Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

To: Scientific and Engineering Computation Group

From: M. Douglas McIlroy

Date: 12 September 1955

SUBJECT: ITERATIVE SOLUTION OF LINEAR SYSTEMS HAVING SPARSE MATRICES

Abstract: A preliminary routine for solving large-order linear systems having sparse matrices is described. An iterative approach is used in order to take advantage of the absence of most matrix elements. Such a routine could handle a 100 x 100 matrix of this character in core storage.

1. Introduction

In many problems, such as elliptic partial differential equations or network analyses, large sets of simultaneous linear equations are encountered which are characterized by a preponderance of zero elements off the main diagonal of the matrix. These matrices are in addition symmetric and loaded* on the diagonal. In order to capitalize on these properties, the Jacobi iteration described in section 3 was selected as the basis for a Whirlwind routine. For large systems, the iterative approach to solution has two advantages. In the first place, only enough storage registers need be provided to take care of the non-zero elements in the matrix, whereas reductions such as Gauss-Jordan or Crout can not so profit by the character of the matrix. Secondly, initial data are used in each stage of the iteration, obviating the necessity of carrying large numbers of superfluous digits in order to overcome round-off. One can expect, though, that the iteration will be very slow in convergence for large systems; and this factor must be weighed against the advantages.

2. Description of the Program, fc TAPE 141-94-107

2.1 Input Data

2.1.1 Preset Parameters

zm1 = n	order of matrix
zm2 = a	number of non-zero superdiagonal elements

* "Loaded" means $\sum_{j=i} |a_{ij}| \leq a_{ii}$ for all i.

zm3 = location of 1st register of data

zm4 = location of 1st temporary register

zm5 = zm1+zm1 not assigned by programmer

2.1.2 Data Storage

Data are stored sequentially in the following order beginning with register zm3:

a) n double registers containing right-hand side

b) P, any distinctive one-register number, preferably not zero, to be used for checking purposes

c) index to superdiagonal elements (see 2.2)

d) P, same as b

e) a double registers listing superdiagonal elements row-wise

f) P, same as b

g) n double registers containing diagonal elements

h) P, same as b

2.2 Index

In order to identify the proper location of the listed superdiagonal elements in the matrix, an index is constructed. An index group of registers is included for each row of the matrix. The group must contain a number of binary digits (excluding sign digits) greater than or equal to the number of superdiagonal elements in the row. Thus for the ith row the index group will contain r registers where

$$n-i \leq 15r \leq n-i+14, \qquad i=1, 2, \ldots n-1$$

except for i=n, where r is taken to be 1.

The succeeding digits of the group are 1 or 0 according as the corresponding superdiagonal element of the matrix is non-zero or zero. The signs of the registers are positive.

Example: Row one of a 40 x 40 matrix has non-zero elements in columns 10, 20, 30, 40. The index group for this row is

0.00100
0.04002 (octal)
0.00100

The index groups are listed in order row-by-row.

Example:

Matrix	Index
$\begin{bmatrix} 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} & \frac{1}{4} \\ & \frac{1}{4} & 1 & \\ \frac{1}{4} & & 1 & \\ \frac{1}{4} & \frac{1}{4} & & 1 \end{bmatrix}$	0.54000 0.50000 0.00000 0.00000 0.00000

Note that the last index register always contains zero.

The number of index registers may be determined by the formula

$$1 + (Q+1)(R+15Q/2)$$

where Q and R are the quotient and remainder in the division (n-1)/15.

2.3 Length of Routine

Instructions: 238

Temporary registers: 6n registers starting at zm 4

Data storage: 4n + 2a + (Q+1)(R+15Q/2) + 5 registers

2.4 Set-up and Checking

Built into the routine is a checking facility to guard against gross coding errors in the input data. Before the first entry to the iteration, the checking procedure must be performed. It is entered by

```
sp  e8          entry for checking
```

Check alarms are generated for any of several errors:

Check alarm in register	Cause
1e1	Incorrect number of elements in vector b, or incorrect number of index registers
2e1	Number of off-diagonal elements listed disagrees with zm2
3e1	Incorrect number of diagonal elements
e5-2	Number of non-zero index bits disagrees with zm2

2.5 Operation and Results

There are two entries for calculating with the routine. The first time the routine is used with new data it must be entered by

```
sp  e7          entry with new data
```

For further iterations it is entered by

```
sp  b4          entry for iterations
```

At the end of the kth iteration, the latest results are available in the temporary registers starting with zm 4:

2n registers contain $x_{3k+2} - x_{3k+1} = \Delta x_{3k+1}$

2n registers contain x_{3k+2} $\quad k = 0,1,2,\ldots$

2n registers contain x_{3k+3}

A quick measure of accuracy at this stage is given by the contents of register g1. This quantity is the exponent of the greatest Aitken correction made during the iteration. The routine provides, as a guard against instability, that this quantity does not increase. This is done by taking

$$x_{3k+3} = x_{3k+2}$$

if the criterion is not met. Thus, for a complete check of precision the quantities Δx_{3k+1} might be examined. One must multiply these by something like a factor of 5 (this is empirical) to get an estimate of accuracy.

2.6 Time and Space Comparisons

In its present form, the routine requires about 22 ms per superdiagonal element plus 28 ms per row for each complete Aitken-Jacobi cycle. Since the number of nonvanishing elements in matrices of the type involved here is practically proportional to the order of the matrix, the length of an iteration is proportional to n. The number of iterations required depends on the character of the matrix involved and the accuracy desired, but a fair guess is that (for a given accuracy) this number is proportional to n, giving a total operating time proportional to n^2, compared to the time for reduction schemes which is proportional to n^3. Input time is of course reduced using the iterative scheme.

For 4-decimal accuracy, in a matrix with 4 off-diagonal elements per row, limited experience leads one to expect the iterative method to take about $75n^2$ ms while Crout's method (on numbers in core memory) takes about $3n^3$. Consequently, we expect no profit to be gained by iteration on matrices of order less than 25. It happens that this is just about the maximum matrix which can be handled by the present Crout reduction routine. The iterative procedure steps in just at the point where storage limitations cause the Crout method to break down.

Storage requirements are small for the iterative routine. Instead of 2n(n+2) registers being required for data storage as in the Crout routine, only 8n registers are necessary for various vectors plus about $n^2/30$ index registers and the registers for the non-zero off-diagonal elements. A 100 x 100 symmetrix matrix with 400 non-zero off-diagonal elements would readily fit into core storage.

3. Theory

In matrix notation, the routine solves

(1) $Ax = b$

by the iteration

(2) $x_{i+1} = b - (A-I)x_i$

for which the error at the i+1st step is given by

(3) $e_{i+1} = (I-A)e_i$

where $e_i = x_i - \bar{x}$ and $\bar{x}$ is the solution of (1).

By the difference equation (3) we see that the iteration converges if

(4) $|1 - \lambda_j| < 1 \qquad j = 1,2...n$

where λ_j are the eigenvalues of A. In general, this criterion is satisfied if the rows of a loaded matrix are normalized by dividing through by the diagonal elements.[1] In the case of electric networks with real impedance or of elliptic partial differential equations, the requirement is met, for otherwise a nontrivial solution of the problem

$$Ax = 0$$

with vanishing boundary values would exist.

Apparently convergence is good if the eigenvalues of the normalized problem are clustered about unity, but poor if there are eigenvalues near either end of the interval (0,2). A further difficulty of the process is that it is first-order, with e_i decreasing by a constant factor at each stage. Roughly speaking, the ^{2}process may be made second order by using Aitken's ^{2}process on the components of x_i according to

(5) $$x_{3k+3} = x_{3k} - \frac{(\Delta x_{3k})^2}{\Delta^2 x_{3k}}$$

where the symbolic division means a component-by-component operation. Unfortunately, it is possible for the denominator in (5) to vanish while the numerator remains different from zero, so the Aitken correction can lead to instability. Consequently, in the routine, provision is made to limit the magnitude of this correction by taking

$$x_{3k+3} = x_{3k+2}$$

in cases where the Aitken correction would be large.

---

1. A. S. Householder, Principles of Numerical Analysis, New York, 1953, p. 148, p. 126.

4. Possible Improvements

In cases where four or less decimal digits are required, great advantage could be obtained by using single length arithmetic (perhaps dispensing with the Aitken modification, which would require safeguards for the division).

For use with elliptic equations, where all the off-diagonal elements are equal, the listing and looking up of these elements should be eliminated, resulting in a general relaxation routine. Also, since all the diagonal elements are equal, the normalizing feature should be dropped and the matrix should be put into normalized form before the calculation. With these two modifications, a total time for 4-place accuracy in solving elliptic equations in two dimensions might be reduced to something like

$$10n^2ms$$

with storage requirements about $2n + n^2/30$. A 200 x 200 matrix could be fit in core storage, but the time it would take is unknown; for an extrapolation to $n = 200$ of the time estimates based on experience with $n = 20$ is of dubious value.

In the case of larger matrices, the indexing scheme used here may be replaced by a still simpler one with a resulting improvement in utilization of storage. This method is to actually store the indices, i and j, for each non-vanishing element. These indices could be packed in one Whirlwind register if $n \leqslant 2^8$. When $n^2/30$ exceeds the number of superdiagonal elements, this procedure leads to a saving in data storage space; it also would provide a simpler program.

The method described here could be used for non-symmetric matrices; but would probably not be useful, for nonsymmetric problems with sparse matrices are rare, and the advantages of iteration arise from this particular characteristic.

Signed: M. Douglas McIlroy
M. Douglas McIlroy

MDM:gn