
'THEN' 'BEGIN'
  'IF' MEASURED 'THEN' 'GOTO' FINISHED WITH THIS PIVOT
  'ELSE' 'GOTO' REMEASURE; 'END';

ACCEPT POSITIVE AND ONE NEGATIVE:
'IF' POSITIVE 'AND' TQUO<QUO 'THEN' 'GOTO' RECORD;
'IF' 'NOT' POSITIVE 'AND' TQUO<PREV POS
'AND' QUO = PREV POS 'THEN' 'GOTO' RECORD;

DO NOT ACCEPT IF EXCEEDING EARLIER POSITIVE:
'IF' TQUO > PREV POS 'THEN' 'BEGIN'
  'IF' MEASURED 'THEN' 'GOTO' FINISHED WITH THIS PIVOT
  'ELSE' 'GOTO' REMEASURE; 'END';
```

```
DO NOT INCREASE IN NON PREFERRED DIRECTION:
'IF' POSITIVE 'AND' TQUO > QUO 'AND' T[M+1,J]>0
'THEN' 'BEGIN'
  'IF' MEASURED 'THEN' 'GOTO' FINISHED WITH THIS PIVOT
  'ELSE' 'GOTO' REMEASURE; 'END';

NORMALLY FLY THROUGH NON LIMIT CUTS:
'IF' REENTRY=1 'AND' TRYN>2*N 'AND' TRYN<3*N+1
'THEN' 'BEGIN'
  'IF' LAST>FIRST 'AND' 'NOT' POSITIVE 'THEN' 'BEGIN'
    'IF' 'NOT' MEASURED 'THEN' 'GOTO' REMEASURE 'ELSE'
    'GOTO' FINISHED WITH THIS PIVOT; 'END'; 'END';

DO NOT ACCEPT ALL NEGATIVE:
'IF' 'NOT' POSITIVE 'AND' 'NOT' QUO = PREV POS
'THEN' 'BEGIN'
  'IF' 'NOT' MEASURED 'THEN' 'GOTO' REMEASURE
  'ELSE' 'GOTO' FINISHED WITH THIS PIVOT; 'END';

'IF' ('NOT' UPPERBOUND 'AND' T[I,J]<0 'AND' T[I,N+1]<0)
'OR' (UPPERBOUND 'AND' T[I,J]>0 'AND' T[I,N+2]<0)
'THEN' POSITIVE:='FALSE' 'ELSE' POSITIVE:='TRUE';

RECORD:
'IF' POSITIVE 'AND'
TQUO*ASC<HIG 'THEN' 'GOTO' FINISHED WITH THIS COLUMN;

'IF' 'NOT' TQUO < PREV POS
'OR' ('NOT' POSITIVE 'AND' ABS(T[I,J])>T[M+2,J])
'OR' ('NOT' POSITIVE 'AND' 1/ABS(T[I,J])>T[M+2,J])
'OR' ('NOT' POSITIVE 'AND' T[M+1,J]<0)
'THEN' 'BEGIN'
  'IF' MEASURED 'THEN' 'GOTO' FINISHED WITH THIS PIVOT
  'ELSE' 'GOTO' REMEASURE; 'END';

TRY TO AVOID OUTSIZE PIVOTS:
'IF' (ABS(T[I,J]) < 0.1 'OR' ABS(T[I,J]) > 10)
'AND' 'NOT' (ROWLST[I]>1000 'AND' ROWLST[I]<1001+NEQ)
'THEN' 'BEGIN'
  'IF' 'NOT' POSITIVE
  'THEN' 'BEGIN'
    'IF' MEASURED 'THEN' 'GOTO' FINISHED WITH THIS PIVOT
    'ELSE' 'GOTO' REMEASURE; 'END'; 'END';

QUO:=TQUO; TRYR:=I; TRYN:=ROWLST[I];
'IF' UPPERBOUND 'THEN' TRYN:=TRYN+10000;

'IF' POSITIVE 'THEN' PREV POS:=QUO;

'IF' 'NOT' MEASURED 'THEN' 'GOTO' REMEASURE
'ELSE' 'GOTO' FINISHED WITH THIS PIVOT;
```

```
      TRY UPPER BOUND;
      UPPERBOUND := 'TRUE'; TQUO := -T[I,N+2]/T[I,J];
      'IF' TQUO=0 'THEN' TQUO:=-0.000000000001/T[I,J];
      'IF' 'NOT' T[I,N+2] < 0
      'THEN' POSITIVE := 'TRUE'
      'ELSE' POSITIVE := 'FALSE';
      'GOTO' MEASURE;

      FINISHED WITH THIS PIVOT: 'END';

    PIWEIGHT:=10;

    'IF' 'NOT' TRYR=0 'THEN' 'BEGIN'
      PIWEIGHT := ABS(T[TRYR,J]);
      'IF' PIWEIGHT>1 'THEN' PIWEIGHT:=1/PIWEIGHT;
      'IF' PIWEIGHT<0.1 'THEN' PIWEIGHT:=PIWEIGHT*PIWEIGHT
      *PIWEIGHT; 'END';

    'IF' QUO*ASC*PIWEIGHT > HIG 'THEN' 'BEGIN'
      HIG:=QUO*ASC*PIWEIGHT;
      R:=TRYR;
      ROWN:=TRYN; K:=J; COLN:=COLLST[J]; 'END';

    FINISHED WITH THIS COLUMN: 'END';


'IF' COLN = 0 'THEN' 'BEGIN'
  'IF' FEASIBLE 'THEN' 'GOTO' OUT 'ELSE' 'GOTO' EMPTY; 'END';

RESTORE CORRECT VALUE OF QUO:
'IF' ABS(ROWN-COLN)=10000
'THEN' R:=0;
'IF' R=0 'THEN' QUO:=T[M+2,K] 'ELSE' 'BEGIN'
  'IF' ROWLST[R]=ROWN 'THEN' QUO:=T[R,N+1]/T[R,K]
  'ELSE' QUO := -T[R,N+2]/T[R,K]; 'END';

MAKE THE STEP:
'IF' REENTRY=2 'THEN' 'GOTO' OUT;
COLLST[K]:=ROWN;
'FOR' I:=1 'STEP' 1 'UNTIL' R-1, R+1 'STEP' 1 'UNTIL' M+1 'DO'
'IF' 'NOT' T[I,K]=0
'THEN' 'BEGIN'
  PRODUCT(T[I,N+1],T[I,K],-QUO);
  PRODUCT(T[I,N+2],T[I,K],QUO); 'END';
'IF' ABS(ROWN-COLN)=10000 'THEN' 'BEGIN'
  'COMMENT'
  DIRECT HIT OF THE UPPER BOUND ON THE COLUMN-VARIABLE.
  NO FULL UPDATING OF THE TABLEAU IS REQUIRED;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO'
  'IF' T[I,K]≠0 'THEN' T[I,K]:=-T[I,K];
  'GOTO' CHECK FOR STATUS;  'END';

ATTEND UPPERBOUNDS OF PIVOTAL PAIR:
COP:=T[M+2,K]; NUM:=T[M+2,K]:=T[R,N+1]+T[R,N+2];
T[R,N+1] := QUO; T[R,N+2] := COP-QUO;
```

```
REFORMULATE ROW WITH UPPER BOUND NAME:
'IF' ROWN > 10000 'THEN'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' T[R,J]:=-T[R,J];

UPDATE:
PIV:=T[R,K];

UPDATE THE PIVOTAL ROW:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' 'NOT' T[R,J]=0 'THEN'
T[R,J]:=T[R,J]/PIV;

UPDATE THE OTHER ROWS:
'FOR' J:=1 'STEP' 1 'UNTIL' K-1,K+1 'STEP' 1 'UNTIL' N 'DO'
'IF' 'NOT' T[R,J] = 0 'THEN'
'FOR' I:=1 'STEP' 1 'UNTIL' R-1,R+1 'STEP' 1 'UNTIL' M+1 'DO'
'IF' 'NOT' T[I,K] =0 'THEN'
PRODUCT(T[I,J],-T[R,J],T[I,K]);

CLEAN AND UPDATE PIVOTAL COLUMN:
'FOR' I:=1 'STEP' 1 'UNTIL' M+1 'DO' 'BEGIN'
  'IF' I < M+1 'AND' ABS(T[I,K]) < 0.0000001 'THEN' T[I,K]:=0;
  'IF' 'NOT' T[I,K]=0 'THEN' T[I,K]:=-T[I,K]/PIV; 'END';
T[R,K]:=1/PIV;

ROWLST[R]:=COLN;

IF NECESSARY REFORMULATE NEW ROW WITH UPPERBOUND NAME:
'IF' COLN > 10000 'THEN' 'BEGIN'
  ROWLST[R]:=COLN-10000;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' 'NOT' T[R,J]=0
  'THEN' T[R,J]:=-T[R,J];
  COP:=T[R,N+1]; T[R,N+1]:=T[R,N+2]; T[R,N+2]:=COP; 'END';

CHECK FOR STATUS:
'IF' 'NOT' INVERTED 'THEN' 'GOTO' RETURN IN INVERSION;
'GOTO' OUT;


EMPTY:
REENTRY := -1;
'GOTO' END OF INTP;

UNBOUNDED:
REENTRY := 1;
NEWLINE(1);
WRITETEXT('('UNBOUNDED%COLUMN')');
PRINT(COLLST[J],7,0);
NEWLINE(1);
'GOTO' END OF INTP;


OUT:
REENTRY := 0;
'IF' FEASIBLE 'THEN' 'GOTO' END OF INTP;
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M 'DO'
'IF' (T[I,N+1] < 0 'OR' T[I,N+2] < 0)
'THEN' REENTRY := -10;

END OF INTP:  'END';
```

The following comments and elucidations seem appropriate in connection with the textual listing of INTP as given above.

INTP is an adaptation of the LINP procedure from section 12.3, and the textual differences are mainly the following.

a) The name itself, i.e.

LINP and END OF LINP: at the end

has been replaced by

INTP and END OF INTP:

b) Some additional parameters

The reasons for turning R, K, ROWN and COLN which are internally declared variables in LINP (and hence not accessible outside the procedure body), into parameters of the procedure were already discussed above.

There also is the integer array IROWLST, the list of integer restricted rownames. While the normal list of rownames is filled (by INTP as well as by LINP) on the first "normal" entry, IROWLST should be pre-filled before the first entry.

The last point relates to only one instruction in the programme, the settling of a "fancy-high" upper limit where a zero is supplied on first entry. This is because upper limits on integer restricted slack-variables have to be integer numbers, and we cannot exclude the possibility that the main mixed integer programming procedure would come up with the integer number zero. This would in effect turn the restriction into an equation, and such a zero should obviously be respected.

The technical side of the coding of integer-restricted slacks will be discussed in more detail in the next section.

c) Full inclusion of the upperbounds distances column in the "fancy-high" initial value loop, the anti-degeneracy-loop and the "Phase I" search operations. This latter point, combined with the "efficient" rule of row-selection, made it necessary to re-write the row-selection loop more or less entirely, and accounts easily for the longest stretch of amended text.

d) Removal of the "wrong sign" equation loop. Turning round the sign of a row representing an equation complicates the interpretation of the corresponding entry in the upper limits distances column. It was considered more simple to leave this point to the user. Thus, if an equation is offered with a positive constant

it may in effect stay an amply fulfilled restriction.

e) During testing, some further modifications were introduced in connection with the problem of numerical accuracy.

In particular, when as a result of cuts, an integer vertex is reached, one has to be sure that the solution vector is also found integer, within the specified tolerance, even where it is, due to rounding error, not exactly integer. The following points are to be mentioned in this connection.

(e1) Calculation in double length precision. Storage of numbers is in single length precision floating point, but the usual combination of multiplication and subtraction in the context of tableau-updating is done in double length precision. The significance of the internal procedure PRODUCT is simply to set the call to a library procedure which does the double length calculation.

(e2) The rule of the highest step is modified. Pivots of which the absolute value is close to unity are preferred over pivots which are either smaller or bigger in absolute value. This is not an absolute preference, rather the choice of pivots is loaded in that way. This is the significance of the real variable PIWEIGHT.

f) Preference for "uniform" columns

This point relates to Phase I, i.e. to tableaux in which negative elements occur either in the value column or in the upper limit distances column.

Uniform columns are those columns for which not only some, but all negative slacks become less negative or at least not more negative on entering the corresponding variable in the list of basic variables.

The reason why INTP prefers uniform columns to non-uniform columns is that additional violated restrictions (cuts) may be introduced between steps.

Strictly speaking we lack a convergence proof; since the LP problem is changed between steps.

But obviously, as long as all slack-variables are made less negative at each step, this problem does not arise - if it is a problem at all.

21.5 The coding of integer requirements and cuts

The following information needs to be coded

- whether an initially specified variable is continuous or integer restricted
- whether a subsequently developed cut has a continuous or an integer-restricted slack.

The first problem, to have a record of the type of variable initially arises as soon as the problem is specified at the outset.

We need to keep a record, not only of which variables are specified as continuous but also which slack-variables are integer-restricted by implication. It is here assumed that the initial information is given in the tableau with other "ordinary" linear programming information. A variable will hold to be integer restricted if that variable is

a) sign-restricted, i.e. restricted to be not below a lower limit
b) all coefficients in its associated column, including the preference coefficient and the upper limit are supplied integer-valued.

During later iterations these integer properties of the initial tableau will be lost, and we need to code them for later reference.

To this end we can duplicate the name-lists which are associated with the set-up tableau and replace the names in the duplicate list which refer to continuous variables by zeros.

Example

Name			x_1	x_2	$\leq$	Value	Distance
	code		1	2			
		i code	1	2			
s_1	1001	1001	8	6		15	100
s_2	1002	1002	2	-4		-1	100
			-3	-1		0	
Upper b			101	100		x	
Lower b			0	0		x	

for our demonstration example with both variables integer restricted. On the other hand, the initial tableau-matrix

8	6	0	15
2	-4	1	-1
-3	-1	10	0
1000	1000	1000.1	X

would imply that the third variable was continuous. As a result the second slack would be continuous, but in the first restriction there is a zero coefficient for x_3, and s_2 is still integer restricted.

The initial coding would therefore be

Name			x_1	x_2	x_3	$\leq$	Value	Upper b
	Code		1	2	3			
		i Code	1	2	0			
s_1	1001	1001	8	6	0		15	1000
s_2	1002	0	2	-4	1		-1	1000
τ			-3	-1	10		0	X
Upper b			1000	1001	1000.1		X	X
Lower b			0	0	0		X	X

The second problem, the recognition of previous cuts relates to augmentation and combination. We need to know whether these operations are appropriate i.e. whether the slack of a particular cut-restriction is integer restricted.

Cuts which are written as explicit restrictions, are therefore coded with an enlargement. For limit cuts this enlargement is equal to n, the number of specified variables, irrespective of whether the cut is on the variable itself or on a combination. The associated textual name is x^*_k. Hence, if x_5 is an integer restricted variable (k = 5) and there are 10 specified variables

(n= 10), x^*_5 is the integer-restricted slack of a cut on or in association with x_5, and this slack is coded as 15.

For cuts which do not have integer-restricted slacks, the corresponding enlargement is 2 n, e.g. 25 for x^{**}_5 if n is 10.

21.6 Summary of the Cutting Algorithm

We are now in a position to solve a problem by an algorithm which is substantially the one which we will actually implement, and summarise the algorithm in the process. The commented text in the next section contains some further refinements, but the summary in this section provides the main structure.

The problem which we take is similar to one which we used before, except for the sign of the x_3 preference coefficient.

Maximise

$$\tau = 3\,x_1 + x_2 + x_3$$

Subject to

$$8\,x_1 + 6\,x_2 + 3\,x_3 \leq 15$$

$$2\,x_1 - 4\,x_2 + x_3 \leq -1$$

$$(0 \leq x_1 \leq 100,\ 0 \leq x_2 \leq 100,\ 0 \leq x_3 \leq 100)$$

x_1, x_2 integer-restricted x_3 continuous.

Stage 1

Put the initial tableau and fill the integer-lists

Name			x_1	x_2	x_3	$\leq$	Value	
	code		to be filled by INTP					
		i code	1	2	0			
s_1	-	0	8	6	3		15	100
s_2	-	0	2	-4	1		-1	100
	X	X	-3	-1	-1		-	-
Upper b	X	X	100	100	99.99		X	X
Lower b			0	0	0		X	X

Stage 2

Make an initial step.

Attempt, and if indicated, make one step. The LP algorithm is entered with the re-entry parameters as zero, hence before the one step is made we fill the ordinary name-lists and enter any variables of "absolute" type. "Phase 0" of INTP is in this respect comparable with LINP. The one step is the following:

Name			x_1	x_2	x_3	$\leq$	Value	Distance
	Code		1	2	3			
		i code	1	2	0			
s_1	1001	0	8	(6)	3		15	100
s_2	1002	0	2	-4	1		-1	100
τ			-3	-1	-1		-	-
upper b			100	100	100		x	x
lower b			0	0	0		x	x

Name			x_1	s_1	x_3	$\leq$	Value	Distance
	code		1	1001	3			
		i code	1	0	0			
x_2	2	2	1 1/3	1/6	1/2		$2\frac{1}{2}$	$92\frac{1}{2}$
s_2	1002	0	7 1/3	2/3	3		9	90
τ			-1 2/3	1/6	$-\frac{1}{2}$		$2\frac{1}{2}$	
upper b			100	115	99.99		x	x
lower b			0	0	0		x	x

Stage 3

Check and limit

Check

Establish whether the current solution is:

a) in optimal form
b) feasible in relation to the already present restrictions
c) integer-valued, and
d) whether any new cuts are being made at this stage; these are of course unsatisfied, i.e. violated restrictions

Limit

Make preliminary(limit) cuts.

At this stage we make cuts of the following types, in that order of priority

1) Upper and lower limit cuts on specified variables. (See sections 10.3, 10.4 and 20.3 concerning the computational implementation of such restrictions, by putting a negative entry in one of the two value columns.)

2) Upper and lower limit cuts on integer-restricted slack-variables. This is also done by adjustment of one of the two value-columns, and no record is kept of the true values of slack-variables which may therefore be under-stated.

Since cuts of priority classes 1) and 2) do not require additional tableau-space, all possible cuts of this type are made.

3) Limit cuts on combinations

We reserve space for at most n+1 explicit cut-restrictions. This is one more than the maximally possible number of binding cuts, but we may come back at this stage for a number of successive steps, developing several cuts on the same variable. It is therefore possible that we shall have to refrain from further cutting simply because the space reservation has been exhausted.

We find that the solution is

a) not optimal
b) feasible in relation to the existing restrictions
c) not integer, and
d) not satisfying all cuts (we are making a new cut on x_2).

The x_2-variable is recorded as being $\frac{1}{2}$ unit <u>over</u> its new upper limit of 2.

The tableau becomes:

Name			x_1	s_1	x_3	$\leq$	Value	Distance
	code		1	1001	3			
		i code	1	0	0			
x_2	2	2	1 1/3	1/6	$\frac{1}{2}$		$2\frac{1}{2}$	$-\frac{1}{2}$
s_2	1002	0	7 1/3	2/3	3		9	90
τ			-1 2/3	1/6	$-\frac{1}{2}$		$2\frac{1}{2}$	
Upper b			100	115	99.99			
Lower b			0	0	0			

Stage 4

Attempt, and if indicated, make one step (The problem may be found empty at this stage, this obviously would terminate the algorithm). In view of the particular features of the amended LP algorithm coming into play at this point we will describe the rules for making this one step in detail. The substitute objective function is in the first instance obtained by (the summation of all violated restrictions), the violated b_2-row.

	x_1	s_1	x_3
b_2	-1 1/3	1/6	1/2
$\sum$	-1 1/3	-1/6	-1/2

The taking of the squares of these numbers is in this case irrelevant. The x_1 and x_3 columns are "preferred" columns because they indicate an increase in both the substitute objective function and the specified objective function.

In a "preferred" column, violated restrictions are disregarded and we exchange x_1 against s_2, on the indication of the rule of the highest step , applied to the specified preference function.

The tableau becomes

Name			s_2	s_1	x_3	$\leq$	Value	Distance
	code		1002	1001	3			
		icode	0	0	0			
x_2	2	2	-2/11	1/22	-1/22		19/22	1 3/22
x_1	1	1	3/22	1/11	9/22		1 5/22	98 17/22
τ			5/22	7/22	2/11		4 6/11	
Upper b			99	115	99.99			
Lower b			0	0	0			

Stage 5

Consult the logical findings from stage 3. If the problem was found to be: not optimal, or not feasible, or not satisfying a cut, return to stage 3.

We <u>are</u> referred back to Stage 3 on two separate indications

a) the problem was not optimal at that stage, and
d) a new cut was made (the upper limit cut on x_2).

Application of the check and limit procedure now indicates that the problem now is

a) in optimal form
b) feasible
c) not integer (both x_1 and x_2 are fractional)
d) not satisfying all cuts, because a new cut (a reduced upper limit on x_1) is introduced. The tableau becomes

Name			s_2	s_1	x_3	$\leq$	Value	Distance
	code		1002	1002	3			
		icode	0	0	0			
x_2	2	2	2/11	1/22	-1/22		14/22	1 3/22
x_1	1	1	3/22	1/11	9/22		1 5/22	-6/22
τ			5/22	7/22	2/22		4 6/11	
Upper b			99	115	99.99		x	x
Lower b			0	0	0		x	x

Stage 4 does indicate a step; x_3 becomes a basic variable and the upper limit on x_1 becomes binding.

This involves an activation of an upper limit facility of the LP algorithm which is slightly unusual and we will illustrate it.

First the x_1-row is replaced by the b_1-row, the tableau now becomes:

Name			s_2	s_1	x_3	$\leq$	Value	Distance
	code		1002	1001	3			
		icode	0	0	0			
x_2	2	2	-2/11	1/22	-1/22		19/22	1 3/22
b_1	10001	1	-3/22	-1/11	-9/22		-5/22	1 5/22
τ			5/22	7/22	2/22		4 6/11	
Upper b			99	115	99.99		x	x
Lower b			0	0	0		x	x

The step is now made in the normal way and the tableau becomes:

Name			s_2	s_1	b_1	$\leq$	Value	Distance
	code		1002	1001	10001			
		i code	0	0	0			
x_2	2	2	-1/6	1/18	-1/9		8/9	1 1/9
x_3	3	0	1/3	2/9	-2 4/9		5/9	99 4/9
τ			13/66	59/198	2/9		4 49/99	
Upper b			99	115	1			

Stage 5 once more indicates a referral to Stage 3, but this time solely on the indication that d) there was an unsatisfied cut when stage 3 was left the last time.

In stage 3 we now find that the solution is

a) optimal
b) feasible
c) not integer, and
d) satisfying all cuts

(x_3 is continuous and no limit cut on x_2 can be made). The re-entry of the LP algorithm in Stage 4 now is trivial - no step is indicated - In Stage 5, we are now no longer referred back to Stage 3.

Stage 6

Remove any amply fulfilled explicit cuts from the tableau. (None).

Stage 7

Check for normal end of the algorithm.

We only get at this stage, if the problem, as currently formulated, is optimal and feasible. Therefore, if the solution is integer, the normal exit is activated. For the present example, this is not yet the case, hence we proceed to the next stage.

Stage 8

Write a main cut on a still fractionally valued variable. As the priority classes of cuts have been exhausted, this is not

a limit cut. However, it is not necessarily as elementary cut. The tableau becomes the following:

Name			s_2	s_1	b_1		Value	Distance
	code		1002	1001	1001			
		icode	0	0				
x_2	2	2	-1/6	1/18	-1/9		8/9	1 1/9
x_3	3	0	1/3	2/9	-2 4/9		5/9	99 4/9
x^{**}_2	8	X	$-1\frac{1}{2}$	-1/16	-1		-1	100
τ			13/66	59/198	2/9		4 49/99	
Upper b			99	115	1		x	x
Lower b			0	0	0		x	x

Stage 9

Search for a pivot. We may enter this stage with or without a column-restriction. Initially we always enter without a column restriction, but at a later stage the choice of pivots may be restricted to a particular column. Try to find a hypothetical pivot in the main tableau. (The problem may be found empty - of integer solutions -) in which case the algorithm is terminated prematurely. We may also on re-entering this stage, find the problem empty, as far as pivots in a particular column is concerned. We re-enter this stage with column-restriction after making a subsidiary cut. If no pivot in that column may be found the column-restriction is to be removed. In fact a hypothetical pivot is indicated. The s_2-variable is to be introduced into the basis, making the x^{**}_2-cut binding.

Stage 10

Make a column-extract, according to the indicated column.

Name			s_2	$\leq$	Value	Distance
	code		1002			
		icode	0			
x_2	2	2	-1/6		8/9	1 1/9
x_3	3	0	1/3		5/9	99.44
x^{**}_2	8	X	$-1\frac{1}{2}$		-1	100
τ			13/66		4.49/49	
Upper b			99		x	x
Lower b			0		x	x

Stage 11

Make a step in the column extract (This is always possible when this stage is entered). The updated column-extract becomes

Name			x^{**}_2	$\leq$	Value	Distance
	code		8			
		icode	X			
x_2	2	2	-1/9		1	1
x_3	3	0	2/9		1/3	99.67
s_2	1002	0	-2/3		2/3	98 1/3
τ			13/99		4 36/49	
Upper b			99			
Lower b						

Stage 12

Go to a stage, as indicated by the updated column-extract and the index of the solution column. If the (hypothetical) solution is integer, or not feasible even after updating (while the column was chosen freely) go to Stage 3. If the hypothetical solution is integer, or not feasible even after updating, but the column was prescribed, remove the column-restriction, and go to stage 9. An integer-valued hypothetical vertex means that the cuts have been successful, as far as activating the particular column is concerned. If the solution cannot be made feasible by that column only, there is no purpose in developing a (further) subsidiary cut at least not in connection with this particular variable becoming a basic variable.

We are referred to stage 3, on the indication of an integer solution-vector (x_3 is continuous, i.e. is allowed to remain fractional).

In stage 3, we find that the main solution-vector is

a) in optimal form
b) not feasible
c) not integer
d) not satisfying a cut (the x^{**}_2-cut)

No priority cut is indicated
In stage 4, we make a step, which is the same as already indicated as hypothetical step in Stage 9. The tableau becomes:

Name			x^{**}_2	s_1	b_1	$\leq$	Value	Distance
	code		8	1001	1001			
		icode	0	0	1			
x_2	2	2	-1/9	1/16			1	1
x_3	3	0	2/5	5/24	-2 1/3		1⅓	99 2/3
s_2	1002	0	-2/3	1/24	2/3		2/3	98 1/3
τ			13/99	51/176	1/11		4 36/49	
Upper b			99	115	1		x	x
Lower b			0	0	0		x	x

In stage 5 we are once more referred back to stage 3, as on the previous exit of stage 3 the solution was not feasible and there was an unsatisfied cut. Back in stage 3, we find that the solution is

a) in optimal form
b) feasible
c) integer and
d) satisfying all cuts.

No new cuts are called for.

In stage 4 the entry of the LP algorithm is now trivial.

In stage 5, we are not referred back to stage 3.

In stage 6, we do not need to remove any cuts.

In stage 7, we conclude that the problem has been solved.

In this particular example, no stage beyond stage 12 was activated. We need at least two integer-restricted basic variables for that, not counting variables which are already at upper limits.

However, in the general case, there are more stages, i.e.

Stage 13

We only enter stage 13 with a fractional hypothetical vertex. Accordingly, find the name of the first integer-restricted, but fractionally valued variable in the hypothetical vertex, as updated in the column-extract. This is the variable on which a subsidiary cut is to be developed. We do not cut on the variable which has already been cut on in the same column-extract. The development of such a cut in a row-extract would require several steps; and it becomes more practical to refer back to the full tableau.

Stage 14

Make a row-extract. Depending on the name of the last subsidiary cut, this row-extract initially may contain one, or two ordinary rows from the main tableau. The row which was selected in stage 9 as pivotal row is always in the extract. If the selected cut relates to the variable selected in stage 9 (i.e. the one column in the unupdated form of the column extract), the row extract contains only one ordinary row. Otherwise find the row to which the subsidiary cut last selected in stage 13 refers. This is one of the rows of the main tableau in that case, and

it should be copied into the extract.

Stage 15

Implement the hypothetical step suggested in stage 9, in the row-extract.

Stage 16

Cut on the variable indicated as to be cut on.

Stage 17

Make the backward step in the row extract

Stage 18

Assemble the new subsidiary cut with the main tableau. If the subsidiary cut arose from entering the same column in the main tableau as was already used to form a previous subsidiary cut immediately before, overwrite the previous one; otherwise extend the tableau.

Go to stage 9.

As the various exits of the algorithm are already indicated in earlier stages this is the last stage.

21.7 Commented text of a mixed integer programming procedure

The programmed procedure offered in this section broadly follows the outline scheme discussed in Section 21.6.

The following points, which cropped up at the stage of programme-testing are useful to eludicate on.

First of all, recall Sections 19.2 and 19.3. It was there assumed that when the significance of a zero-one variable really is to indicate whether or not a certain relationship is at all applicable, a "fancyhigh" coefficient is written.

These "fancyhigh" numbers can, however, become a major problem when the method of cuts is employed to solve the problem, for the reasons indicated at the end of Section 21.4.

The first part of the interlude between stages 1 and 2 reduces such numbers to a figure which is (in absolute value) just one unit higher than what is needed to "set free" a particular relation.

The second part of the interlude will be more or less obvious.

Secondly, there are more indications for the generation of cuts than discussed in the previous section.

Most of them are controlled by the incoming variable intercept, and are largely designed to avoid certain steps altogether.

The following two points are mentioned here:

Cuts on incoming variables, creating degeneracy

If an integer-restricted incoming variable is provisionally indicated as a candidate for entering the basis (with a fractional value) we develop the hypothetical vertex and cut with the help of a row-extract, even if no main cut is so far present. It may then well happen that the unupdated form of such a cut is satisfied in degeneracy by the integer value zero. Therefore, subsequent searches for pivotal columns will give preference to other columns, if any are available. The rule of the highest step or the version of that rule which is implemented in INTP will try to avoid steps of zero length.

Pivot-size cuts

This refers to what is essentially a diversionary device. If a candidate-pivot of undesirable absolute value is thrown up, a further cut is made, simply to perturb the problem and to allow INTP to have a fresh go at perhaps finding a pivot of a more desirable order of magnitude.

```
'PROCEDURE' MIXI(T,M,N,NEQ,NAV,ROWLST,COLLST,NCUTS);
'VALUE' M; 'INTEGER' M,N,NEQ,NAV,NCUTS;
'ARRAY' T; 'INTEGER' 'ARRAY' ROWLST,COLLST;

'BEGIN'
  'INTEGER' I,J,R,K,ROWN,COLN,CUTN,NAME,REENTRY,II,JJ,MM,
  MMM,DNAV,KK,KKK,F,P,CW,RR,DNCUTS,LOOPLENGTH,NIRV,CUTR,
  SAVED R T,SAVED R E, CHECKI;
  'REAL' QUO,LV,NLV,NUM,COP,FANCYHIGH;
  'BOOLEAN' OPTIMAL,FEASIBLE,INTEGER,SATISFIED,POS,NEG,MAIN,
  RETURNED,RECALLING TO STAGES 3 AND 4,INTERCEPT,DEGENERATE,
  OVERWRITING FOR COMBINATION,PIVOT SIZE CUT,
  STEP ALREADY KNOWN;
  'INTEGER' 'ARRAY' IROWLST[1:M],ICL,ICOLLST[1:N],DLST[1:3],
  FDLST[1:M+N+N+3];
  'ARRAY' ROWEXTR[1:6,1:N+2],COLEXTR[1:M+N+N+4,1:3];

  'PROCEDURE' INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,
  IROWLST,R,K,ROWN,COLN,REENTRY);
  'ARRAY' T; 'INTEGER' M,N,NEQ,NAV,R,K,ROWN,COLN,REENTRY;
  'INTEGER' 'ARRAY' ROWLST,COLLST,IROWLST;
  'ALGOL';
```

```
'PROCEDURE' ORDL(T,M,N,ER,RH,ROWLST,COLLST);
'ARRAY' T; 'INTEGER' M,N,ER,RH;
'INTEGER' 'ARRAY' ROWLST,COLLST;
'ALGOL';


'COMMENT'
MIXED INTEGER PROGRAMMING BY CUTS.
THE TABLEAU SHOULD BE SUPPLIED IN THE USUAL LP-PRESENTATION,
WITH ONE MODIFICATION.
UPPER LIMITS ON EACH SUMMATION-EXPRESSION PER I-TH ROW
THE MAXIMUM VALUE OF SIGMA -A[I,J] * X[J],
SHOULD BE ENTERED IN THE UPPERBOUNDS DISTANCES COLUMN.

SIGN-RESTRICTED VARIABLES ( J > NAV ) ,
FOR WHICH EACH ELEMENT OF THE CORRESPONDING CULUMN OF THE
TABLEAU, INCLUDING PREFERENCE COEFFICIENT AND UPPER LIMIT,
IS SUPPLIED INTEGER, ARE INTERPRETED AS INTEGER-RESTRICTED,
OTHER SPECIFIED VARIABLES ARE ASSUMED TO BE CONTINUOUS.
ON EXIT, NCUTS ROWS WILL HAVE BEEN ADDED TO THE TABLEAU.

THE TABLEAU SHOULD BE DECLARED AS BEING OF ORDER M+N+NIRV+4
BY N+2.
HERE NIRV IS THE NUMBER OF INTEGER RESTRICTED VARIABLE.
THIS VARIABLE IS NOT DECLARED EXPLICITLY, BUT IS IMPLIED BY
THE NUMBER OF ALL-INTEGER COLUMNS, WHICH MAY GIVE RISE TO
THE DEVELOPMENT OF CUTS.

THE LIST OF ROWNAMES SHOULD BE DECLARED AS BEING OF
ORDER M+N+NIRV+2.

THE LIST OF COLUMN-NAMES SHOULD BE DECLARED AS BEING OF
ORDER N.
;

STEP ALREADY KNOWN:=RETURNED:='FALSE';

SAVE COPIES OF PARAMETERS:
MMM:=M;  DNAV:=NAV;

INITIATE DLST AND ROWEXTRACT WITH COPYABLE NONSENSE:
'FOR' I:=1,2,3 'DO' DLST[I]:=-12345;
'FOR' I:=1,2,3,4,5 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' ROWEXTR[I,J]:=123.45;

STAGE 1:
ATTEND UPPER LIMITS AND CODE INTEGER REQUIREMENTS:
'BEGIN' 'COMMENT'
  NOTE THAT THE TABLEAU SHOULD BE PRE-FILLED BEFORE ENTRY.
  ;

  FANCYHIGH:=100;

  'FOR' J:= NAV+1 'STEP' 1 'UNTIL' N 'DO'
  'IF' T[M+2,J] = 0 'THEN' T[M+2,J] := FANCYHIGH;
```

```
  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'IF' T[I,N+2]=0 'THEN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    'IF' T[I,J]=0 'OR' (J>NAV 'AND' T[I,J]>0)
    'THEN' 'GOTO' END OF UPPERBOUND ADJUSTMENT LOOP;
    'IF' J>NAV 'THEN' LV:=T[M+2,J] 'ELSE' LV:= FANCYHIGH;
    T[I,N+2]:=T[I,N+2]+ABS(T[I,J]*LV);
    END OF UPPERBOUND ADJUSTMENT LOOP: 'END';

  CODE INTEGER REQUIREMENTS:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
    ICOLLST[J]:=J;
    'IF' J< NAV+1 'THEN' 'BEGIN'
      'COMMENT'
      FREE VARIABLES ARE NEVER INTEGER RESTRICTED;
      ICOLLST[J]:=0; 'END'
    'ELSE' 'FOR' I:=1 'STEP' 1 'UNTIL' M+2 'DO'
    'IF' 'NOT' T[I,J]=ENTIER(T[I,J]) 'THEN' ICOLLST[J]:=0;
    'END';

  'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    IROWLST[I]:=1000+I;
    'IF' I < NEQ+1 'THEN' 'BEGIN'
      'COMMENT'
      EQUATION-SLACKS ARE INTEGER RESTRICTED TO THE
      INTEGER VALUE OF ZERO; 'END'
    'ELSE' 'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
    'IF' 'NOT' T[I,J]=0 'AND' ICOLLST[J]=0
    'THEN' IROWLST[I]:=0;
    'END';

  NIRV:=0;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' ICOLLST[J]=J 'THEN' NIRV:=NIRV+1;
  NCUTS:=0;
  END OF STAGE 1: 'END';

INTERLUDE:

REDUCE FANCYHIGH COEFFICIENTS:

'FOR' R:=1 'STEP' 1 'UNTIL' M 'DO'
'FOR' K:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' ICOLLST[K]=K 'AND' T[R,K] < -2 'AND' T[M+2,K]=1
'AND' T[M+1,K]=0
'THEN' 'BEGIN'

  'FOR' J:=1 'STEP' 1 'UNTIL' NAV 'DO'
  'IF' T[R,J]#0 'THEN' 'GOTO' END OF COEFFICIENT REDUCTION;

  T[M+3,N+1]:=T[R,N+1];

  'FOR' J:=1 'STEP' 1 'UNTIL' K-1,K+1 'STEP' 1 'UNTIL' N 'DO'
  'IF' T[R,J]>0
  'THEN' T[M+3,N+1,J]:=T[M+3,N+1]-T[R,J]*T[M+2,J];

  'IF' T[R,K]<T[M+3,N+1]-1
  'THEN' 'BEGIN'
    T[R,K]:=T[M+3,N+1]-1;
    'IF' T[R,K]>0 'THEN' T[R,K]:=0;
    'GOTO' REDUCE FANCYHIGH COEFFICIENTS; 'END';
```

```
  END OF COEFFICIENT REDUCTION: 'END';


TAKE OUT COMMON INTEGER FACTORS:
'FOR' F:=2 'STEP' 1 'UNTIL' 1000 'DO' 'BEGIN'
FIND A PRIME FACTOR F:
  'FOR' P:=2 'STEP' 1 'UNTIL' F-1 'DO'
  'IF' 'NOT' F/P = ENTIER(F/P) 'THEN' 'GOTO' NEXT F;

  'FOR' I:= NEQ+1 'STEP' 1 'UNTIL' M 'DO' 'BEGIN'
    'FOR' J:=NAV+1 'STEP' 1 'UNTIL' N 'DO'
    'IF' 'NOT' T[I,J]=0 'AND' 'NOT' ICOLLST[J]=0
    'THEN' 'GOTO' START OF DIVISIONLOOP;
    'GOTO' END OF DIVISIONLOOP;

    START OF DIVISIONLOOP:
    'FOR' J:= NAV+1 'STEP' 1 'UNTIL' N 'DO'
    'IF' 'NOT' T[I,J]/F=ENTIER(T[I,J]/F)
    'AND' 'NOT' ICOLLST[J]=0
    'THEN' 'GOTO' END OF DIVISIONLOOP;

    NOW TAKE F OUT:
    'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' T[I,J]:=T[I,J]/F;
    'IF' 'NOT' IROWLST[I] = 0 'THEN' 'FOR' J:=N+1,N+2 'DO'
    T[I,J]:=ENTIER(T[I,J]);
    'GOTO' START OF DIVISIONLOOP;
    END OF DIVISIONLOOP: 'END';
  NEXT F: 'END';


STAGE 2:
MAKE AN INITIAL STEP:
'BEGIN' 'COMMENT'
  THE INITIAL CALL TO THE LP ALGORITHM FILLS THE ORDINARY
  NAMELISTS, ATTENDS DEGENERACY -BOTH PRIMAL AND DUAL-,
  PIVOTS THE 'ABSOLUTE' VARIABLES INTO THE BASIS,
  AND MAKES, IF POSSIBLE, ONE NORMAL STEP.
  ;
  R:=K:=ROWN:=COLN:=REENTRY:=0;
  INTP(T,M,N,NEQ,NAV,ROWLST,COLLST,IROWLST,
  R,K,ROWN,COLN,REENTRY);
  'IF' REENTRY = -1 'THEN' 'GOTO' EMPTY;
  'IF' REENTRY = 1 'THEN' 'GOTO' UNBOUNDED;
  END OF STAGE 2: 'END';


STAGE 3:
MAIN := 'TRUE'; CUTN := 0;
CHECK AND LIMIT:
'BEGIN' 'COMMENT'

  THIS BLOCK OF PROGRAMME CHECKS WHETHER THE CURRENT SOLUTION
  IS:

  A     IN OPTIMAL FORM
```

```
B    FEASIBLE WITH RESPECT TO THE SO FAR SPECIFIED
RESTRICTIONS

C    INTEGER TO A TOLERANCE OF 0.001

D    SATISFIED.

THE SOLUTION IS HELD TO BE NOT SATISFIED WITH RESPECT TO
CUTTING REQUIREMENTS,
IF  N E W  CUTS ARE BEING FORMULATED BY THIS BLOCK.

THE FOLLOWING PRIORITY CLASSES OF CUTS ARE FORMULATED BY THE
CHECK AND LIMIT  BLOCK

1   UPPER AND LOWER LIMIT CUTS ON INTEGER RESTRICTED
SPECIFIED VARIABLES,
BY WAY OF ADJUSTING THE UPPERBOUNDS DISTANCES COLUMN
OR THE VALUE COLUMN AND THE NON-UPDATED LOWER LIMIT.


2   UPPER OR LOWER LIMIT CUTS ON OTHER INTEGER RESTRICTED
VARIABLES, E.G. INTEGER-RESTRICTED SLACK-VARIBALES
OR CUTS ON COMBINATIONS OF VARIABLES.
ALSO BY WAY OF ADJUSTING THE UPPERBOUNDS DISTANCES COLUMN.

3   LIMIT CUTS ON COMBINATIONS,
ASSOCIATED WITH SPECIFIED VARIABLES.

THE LAST CLASS OF CUTS HAS THE LOWEST PRIORITY AND
IS MADE ONLY WHEN NO CUT OF A HIGHER PRIORITY CLASS HAS BEEN
MADE IN THE PARTICULAR RUN THROUGH THE BLOCK SO FAR.
AN AMPLY FULFILLED EXPLICIT CUT MAY BE OVERWRITTEN BY A LATER
CUT ON THE SAME VARIABLE -OR ON ITS CUT-.
;

CLEAN OUT ROUNDING ERRORS:
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' ABS(T[I,J]) < 0.0000001 'THEN' T[I,J]:=0;

CHECK FOR OPTIMALITY:
OPTIMAL := 'TRUE';
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
'AND' T[M+NCUTS+1,J] < 0 'THEN' OPTIMAL := 'FALSE';

CHECK FOR BEING FEASIBLE:
FEASIBLE := 'TRUE';
'FOR' I:=NAV+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' T[I,N+1] < 0  'OR'  T[I,N+2] < 0
'THEN' FEASIBLE := 'FALSE';
```

```
CHECK FOR BEING INTEGER AND LIMIT:
INTEGER := SATISFIED := 'TRUE';
LOOPLENGTH := M+NCUTS;

'FOR' I:= M+NCUTS+2, NAV+1 'STEP' 1 'UNTIL' LOOPLENGTH
'DO' 'BEGIN'

  'IF' I=M+NCUTS+2 'THEN' 'GOTO' BEGIN OF COMPARING;

  DO NOT CUT WITH EQUATIONS:
  'IF' ROWLST[I]>1000 'AND' ROWLST[I]<1000+NEQ
  'THEN' 'GOTO' END OF LIMITLOOP;

  'IF' ROWLST[I] > N 'THEN' 'GOTO' SLACK OR CUT;
  'IF' ICOLLST[ROWLST[I]] = 0
  'THEN' 'GOTO' END OF LIMITLOOP;

  ONE MAIN LIMIT CUT ON ONE VARIABLE IS ENOUGH:
  'IF' MAIN 'AND' ROWLST[I]<N+1 'THEN' 'BEGIN'
    'FOR' II:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
    'IF' ROWLST[II]=ROWLST[I]+N
    'AND' (T[II,N+1] < 0 'OR' T[II,N+2] < 0)
    'THEN' 'GOTO' END OF LIMITLOOP; 'END';


  SLACK OR CUT:

  DO NOT CUT ON NON LIMIT CUTS:
  'IF' ROWLST[I] > 2*N 'AND' ROWLST[I] < 1001
  'THEN' 'GOTO' END OF LIMITLOOP;

  DO NOT CUT ON AN UNSATISFIED LIMIT CUT:
  'IF' ROWLST[I]>N 'AND' ROWLST[I]<2*N+1
  'AND' (T[I,N+1]<0 'OR' T[I,N+2]<0)
  'THEN' 'GOTO' END OF LIMITLOOP;

  'IF' ROWLST[I] < 1001
  'THEN' 'GOTO' START INTEGERCHECK CUM LIMIT;
  'IF' IROWLST[ROWLST[I]-1000] = 0
  'THEN' 'GOTO' END OF LIMITLOOP;

  START INTEGERCHECK CUM LIMIT:
  'IF' ABS(ENTIER(T[I,N+1]+0.001) - T[I,N+1]) < 0.002
  'AND' MAIN 'THEN' 'BEGIN'
    'FOR' J:=1,2 'DO' T[I,N+J]:=ENTIER(T[I,N+J]+0.003);
    'FOR' J:=1,2 'DO' 'IF' T[I,N+J]=0
    'THEN' T[I,N+J]:=0.0000000001;
    'GOTO' END OF LIMITLOOP; 'END';
  'IF' ROWLST[I] < N+1 'THEN' INTEGER := 'FALSE';
  'IF' T[I,N+1] < 0 'OR' T[I,N+2] < 0
  'THEN' 'GOTO' END OF LIMITLOOP;

  ONE CANNOT ADJUST ON A NON BASIC VARIABLE:
  'IF' 'NOT' MAIN 'AND' I=1 'THEN' 'GOTO' END OF LL;

  CHECK WHETHER UPPER LIMIT IS POSSIBLE:
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
  'AND' T[I,J] < 0 'THEN' 'GOTO' END OF UL;
```

```
UPPER LIMIT:
'IF' T[I,N+2] > T[I,N+1]
'THEN' T[I,N+2] := ENTIER(T[I,N+1]) + 0.0000001 - T[I,N+1]
'ELSE' T[I,N+2] := T[I,N+2] - ENTIER(T[I,N+2]) -0.9999999;
SATISFIED := 'FALSE';
'GOTO' END OF LIMITLOOP;

END OF UL:

CHECK WHETHER LOWER LIMIT IS POSSIBLE:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
'AND' T[I,J] > 0 'THEN' 'GOTO' END OF LL;

LOWER LIMIT:
'IF' ROWLST[I] < N+1 'THEN' 'BEGIN'
  ADJUST NON UPDATED FORM OF LOWER LIMIT:
  T[M+NCUTS+3,ROWLST[I]] :=
  T[M+NCUTS+3,ROWLST[I]] + ENTIER(T[I,N+1]+1); 'END';
'IF' T[I,N+1] < T[I,N+2]
'THEN' T[I,N+1] := T[I,N+1] - ENTIER(T[I,N+1]) -0.9999999
'ELSE' T[I,N+2] := ENTIER(T[I,N+2]) + 0.0000001 - T[I,N+1];
SATISFIED := 'FALSE';
'GOTO' END OF LIMITLOOP;

END OF LL:

'IF' ROWLST[I] > 2*N 'THEN' 'GOTO' END OF LIMITLOOP;

BEGIN OF LOWER LIMIT LOOP:
'IF' ROWLST[I] < N+1 'THEN' 'BEGIN'
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
  'AND' T[I,J] > 0 'THEN' 'GOTO' BEGIN OF COMBLOOP;
  'GOTO' PREPARE LL;  'END';

DO NOT ADJUST ON LOWER LIMIT NON BASIC VAR:
'IF' 'NOT' MAIN 'AND' I=1 'THEN' 'GOTO' END OF LIMITLOOP;

INVESTIGATE LOWER LIMIT ON CUTSLACK:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' 'NOT' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
'AND' T[I,J] > 0 'THEN' 'GOTO' BEGIN OF COMBLOOP;

WRITE LL ON CUT BY ADJUSTING VALUE COL:
'IF' T[I,N+1] > T[I,N+2]
'THEN' T[I,N+1]:=T[I,N+1] - ENTIER(T[I,N+1]) - 0.9999999
'ELSE' T[I,N+1] := ENTIER(T[I,N+2]) + 0.0000001 - T[I,N+2];
SATISFIED := 'FALSE';
'GOTO' END OF LIMITLOOP;

PREPARE LL:
'IF' ROWLST[I] > N 'THEN' 'BEGIN'
  R:=I; 'GOTO' REWRITE L CUT; 'END';
NAME := ROWLST[I];
```

```
'IF' 'NOT' MAIN 'THEN' 'GOTO' NO SKIPPING;

OVERWRITE OUTSTANDING NON LIMIT CUT:
OVERWRITE SATISFIED LIMIT CUT:
'FOR' II:=NAV+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' ROWLST[II]=NAME+2*N
'OR' (ROWLST[II]=NAME+N 'AND'
T[II,N+1] > 0 'AND' T[II,N+2] > 0) 'THEN' 'BEGIN'
  R:=II;  'GOTO' WRITE LL CUT; 'END';

NO SKIPPING:
'IF' NCUTS>N+NIRV 'THEN' 'GOTO' END OF LIMITLOOP;
'FOR' II:= 3,2 'DO' 'FOR' JJ:=1 'STEP' 1 'UNTIL' N+2 'DO'
T[M+NCUTS+II,JJ]:=T[M+NCUTS+II-1,JJ];
NCUTS:=NCUTS+1;  R:=M+NCUTS;

REWRITE LCUT:
WRITE LL CUT:
'IF' ROWLST[I]<N+1 'THEN' ROWLST[R] := ROWLST[I]+N;
T[R,N+2]:=T[I,N+2];
'IF' T[R,N+2] > T[I,N+1]
'THEN' T[R,N+1]:=T[I,N+1] - 0.9999999 - ENTIER(T[I,N+1])
'ELSE' T[R,N+1] := ENTIER(T[I,N+2]) +0.0000001 - T[I,N+2];
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  'IF' T[I,J]=0
  'OR' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
  'THEN' 'BEGIN'
    T[R,J]:=0; 'GOTO' LCUT COEFF WRITTEN; 'END';
  T[R,J]:=T[I,J];
  'IF' T[I,J] - ENTIER(T[I,J]) < 0.0000001
  'THEN' T[R,J]:=ENTIER(T[I,J]);
  'IF' T[I,J] - ENTIER(T[I,J]) > 0.9999999
  'THEN' T[R,J] := 1 + ENTIER(T[I,J]);
  'IF' ABS(T[R,J]-T[I,J]) < 0.0000001
  'THEN' T[I,J]:=T[R,J];

  'IF' ICL[J]=0 'OR' ABS(T[I,J]) < 1 'THEN' T[R,J]:=T[I,J]
  'ELSE' 'BEGIN' T[R,J]:=T[I,J]-ENTIER(T[I,J]);
    'IF' T[R,J] < 0 'THEN' T[R,J]:= 1 + T[R,J];
    T[R,N+2] := T[R,N+2]+ABS(T[R,J]-T[I,J])*T[M+NCUTS+2,J];
    'END';
  LCUT COEFF WRITTEN: 'END';
SATISFIED := 'FALSE';
'GOTO' END OF LIMITLOOP;

BEGIN OF COMBLOOP:
'IF' ROWLST[I] > N 'THEN' 'BEGIN'
  R:=I; 'GOTO' BEGIN OF COMPARING; 'END';
'IF' NCUTS=N+1 'THEN' 'GOTO' END OF LIMITLOOP;
'IF' 'NOT' SATISFIED 'THEN' 'BEGIN'
  'FOR' II:=NAV+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  'IF' ROWLST[II]=ROWLST[I]+N
  'THEN' 'GOTO' END OF LIMITLOOP; 'END';

BEGIN OF COMPARING:
POS:=NEG:= 'TRUE';
```

```
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'

  NAME := 0;
  'IF' COLLST[J] < N+1 'THEN' NAME := COLLST[J];
  'IF' COLLST[J] > 10000 'AND' COLLST[J] < 10001 +N
  'THEN' NAME := COLLST[J] - 10000;

  'IF' 'NOT' NAME=0 'THEN' 'BEGIN'
    'IF' 'NOT' ICOLLST[NAME]=0 'THEN' 'GOTO' RESTRICT
    'ELSE' 'GOTO' CHECK SIGN; 'END';

  'IF' COLLST[J] > 1000 'AND' COLLST[J] < 1001+M
  'THEN' NAME:=COLLST[J] - 1000;
  'IF' COLLST[J] > 11000 'AND' COLLST[J] < 11001 + M
  'THEN' NAME := COLLST[J] - 11000;

  'IF' 'NOT' NAME=0 'THEN' 'BEGIN'
    'IF' 'NOT' IROWLST[NAME]=0 'THEN' 'GOTO' RESTRICT
    'ELSE' 'GOTO' CHECK SIGN; 'END';
  'IF' COLLST[J] < 2*N+1
  'OR' (COLLST[J]>10000+N 'AND' COLLST[J]<10001+2*N)
  'THEN' 'GOTO' RESTRICT;

  CHECK SIGN:  ICL[J]:=0;
  'IF' T[I,J] < 0 'THEN' POS:='FALSE' ;
  'IF' T[I,J] > 0 'THEN' NEG:='FALSE' ;
  'IF' 'NOT' (POS 'OR' NEG) 'THEN' 'GOTO' END OF LIMITLOOP;
  'GOTO' END OF SIGN CHECK;
  RESTRICT:  ICL[J]:=COLLST[J];
  IF NOT INTEGER CLEAN OFF ROUNDING ERROR:
  T[M+NCUTS+2,J]:=ENTIER(T[M+NCUTS+2,J]+0.5);
  END OF SIGN CHECK:  'END';

'IF' I=M+NCUTS+2 'THEN' 'GOTO' END OF LIMITLOOP;

'IF' (POS 'OR' NEG) 'THEN' 'BEGIN'

  IF APPROPRIATE OVERWRITE FOR COMBINED CUT:

  OVERWITE SATISFIED LIMITCUT BY ITS OWN CUT:
  'IF' ROWLST[I]>N 'THEN' 'BEGIN'
    R:=I; 'GOTO' WRITE A COMBINED CUT; 'END';

  LOOK FOR AN OLD CUT TO OVERWRITE:
  OVERWRITING FOR COMBINATION := 'FALSE';
  'FOR' II:=NAV+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  'IF' ROWLST[II]=NAME+2*N
  'OR' (ROWLST[II]=NAME+N 'AND'
  T[II,N+1] > 0 'AND' T[II,N+2] > 0) 'THEN' 'BEGIN'
    OVERWRITING FOR COMBINATION := 'TRUE';
    R:=II;  'GOTO' WRITE A COMBINED CUT; 'END';

  CHECK SPACE FOR COMBINED CUT:
  'IF' MAIN 'AND' NCUTS>N+NIRV
  'THEN' 'GOTO' END OF LIMITLOOP;
  OVERWRITING FOR COMBINATION := 'FALSE';
```

```
      WRITE A COMBINED CUT:
      'IF' NEG 'THEN' POS:='FALSE';

      'IF' ROWLST[I] > N
      'OR' OVERWRITING FOR COMBINATION 'THEN' 'GOTO' WRITE;

      ENLARGE:
      'FOR' II:= 3,2 'DO'
      'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
      T[M+NCUTS+II,J]:=T[M+NCUTS+II-1,J];
      NCUTS:=NCUTS+1;   R:=M+NCUTS;

      WRITE:
      'IF' ROWLST[I]<N+1 'THEN' ROWLST[R]:=ROWLST[I]+N;
      'FOR' J := N+1,N+2 'DO' T[R,J] := T[I,J];
      'IF' POS 'THEN' JJ:=N+1 'ELSE' JJ:=N+2;
      'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
        T[R,J]:=NUM:=T[I,J]; 'IF' NUM < 0 'THEN' NUM:=-NUM;
        CW:=ENTIER(NUM);
        'IF' 'NOT' NUM=0 'AND' 'NOT' ICL[J]=0 'THEN' 'BEGIN'
          'IF' NUM - CW > 0.9999999 'THEN' CW := 1+CW;
          'IF' ABS(NUM-CW) < 0.0000001 'THEN' NUM:=CW;
          'IF' T[I,J] > 0 'THEN' T[I,J] := NUM
          'ELSE' T[I,J] := -NUM;
          'IF' (POS 'AND' T[I,J] < 0)
          'OR' (NEG 'AND' T[I,J] > 0)
          'THEN' CW:=CW+1;
          'IF' T[I,J] < 0 'THEN' NUM := -NUM;
          'IF' T[I,J] < 0 'THEN' CW := -CW;
          T[R,JJ]:=T[R,JJ]+T[M+NCUTS+2,J]*ABS(CW);
          NUM:=NUM-CW; T[R,J]:=NUM; 'END'; 'END';

      'IF' POS 'THEN' JJ:=N+2 'ELSE' JJ:=N+1;
      T[R,JJ] := T[R,JJ] - ENTIER(T[R,JJ]) -0.999999;
      SATISFIED := 'FALSE';
      END OF COMBINED CUT WRITING:  'END';

    END OF LIMITLOOP: 'END';

  END OF INTEGERCHECK CUM LIMIT:

  'IF' 'NOT' MAIN 'THEN' 'GOTO' STAGE 16;

  RECALLING TO STAGES 3 AND 4 := 'FALSE';

  END OF STAGE 3: 'END';


INCOMING VARIABLE INTERCEPT:

CORRECT DEGENERATE INFEASIBILITY:
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' 'NOT' (ROWLST[I]>1000 'AND' ROWLST[I]<1001+NEQ) 'THEN'
'FOR' J:=1,2 'DO'
'IF' T[I,N+J] > -0.0001 'AND' T[I,N+J] < 0.0000000001
'THEN' T[I,N+J]:=0.0000000001;
```

```
'IF' NCUTS > N+NIRV-1 'THEN' 'GOTO' STAGE 4;

R:=K:=ROWN:=COLN:=0;
REENTRY:=2;
INTP(T,M+NCUTS,N,NEQ,NAV,ROWLST,COLLST,IROWLST,
R,K,ROWN,COLN,REENTRY);

INCOMING VARIABLE INTERCEPT REENTRY:
PIVOT SIZE CUT := 'FALSE';

'IF' COLN=0 'OR' R=0 'THEN' 'GOTO' STAGE 4;

'IF' COLN<3*N+1 'OR' (COLN>10000 'AND' COLN<10001+2*N)
'OR' ABS(T[R,K])>10 'OR' ABS(T[R,K])<0.1
'THEN' 'BEGIN'
  'IF'
  (ICL[K]=COLLST[K] 'AND' T[M+NCUTS+2,K]<1.001)
  'OR' (COLN>N 'AND' COLN <3*N+1)
  'OR' ABS(T[R,K])>10 'OR' ABS(T[R,K])<0.1
  'THEN' 'BEGIN'

    'IF' (ABS(T[R,K])<0.1 'OR' ABS(T[R,K])>10)
    'AND' 'NOT' NCUTS > N+NIRV
    'THEN' 'BEGIN'
      RECALLING TO STAGES 3 AND 4 := 'TRUE';
      PIVOT SIZE CUT := INTERCEPT := 'TRUE';
      'GOTO' STAGE 8;  'END';

    'IF' ROWN=ROWLST[R] 'THEN' QUO:=T[R,N+1]/T[R,K]
    'ELSE' QUO:=-T[R,N+2]/T[R,K];
    'IF' QUO<0.001 'OR' QUO>T[M+2,K]-0.001
     'THEN' 'GOTO' STAGE 4;

    SET FOR AUXILIARY CUT:
    KK:=K; RECALLING TO STAGES 3 AND 4 := 'TRUE';
    INTERCEPT := 'TRUE';
    'GOTO' STAGE 10; 'END'; 'END';

STAGE 4:

ATTEND DEGENERACY:
DEGENERATE := 'FALSE';
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'FOR' J := 1,2 'DO'
'IF' ABS(T[I,N+J])<0.0000000001 'THEN' 'BEGIN'
  DEGENERATE := 'TRUE'; T[I,N+J]:=0.0000000001;
  'IF' ROWLST[I]>1000 'AND' ROWLST[I]<1001+NEQ
  'THEN' T[I,N+J]:=-T[I,N+J]; 'END';

'IF' DEGENERATE 'THEN' 'GOTO' INCOMING VARIABLE INTERCEPT;

TRY A STEP:
'IF' STEP ALREADY KNOWN 'THEN' 'GOTO'
KNOWN STEP REENTRY OF STAGE 4;
R:=K:=ROWN:=COLN:=0;
```

```
KNOWN STEP REENTRY OF STAGE 4:
STEP ALREADY KNOWN:='FALSE';
REENTRY:=1;
INTP(T,M+NCUTS,N,NEQ,NAV,ROWLST,COLLST,
IROWLST,R,K,ROWN,COLN,REENTRY);
'IF' REENTRY = -1 'THEN' 'GOTO' EMPTY;

'IF' 'NOT' MAIN 'AND' REENTRY = -10
'THEN' 'GOTO' IMPLEMENTATION REENTRY OF STAGE 9;

CUTSHARPER:
'IF' ROWN>N 'AND' ROWN<3*N+1 'THEN' 'BEGIN'
  'IF' ROWN>2*N 'THEN' NAME:=ROWN-2*N
  'ELSE' NAME:=ROWN-N;
  'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  'IF' ROWLST[I]=NAME 'AND'
  ABS(ENTIER(T[I,N+1]+0.001)-T[I,N+1])<0.002 'THEN' 'BEGIN'
    T[I,N+1]:=ENTIER(T[I,N+1]+0.002);
    T[I,N+2]:=ENTIER(T[I,N+2]+0.5);
    'IF' T[I,N+1]=0 'THEN' T[I,N+1]:=0.0000000001;
    'IF' T[I,N+2]=0 'THEN' T[I,N+2]:=0.0000000001; 'END';
  'END';

'IF' RETURNED 'THEN' 'BEGIN'
  RETURNED := 'FALSE'; 'GOTO' STAGE 3; 'END';

STAGE 5:

MAKE LIMITCUTS BINDING WITH PRIORITY:
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' ROWLST[I]>N 'AND' ROWLST[I]<2*N+1
'AND' (T[I,N+1]<0 'OR' T[I,N+2]<0)
'THEN' 'GOTO' STAGE 3;

LOOK FOR SUPPLEMENTARY CUTS ON NON LIMIT CUTS:
'IF' NCUTS<NIRV 'THEN' 'BEGIN'
  'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  'IF' ROWLST[I]>2*N 'AND' ROWLST[I]<3*N+1
  'AND' (T[I,N+1]<0 'OR' T[I,N+2]<0)
  'THEN' 'GOTO' STAGE 9; 'END';

REACT ON CHECK AND LIMIT:
'IF' 'NOT' FEASIBLE 'OR' 'NOT' SATISFIED
'OR' 'NOT' OPTIMAL
'THEN' 'GOTO' STAGE 3;

'IF' OPTIMAL 'AND' FEASIBLE 'AND' INTEGER
'THEN' 'GOTO' END OF MIXI;

STAGE 6:
REMOVE AMPLY FULFILLED CUTS:

'FOR' II:=NAV+1 'STEP' 1 'UNTIL' M+N+NIRV+1+1 'DO' 'BEGIN'

  'IF' II > M+NCUTS
  'THEN' 'GOTO' TRY IF MORE SPACE WILL DO IT;
```

```
  'IF' ROWLST[II] < N+1 'OR' ROWLST[II] > 3*N
  'THEN' 'GOTO' END OF ROW REMOVING LOOP;

  REMOVE ONLY IF SPACE IS PRESSING:
  'IF' NCUTS < N+NIRV 'THEN' 'GOTO' END OF ROW REMOVING LOOP;
  'IF' ROWLST[II] > 2*N 'THEN' 'GOTO' REMOVE NOW;

  REMOVE ONLY ONE LIMIT CUT IN FIRST INSTANCE:
  'FOR' I:=II+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  'IF' (ROWLST[I]>N 'AND' ROWLST[I]<2*N+1)
  'THEN' 'GOTO' END OF ROW REMOVING LOOP;

  REMOVE NOW:
  NCUTS:=NCUTS-1;
  'FOR' I:=II 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  ROWLST[I]:=ROWLST[I+1];
  'FOR' I:=II 'STEP' 1 'UNTIL' M+NCUTS+2 'DO'
  'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' T[I,J]:=T[I+1,J];
  HAVE ROMOVED ONE CUT:

  END OF ROW REMOVING LOOP: 'END';

TRY IF MORE SPACE WILL DO IT:
'IF' NCUTS<NIRV+N-1 'AND' 'NOT' OPTIMAL 'THEN' 'GOTO' STAGE 3;

'IF' 'NOT' OPTIMAL 'THEN' 'GOTO' STAGE 3;


STAGE 7:
CHECK FOR NORMAL END OF ALGORITHM:
'IF' INTEGER 'THEN' 'GOTO' END OF MIXI;

STAGE 8:
RR:=NAV+1;
CUT:
CUTN:=0; NUM:=0.001;
'IF' MAIN 'AND' 'NOT' OPTIMAL 'THEN' NUM:=0.005;

'FOR' I:=RR 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' ROWLST[I] < N+1  'THEN' 'BEGIN'
  'IF' 'NOT' ICOLLST[ROWLST[I]]=0 'THEN' 'BEGIN'
    'IF' 'NOT' ABS(ENTIER(T[I,N+1]+0.5*NUM)-T[I,N+1]) < NUM
    'THEN' 'BEGIN'
      R:=I;
      CUTN:=ROWLST[R]+2*N;
      NUM := ABS(ENTIER(T[I,N+1]+0.5*NUM)-T[I,N+1]);
      'END'; 'END'; 'END';

'IF' MAIN 'AND' CUTN=0 'THEN' 'BEGIN'
  'IF' PIVOT SIZE CUT 'THEN' 'GOTO' STAGE 9
  'ELSE' 'GOTO' STAGE 4; 'END';
```

```
'IF' NCUTS < N+NIRV 'THEN' 'GOTO' ENLARGE FOR NON LCUT;
'FOR' I:=RR 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' ROWLST[I]=CUTN 'AND' T[I,N+1]>0 'AND' T[I,N+2]>0
'THEN' 'BEGIN' CUTR:=I; 'GOTO' WRITE THE CUT; 'END';

ENLARGE FOR NON LCUT:
NCUTS:=NCUTS+1;  CUTR:=M+NCUTS;
'FOR' I:= 2,1 'DO' 'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
T[M+NCUTS+I,J]:=T[M+NCUTS+I-1,J];

WRITE THE CUT:
SATISFIED := 'FALSE';
ROWLST[CUTR]:=ROWLST[R]+2*N;
T[CUTR,N+1] := -0.99999999;  T[CUTR,N+2]:=FANCYHIGH;
RR:=R;

'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'BEGIN'
  T[CUTR,J]:=0;
  'IF' T[R,J]=0
  'OR' (COLLST[J] > 1000 'AND' COLLST[J] < 1001+NEQ)
  'THEN' 'GOTO' COEFFICIENT WRITTEN;
  NUM:=ABS(T[R,J]);

  TRY COMBINATION:
  'IF' 'NOT' NUM < 0.999999 'AND' 'NOT' ICL[J]=0 'THEN' 'BEGIN'
    NUM:=NUM-1; 'GOTO' TRY COMBINATION; 'END';
  'IF' ABS(NUM) < 0.000002 'THEN' 'GOTO' COEFFICIENT WRITTEN;
  QUO:=T[R,N+1]-ENTIER(T[R,N+1]);
  'IF' T[R,J] < 0 'THEN' QUO:=1-QUO;
  QUO := QUO/NUM;
  'IF' 'NOT' ICL[J]=0 'AND' QUO < 1 'THEN' QUO:=1;
  T[CUTR,J]:=-1/QUO;
  COEFFICIENT WRITTEN: 'END';

'IF' PIVOT SIZE CUT 'AND' CUTN=0
'THEN' 'GOTO' STAGE 9;
'IF' PIVOT SIZE CUT
'THEN' 'GOTO' INCOMING VARIABLE INTERCEPT;

'IF' 'NOT' MAIN 'THEN' 'GOTO' STAGE 17;

'IF' RECALLING TO STAGES 3 AND 4 'THEN' 'GOTO' STAGE 4;
'IF' NCUTS=N+NIRV+1 'THEN' 'GOTO' STAGE 3;
'IF'  NCUTS=N+NIRV 'THEN' 'GOTO' STAGE 9;

IMPLEMENTATION REENTRY OF STAGE 9:

STAGE 9:
KK:=KKK:=0;

TRY A HYPOTHETICAL STEP:
K:=KK;
R:=ROWN:=COLN:=0;  REENTRY:=2;
INTP(T,M+NCUTS,N,NEQ,NAV,ROWLST,COLLST,
IROWLST,R,K,ROWN,COLN,REENTRY);

SAVED R T := R;
```

```
'IF' REENTRY = -1 'AND' KK=0 'THEN'
'GOTO' EMPTY OF INTEGER SOLUTIONS;

'IF' R=0 'THEN' 'BEGIN'
  'COMMENT'
  HITTING THE UPPER LIMIT OF THE INCOMING VARIABLE MEANS
  THIS IS A CHEAP STEP;
  FEASIBLE := SATISFIED := 'FALSE';
  MAIN := RETURNED := 'TRUE';
  'GOTO' KNOWN STEP REENTRY OF STAGE 4; 'END';

'IF' PIVOT SIZE CUT 'THEN' 'BEGIN'
  PIVOT SIZE CUT := 'FALSE';
  RR:=R; KK:=K;
  RECALLING TO STAGES 3 AND 4 := 'TRUE';
  'GOTO' STAGE 10; 'END';

'IF' (REENTRY = -1 'OR' T[R,N+1] > 10)
'AND' KK#0 'THEN' 'BEGIN'
  KK:=0; 'GOTO' TRY A HYPOTHETICAL STEP; 'END';

'IF' 'NOT' (ROWN>2*N 'AND' ROWN<3*N+1) 'THEN' 'BEGIN'
  'COMMENT'
  EXISTING CUTS AT THE MAIN VERTEX ARE NOT BEING REACHED
  BY A SINGLE STEP.
  UNDER THOSE CIRCUMSTANCES, THERE IS NO POINT IN MAKING
  HYPOTHETICAL CUTS
  ;

  FEASIBLE := SATISFIED := 'FALSE';
  MAIN := RETURNED := 'TRUE';
  STEP ALREADY KNOWN := 'TRUE';
  'GOTO' INCOMING VARIABLE INTERCEPT REENTRY;  'END';

RR:=R;  KK:=K;

STAGE 10:
DNCUTS:=NCUTS;
MAKE COLUMN EXTRACT:
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS-NAV 'DO'
FDLST[I]:=ROWLST[I+NAV];
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS-NAV+2 'DO' 'BEGIN'
  COLEXTR[I,1]:=T[NAV+I,K];
  'FOR' J:=1,2 'DO' COLEXTR[I,1+J]:=T[NAV+I,N+J]; 'END';
DLST[1]:=COLLST[K];


STAGE 11:
UPDATE COLEXTR:
SAVED R T := R;

R:=ROWN:=0; K:=1; REENTRY:=1;
INTP(COLEXTR,M+NCUTS-NAV,1,NEQ,0,FDLST,DLST,
IROWLST,R,K,ROWN,COLN,REENTRY);

STAGES 12 AND 13:
CUTN:=0;  LV:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS-NAV 'DO'
'IF' COLEXTR[I,2] < 0 'OR' COLEXTR[I,3] < 0
'THEN' 'GOTO' NO SOLUTION IN THIS COLUMN;
```

```
'GOTO' CONTINUE WITH COLUMN AS INDICATED;

NO SOLUTION IN THIS COLUMN:
'IF' RECALLING TO STAGES 3 AND 4 'AND' INTERCEPT
'THEN' 'GOTO' CONTINUE WITH COLUMN AS INDICATED;

'IF' KK=KKK 'OR' KKK=0 'THEN' 'BEGIN'
  R:=SAVED R T; K:=KK; MAIN:='TRUE';
  STEP ALREADY KNOWN:='TRUE';
  'GOTO' INCOMING VARIABLE INTERCEPT REENTRY; 'END';

KKK:=KK;  KK:=0; 'GOTO' STAGE 9;

CONTINUE WITH COLUMN AS INDICATED:

CHECK INTEGER NATURE HYP VERTEX:
INITIATE LIMITING VALUE:
LV := 0;

'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS-NAV 'DO'
'IF' FDLST[I] < N+1 'THEN' 'BEGIN'

  AVOID ZERO:
  'IF' COLEXTR[I,1]=0 'THEN' 'GOTO' NEXT ROW;

  CHECK WHETHER FRACIONAL:
  'IF' ABS(ENTIER(COLEXTR[I,2]+0.001) - COLEXTR[I,2]) < 0.002
  'THEN' 'GOTO' NEXT ROW;

  CHECK WHETHER INTEGER RESTRICTED:
  'IF' ICOLLST[FDLST[I]] = 0 'THEN' 'GOTO' NEXT ROW;

  PREFERENCE ON INCOMING VARIABLE:
  'IF' FDLST[I] = COLLST[KK] 'OR' FDLST[I]=COLLST[KK]-10000
  'THEN' 'BEGIN'
    'COMMENT'
    A CUT ON THE HYPOTHETICAL INCOMING VARIABLE IS INDICATED.
    A ROW EXTRACT OF INITIALLY ONLY ONE ROW IS NEEDED;
    CUTR:=NAV+I;
    CUTN:=FDLST[I];
    'GOTO' STAGE 14; 'END';

    DO NOT CUT OVER AGAIN:
    'FOR' CHECKI := NAV+1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
    'IF' ROWLST[CHECKI]=FDLST[I]+N 'OR' ROWLST[CHECKI]
    =FDLST[I]+2*N 'THEN' 'BEGIN'
      'IF' T[CHECKI,N+1]<0 'OR' T[CHECKI,N+2]<0 'THEN'
      'GOTO' NEXT ROW; 'END';

  'IF' COLEXTR[I,1] > 0
  'THEN' NLV := (COLEXTR[I,2]-ENTIER(COLEXTR[I,2]))
  /COLEXTR[I,1]
  'ELSE' NLV := (COLEXTR[I,3]-ENTIER(1+COLEXTR[I,3]))
  /COLEXTR[I,1];
  'IF' NLV>LV 'THEN' 'BEGIN'
    CUTN:=FDLST[I]; CUTR:=I; LV:=NLV; 'END';
  NEXT ROW: 'END';
```

```
'IF' 'NOT' CUTN=0 'THEN' 'GOTO' STAGE 14;

'IF' RECALLING TO STAGES 3 AND 4 'AND' INTERCEPT
'THEN' 'BEGIN'
  RECALLING TO STAGES 3 AND 4 := INTERCEPT := 'FALSE';
  'GOTO' STAGE 4; 'END';

'IF' KK=KKK 'THEN' 'GOTO' INCOMING VARIABLE INTERCEPT;

KKK:=KK; KK:=0; 'GOTO' TRY A HYPOTHETICAL STEP;

STAGE 14:
MAKE A ROWEXTRACT:
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
FDLST[J]:=COLLST[J];
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
ROWEXTR[1,J]:=T[CUTR,J];

'IF' CUTN#COLLST[KK] 'AND' CUTN#COLLST[KK]-10000
'AND' 'NOT' R=0
'THEN' 'BEGIN'
  'COMMENT'
  TWO SEPERATE ROWS WILL BE NEEDED;
  R:=NAV+R; 'GOTO' COPY; 'END';

MM:=1;
RENAME ROWINDEX TO BE FIRST ROW OF EXTRACT:
R:=1;
'GOTO' COPY ROWNAME;

COPY:  MM:=2;
DLST[2]:=ROWLST[R];
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' ROWEXTR[2,J]:=T[R,J];
RENAME ROWINDEX TO BE SECOND ROW OF EXTRACT:
R:=2;

COPY ROWNAME:
DLST[1]:=ROWLST[CUTR];

COPY LOWER BORDER:
'FOR' I:=1,2,3 'DO' 'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
ROWEXTR[MM+I,J]:=T[M+DNCUTS+I,J];


STAGE 15:
UPDATE ROWEXTR:
K:=KK; COLN:=COLLST[KK]; REENTRY:=1;
SAVED R E:= R;
INTP(ROWEXTR,MM,N,NEQ,0,DLST,FDLST,IROWLST,
R,K,ROWN,COLN,REENTRY);

PUT ROWEXTRACT IN T AND SAVE LEADING ROWS OF T:
'FOR' I:=1 'STEP' 1 'UNTIL' MM+4 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' 'BEGIN'
  COP:=T[I,J]; T[I,J]:=ROWEXTR[I,J];
  ROWEXTR[I,J]:=COP; 'END';
```

```
'FOR' I:=1 'STEP' 1 'UNTIL' MM+1 'DO' 'BEGIN'
  NAME:=ROWLST[I]; ROWLST[I]:=DLST[I];
  DLST[I]:=NAME; 'END';

FDLST[K]:=COLLST[K]; COLLST[K]:=ROWN;
M:=MM; NAV:=NCUTS:=0;
MAIN:='FALSE';
'GOTO' CHECK AND LIMIT;

STAGE 16:  'IF' SATISFIED 'THEN' 'GOTO' STAGE 8;

STAGE 17:

PUT TABLEAUSHUFFLE BACK:
'FOR' I:=1 'STEP' 1 'UNTIL' MM+3 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' 'BEGIN'
  COP:=T[I,J]; T[I,J]:=ROWEXTR[I,J]; ROWEXTR[I,J]:=COP; 'END';

'FOR' I:=1 'STEP' 1 'UNTIL' MM+1 'DO' 'BEGIN'
  NAME:=ROWLST[I]; ROWLST[I]:=DLST[I]; DLST[I]:=NAME; 'END';
COLLST[K]:=FDLST[K]; FDLST[K]:=ROWN;


RESTORE PARAMETERS:
M:=MMM; NAV:=DNAV;
PUT NEW AND CORRECT ORDER PARAMETERS:
MM:=MM+NCUTS; NCUTS:=DNCUTS+NCUTS;

BACKWARD STEP IN ROWEXTR:
R:=SAVED R E; REENTRY:=1;
INTP(ROWEXTR,MM,N,NEQ,0,DLST,FDLST,IROWLST,
R,K,COLN,ROWN,REENTRY);

MAKE CUT ON INCOMING VARIABLE MORE EXACT:
'IF' R=1 'AND' MM=2 'THEN' 'BEGIN'
  'IF' DLST[2]=FDLST[K]+N
  'OR' DLST[2]=FDLST[K]+2*N
  'OR' DLST[2]=FDLST[K]-10000+N
  'OR' DLST[2]=FDLST[K]-10000+2*N
  'THEN' ROWEXTR[2,N+1]:=0.0000000001; 'END';

STAGE 18:
ASSEMBLE WITH MAIN TABLEAU:

'IF' NCUTS=0 'THEN' 'GOTO' ATTEND LIMITCUTS BY COLUMNADJUSTM;

ROWLST[M+NCUTS]:=DLST[MM];

EXTEND TABLEAU FOR SUBSIDIARY CUT:
'FOR' I:=M+NCUTS+3, M+NCUTS+2, M+NCUTS+1 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO' T[I,J]:=T[I-1,J];

COPY THE SUBSIDIARY CUT:
'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
T[M+NCUTS,J]:=ROWEXTR[MM,J];

ATTEND LIMITCUTS BY COLUMNADJUSTM:
'FOR' I:=1 'STEP' 1 'UNTIL' M+NCUTS 'DO'
'IF' ROWLST[I]<N+1 'AND' ROWLST[I] = DLST[1] 'THEN' 'BEGIN'
```

```
   DO NOT ADJUST IF DIFFERENCE COULD BE ROUNDING:
   'IF' ABS(T[I,N+1]-ROWEXTR[1,N+1])
   +ABS(T[I,N+2]-ROWEXTR[1,N+2]) < 0.001
   'THEN' 'GOTO' ATTEND UPPER LIMITS;

   T[I,N+1]:=ROWEXTR[1,N+1]; T[I,N+2]:=ROWEXTR[1,N+2];
   'GOTO' ATTEND UPPER LIMITS; 'END';

ATTEND UPPER LIMITS:
'FOR' I:= 1,2 'DO' 'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
T[M+NCUTS+I,J]:=ROWEXTR[MM+I,J];

'IF' RECALLING TO STAGES 3 AND 4 'THEN' 'BEGIN'
  MAIN := 'TRUE'; INTERCEPT:='FALSE';
  R:=SAVED R T;

   'GOTO' INCOMING VARIABLE INTERCEPT; 'END';

'GOTO' STAGE 9;

UNBOUNDED:
NEWLINE(2);
WRITETEXT('('UNBOUNDED%SOLUTION%OF%THE%CONTINUOUS%PROBLEM')');
NEWLINE(2);
WRITETEXT('('COLUMN')'); PRINT(COLN,7,0);
WRITETEXT('('UNBOUNDED')');
NEWLINE(2);
'GOTO' END OF MIXI;

EMPTY:
NEWLINE(2);
WRITETEXT('('EMPTY%PROBLEM')');
NEWLINE(2);
'IF' 'NOT' SATISFIED 'THEN' 'GOTO' NO INTEGER SOLUTION;
'GOTO' END OF MIXI;

EMPTY OF INTEGER SOLUTIONS:
NEWLINE(2);
NO INTEGER SOLUTION:
WRITETEXT('('PROBLEM EMPTY%OF%INTEGER%SOLUTIONS')');


END OF MIXI:

'IF' 'NOT' FEASIBLE 'OR' 'NOT' INTEGER
'THEN' 'GOTO' FINAL END OF MIXI;

ORDER FOR FINAL EXIT:
ORDL(T,M+NCUTS,N,2,2,ROWLST,COLLST);

REMOVE EVENTUALLY SUPERFLUOUS CUTS:
'FOR' II:=1 'STEP' 1 'UNTIL' M+3*N 'DO' 'BEGIN'
  'IF' II>M+NCUTS 'THEN' 'GOTO' FINAL END OF MIXI;
  'IF' ROWLST[II] > 3*N 'THEN' 'GOTO' FINAL END OF MIXI;
  'IF' ROWLST[II] < N+1 'THEN' 'GOTO'
  END OF SURPLUS ROW REMOVING LOOP;
```

```
  NCUTS:=NCUTS-1;
  'FOR' I:=II 'STEP' 1 'UNTIL' M+NCUTS 'DO'
  ROWLST[I]:=ROWLST[I+1];
  'FOR' J:=1 'STEP' 1 'UNTIL' N+2 'DO'
  'FOR' I:=II 'STEP' 1 'UNTIL' M+NCUTS+2 'DO'
  T[I,J]:=T[I+1,J];
  II:=II-1;
  END OF SURPLUS ROW REMOVING LOOP: 'END';

FINAL END OF MIXI: 'END';
```

REFERENCES

[1] Abadie, J. (Editor)
Nonlinear Programming
Amsterdam, North Holland Publishing Co., 1967
Referred to: This bibliography [3], [40].

[2] Baumol, W.J.
Economic Theory and Operations Analysis
Prentice Hall, NewJersey, U.S.A., 1951, 1965, 1972
Referred to: 11.2

[3] Beale, E.M.L.
An Introduction to Beale's Method of Quadratic Programming
In: Abadie (Editor) [1], 1967, pp. 143-150
Referred to: 16.11

[4] Charnes, A.
Optimality and Degeneracy in Linear Programming.
Econometrica, 20 (1952), pp. 160-170
Referred to: 8.10

[5] Charnes, A., Cooper, W.W., and Henderson, A.
An Introduction to Linear Programming.
New York, John Wiley, 1953, 1958
Referred to: 7.1, 8.10

[6] Cottle, R.W.
The Principal Pivoting Method of Quadratic Programming.
In:
Dantzig, G.B., and Veinott, A.F. (Editors)
Mathematics of the Decision Sciences.
(Proceedings of the 1967 Summer School)
American Mathematical Society, 1968
Referred to: 16.2, 16.3

[7] Dakin, R.J.
A Tree-Search Algorithm for Mixed Integer Programming Problems.
The Computer Journal, 8 (Apr 1965), pp.250-255
Referred to: 20.2, 20.3

[8] Dantzig, G.B.
Linear Programming and Extensions.
Princeton University Press, Princeton, U.S.A., 1963
Referred to: 11.3, 13.1, 16.2

[9] Dantzig, G., Eisenberg, A. and Cottle, R.W.
Symmetric Dual Nonlinear Programs
Operations Research Center, University of California
Researcg Report No. 38
Subsequently published in:
Pacific Journal of Mathematics 15, 1965, pp.809-812

[10] Fiacco, A.V., and McCormick, G.P.
Non-Linear Programming: Sequential Unconstrained Maximization Techniques.
New York, Wiley, 1968.
Referred to: 18.2

[11] Garvin, W.W.
Introduction to Linear Programming.
New York, McGraw Hill, 1960
Referred to: 8.1, 9.5, 11.2, 13.1

[12] Gass, S.I., and Saati, T.
Parametric Objective Function.
Journal of the Operations Research Society of America, 2 (1954), pp.316-319
Referred to: 13.1

[13] Gass, S.I., and Saati, T.
The Computational Algorithm for the Parametric Objective Function.
Naval Research Logistics Quarterly, 2 (1955), pp.39-45
Referred to: 13.1

[14] Geary, R.C.
Elements of Linear Programming With Economic Applications
Griffins Statistical Monographs and Courses Nr. 15
Griffin, London 1973
Referred to: 8.10

[15] Gomory, R.
An Algorithm for Integer Solutions to Linear Problems
In:
Graves, R.L., and Wolfe, P. (Editors)
Recent Advances in Mathematical Programming.
McGraw Hill, 1963
Referred to: 21.1

[16] Gomory, R.
An Algorithm for the Mixed Integer Problem.
Rand Corporation p. 1885, Santa Monica, Cal. U.S.A.,
June 1960
Referred to: 21.1

[17] Goncalves, A.
Primal-Dual and Parametric Methods in Mathematical
Programming.
Thesis, Ph.D., Birmingham, Dept. of Mathematical Statistics,
1970.
Referred to: 17.6

[18] Heesterman, A.R.G.
Allocation Models and their Use in Economic Planning.
Reidel, Dordrecht, Netherlands, 1971
Referred to: 8.10

[19] Heesterman, A.R.G.
Forecasting Models for National Economic Planning.
Reidel, Dordrecht, Netherlands, 1970, 1972
Referred to: 4.1

[20] Heesterman, A.R.G.
Special Simplex Algorithm for Multi-Sector Problems.
Numerische Mathematik 12 (1968) pp. 288-306 (In English)
Referred to: 8.2, 13.2

[21] Hohn, F.
Elementary Matrix Algebra.
The Macmillan Company, New York,
& Collier Macmillan, London, U.K., and Toronto, Canada 1973
Referred to: 14.4

[22] John, F.
Extremum Properties with Inequalities as Subsidiary
Conditions.
(Presented to R. Courant)
New York, Interscience Publishers, 1968.
Referred to: 15.3

[23] Kim, C.
Introduction to Linear Programming.
New York, Rinehart and Winston, 1971
Referred to: 11.3

[24] Kuhn, H.W., and Tucker, A.W.
Nonlinear Programming.
In:
Second Berkeley Symposium on Mathematics and Probability Theory.
University of California Press, Berkeley, Calif., U.S.A., 1951, pp. 481-492
Also reprinted in:
Newman, P. (Editor)
Readings in Mathematical Economics.
The Johns Hopkins Press, Baltimore, U.S.A., 1968
Vol. 1, pp.3-14.
Referred to: 15.3

[25] La Grange (de la Grange)
Méchanique Analytique (Section Quatrième)
Paris, Veuve Desaint, 1788
Reprinted in:
Oeuvres de la Grange
Paris, Gauthier et fils, 1888 (Tome XI)
Referred to: 15.3

[26] Land, A.H., and Doig, A.G.
An Automatic Method for Solving Discrete Programming Problems.
Econometrica 28, pp.497-520, 1960
Referred to: 20.2, 20.3

[27] Mangasarin, O.L.
Nonlinear Programming.
New York, McGraw Hill, 1969
Referred to: 14.2

[28] Maurer, S.
Pivotal Theory of Determinants.
In:
Balinski, M.L.
Pivoting and Extensions: In Honour of A.W. Tucker.
Amsterdam, North Holland Publishing Co., 1974
Referred to: 5.5

[29] Parlett, B.N.
The Symmetric Eigenvalue Problem.
Prentice Hall, Englewood Cliffs, New Jersey, U.S.A., 1980
Referred to: 14.4

[30] Ponstein, J.
Seven Types of Convexity.
Siam Review 9 (1967), pp.115-119
Referred to: 14.2

[31] Powell, M.J.D.
A Fast Algorithm for Nonlinearly Constrained Optimization Calculations.
In:
Numerical Analysis
Conference, Dundee, 1977, Siam
Springer Verlag
Referred to: 18.2

[32] Samuelson, P.A.
Foundations of Economic Analysis.
Harvard University Press, Cambridge, Mass., U.S.A., 1967
Referred to: 15.5

[33] Theil, H.
Principles of Econometrics.
New York, John Wiley & Amsterdam, N.Holland Pubg. Co., 1971.
Referred to: 14.4

[34] Tucker, A.W.
Dual Systems of Homogeneous Linear Inequalities.
In:
Kuhn, H.W., and Tucker, A.W. (Editors)
Linear Inequalities and Related Systems.
Princeton University Press, New Jersey, U.S.A., 1965
Referred to: 11.3

[35] Van de Panne, C.
Methods of Linear and Quadratic Programming.
Amsterdam, North Holland Publishing Co., 1975
Referred to: 8.10, 17.6, 17.6

[36] Van de Panne, C., and Whinston, A.
The Simplex and Dual Method for Quadratic Programming.
Operational Research Quarterly, 15, Nr. 4, pp.355-387
Referred to: 16.2, 16.3

[37] Van de Panne, C., and Whinston, A.
Simplicial Methods in Quadratic Programming.
Naval Research Logistics Quarterly, 2 (1964) pp. 273-302
Referred to: 16.3

[38] Weingartner, M.
Mathematical Programming and the Analysis of Capital Budgeting Problems.
Prentice Hall, Englewood Cliffs, New Jersey, U.S.A., 1963
Referred to: 20.1

[39] Westphal, L.E.
Planning Investment Decisions with Economies of Scale.
North Holland Publishing Co., 1971
Referred to: 19.2

[40] Whinston, A.
Some Implications of the Conjugate Function Theory to Duality.
In Abadie [1], 1967, pp.77-96
Referred to: 15.6

[41] Zionts, S.
Linear and Integer Programming.
Prentice Hall, Englewood Cliffs, New Jersey, U.S.A., 1974
Referred to: 21.1

[42] Zoutendijk, G.
Mathematical Programming Methods.
Amsterdam, North Holland Publishing Co., 1975
Referred to: 8.10

I N D E X