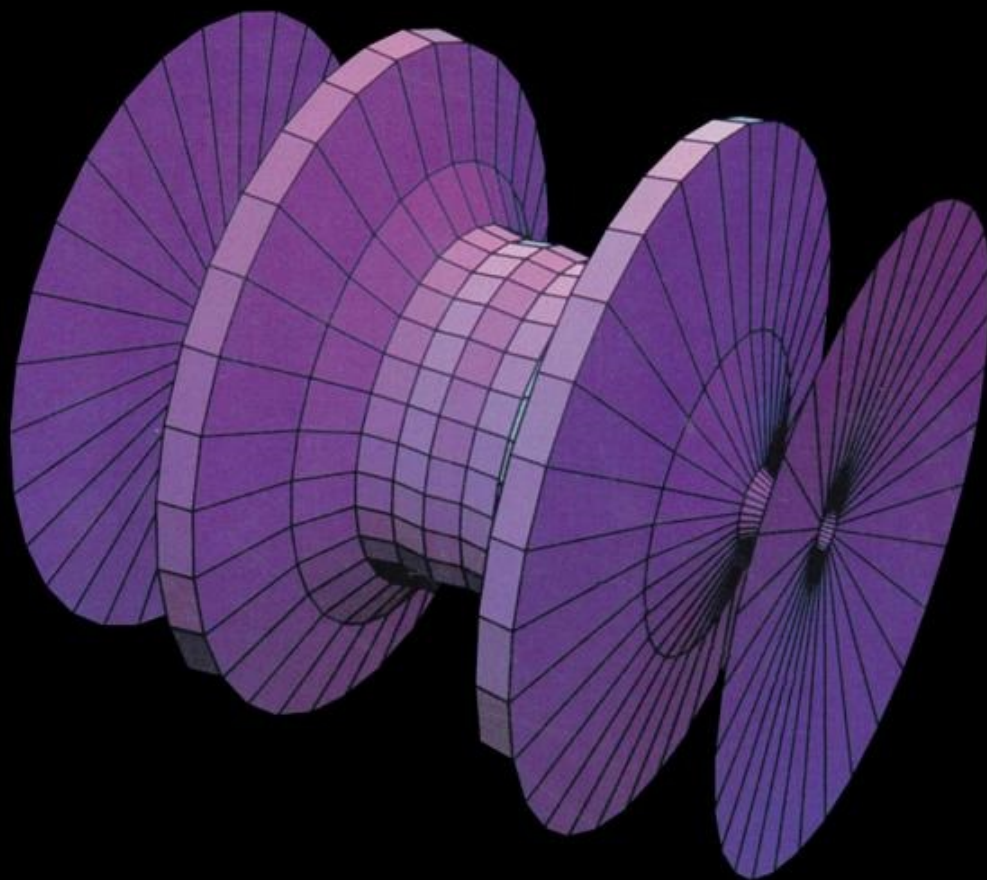


Compatible with
Mathematica Version 2.0

***Mathematica* by Example**



Martha L. Abell
James P. Braselton

Mathematica by Example

Martha L. Abell

*Department of Mathematics and Computer Science
Georgia Southern University
Statesboro, Georgia*

James P. Braselton

*Department of Mathematics and Computer Science
Georgia Southern University
Statesboro, Georgia*



ACADEMIC PRESS, INC.

Harcourt Brace Jovanovich, Publishers

Boston San Diego New York
London Sydney Tokyo Toronto

This book is printed on acid-free paper. ☺

Copyright © 1992 by Academic Press, Inc.
All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

Mathematica is a registered trademark of Wolfram Research, Inc.
Macintosh is a registered trademark of Apple Computer, Inc.
Windows is a registered trademark of Microsoft Corporation.

ACADEMIC PRESS, INC.
1250 Sixth Avenue, San Diego, CA 92101

United Kingdom Edition published by
ACADEMIC PRESS LIMITED
24-28 Oval Road, London NW1 7DX

LCCCN: 91-58715
ISBN: 0-12-041540-2

Printed in the United States of America
92 93 94 95 9 8 7 6 5 4 3 2 1

PREFACE

Mathematica by Example is intended to bridge the gap which has existed between the very elementary handbooks available on *Mathematica* and those reference books written for the more advanced *Mathematica* users. This book is an extension of a manuscript which was developed to quickly introduce enough *Mathematica* commands to a group of students at Georgia Southern University that they could apply *Mathematica* towards the solution of nonlinear ordinary differential equations. In addition to these most basic commands, these students were exposed to the vast uses of lists in *Mathematica*. Having worked through this material, these students were successfully able to take advantage of the capabilities of *Mathematica* in solving problems of interest to our class.

Mathematica by Example is an appropriate reference book for readers of all levels of *Mathematica* experience. It introduces the very basic commands and includes examples of applications of these commands. It also includes commands useful in more advanced areas such as ordinary and partial differential equations. In all cases, however, examples follow the introduction of new commands. Of particular note are the sections covering *Mathematica* Packages (Chapters 7, 8, and 9), because the commands covered in these chapters are absent from most *Mathematica* reference books. The material covered in this book applies to all versions of *Mathematica* as well with special notes concerning those commands available only in Version 2.0. Other differences in the various versions of *Mathematica* are also noted.

Of course, appreciation must be expressed to those who assisted in this project. We would like to thank our department head Arthur Sparks for his encouragement and moral support and for being the instigator of the Computer Calculus Project which initiated the idea of writing a book like *Mathematica by Example*. We would also like to thank Prof. William F. Ames for suggesting that we publish our work and for helping us contact the appropriate people at Academic Press. We would like to express appreciation to our editor, Charles B. Glaser, and our production manager, Simone Payment, for providing a pleasant environment in which to work. We would also like to thank our colleagues for taking the time to review our manuscript as it was being prepared for publication. We appreciated their helpful comments. Finally, we would like to thank those close to us for enduring with us the pressures of meeting a deadline and for graciously accepting our demanding work schedules. We certainly could not have completed this task without your care and understanding.

M. L. Abell

J. P. Braselton

Chapter 1

Getting Started

■ *Mathematica*, first released in 1988 by Wolfram Research, Inc., is a system for doing mathematics on a computer. It combines symbolic manipulation, numerical mathematics, outstanding graphics, and a sophisticated programming language. Because of its versatility, *Mathematica* has established itself as the computer algebra system of choice for many computer users. Overall, *Mathematica* is the most powerful and most widely used program of this type. Among the over 100,000 users of *Mathematica*, 28% are engineers, 21% are computer scientists, 20% are physical scientists, 12% are mathematical scientists, and 12% are business, social, and life scientists. Two-thirds of the users are in industry and government with a small (8%) but growing number of student users. However, due to its special nature and sophistication, beginning users need to be aware of the special syntax required to make *Mathematica* perform in the way intended.

■ The purpose of this text is to serve as a guide to beginning users of *Mathematica* and users who do not intend to take advantage of the more specialized applications of *Mathematica*. The reader will find that calculations and sequences of calculations most frequently used by beginning users are discussed in detail along with many typical examples. We hope that *Mathematica by Example* will serve as a valuable tool to the beginning user of *Mathematica*.

■ A Note Regarding Different Versions of Mathematica

For the most part, *Mathematica by Example* was created with Version 1.2 of *Mathematica*. With the release of Version 2.0 of *Mathematica*, several commands from earlier versions of *Mathematica* have been made obsolete. In addition, Version 2.0 incorporates many features not available in Version 1.2. *Mathematica by Example* adopts the following conventions:

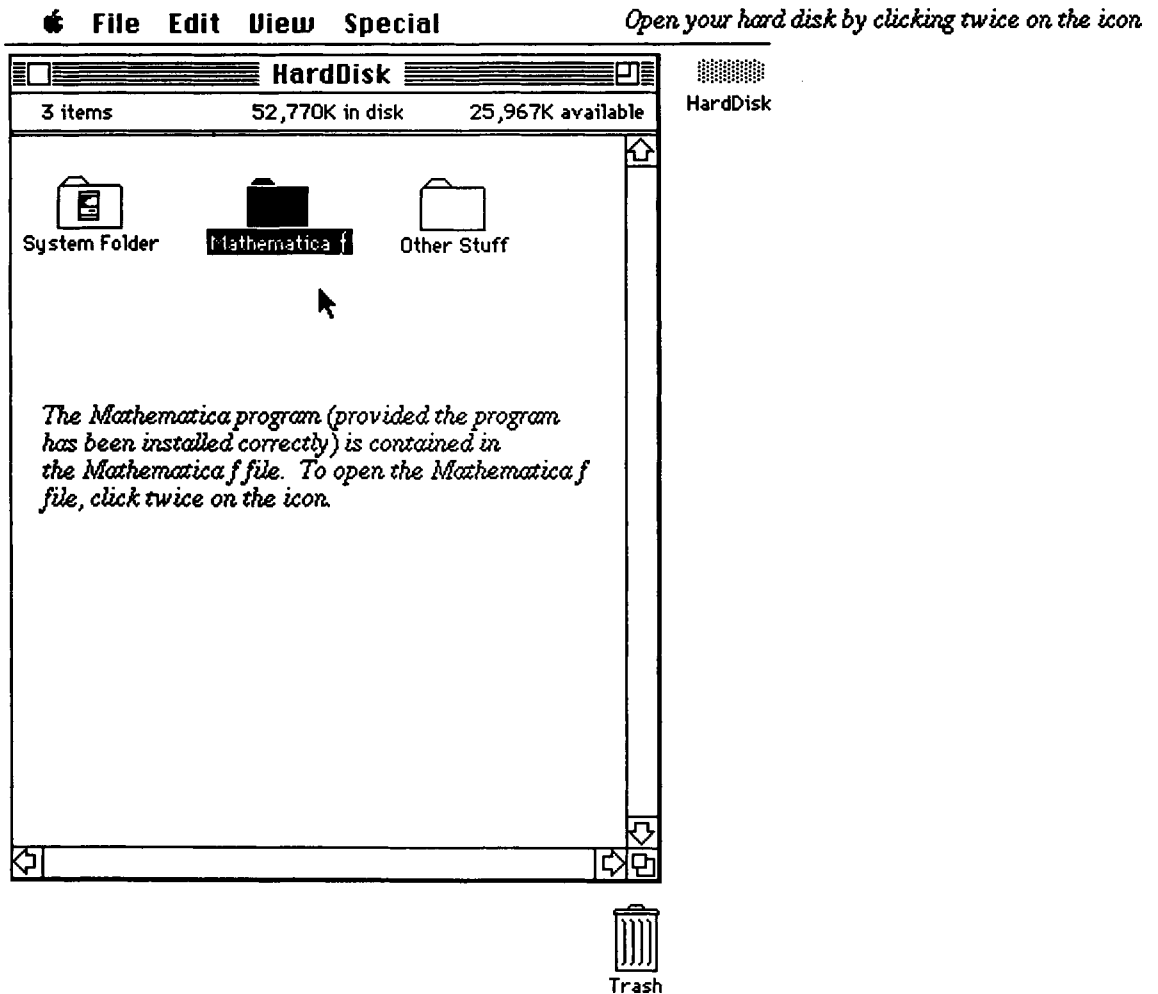
Sections that discuss features of Version 1.2 will begin with symbols like ■ ■ □ ;
unless otherwise noted, these commands are supported under Version 2.0.

Sections that discuss the features of Version 2.0 will begin with symbols like ● ● ○ .
These sections are NOT pertinent to Version 1.2.

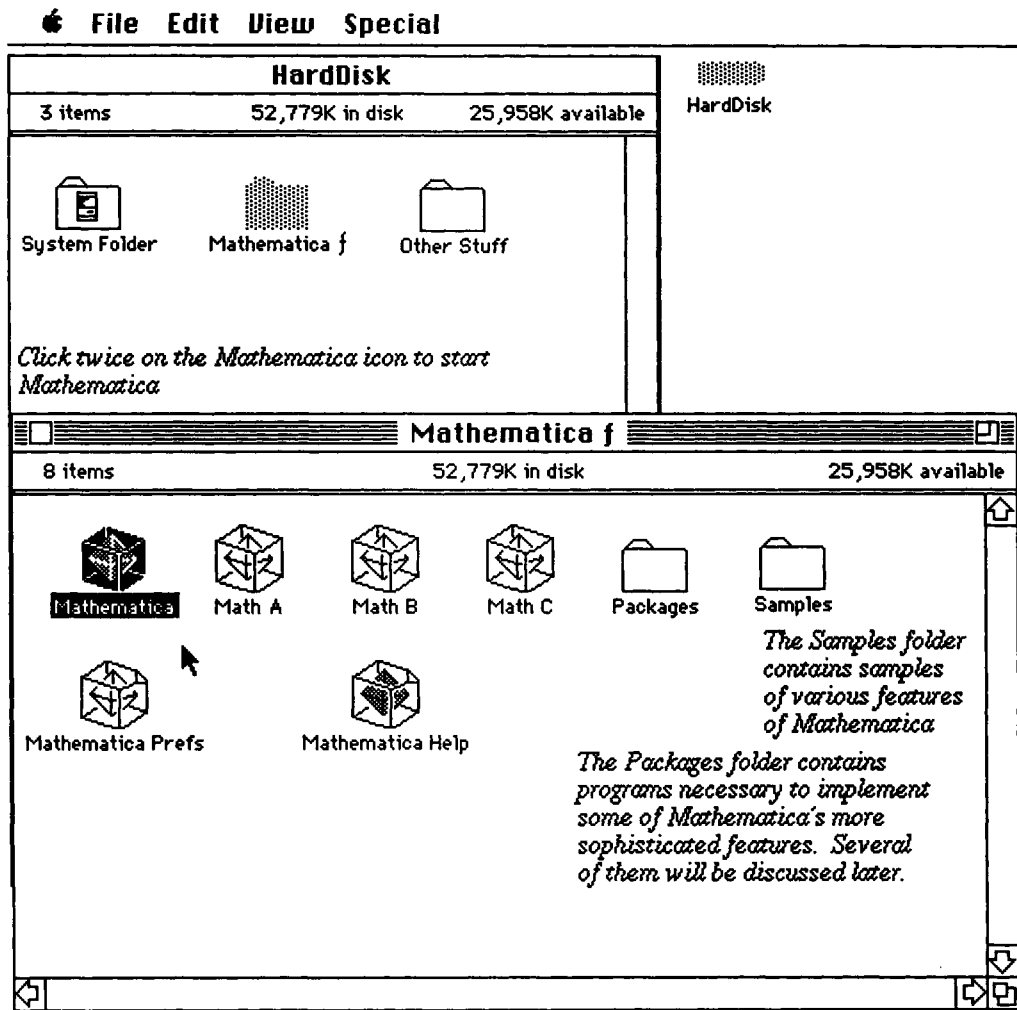
■ 1.1 Macintosh Basics

Since *Mathematica by Example* was created using Macintosh computers, we will quickly review several of the fundamental Macintosh operations common to all application programs for the Macintosh, in particular to *Mathematica*. However, this book is not meant to be an introduction to the Macintosh and the beginning user completely unfamiliar with the Macintosh operating system should familiarize himself with the Macintosh by completing the **Macintosh Tour** and consulting the **Macintosh Reference**. The material that appears in *Mathematica by Example* should be useful to anyone who uses *Mathematica* in a windows environment. Non-Macintosh users may either want to quickly read **Chapter 1** or proceed directly to **Chapter 2**, provided they are familiar with their computer.

After the *Mathematica* program has been properly installed, a user can access *Mathematica* by first clicking twice on the hard disk icon located in the upper right hand corner of the computer screen. The following window will appear:

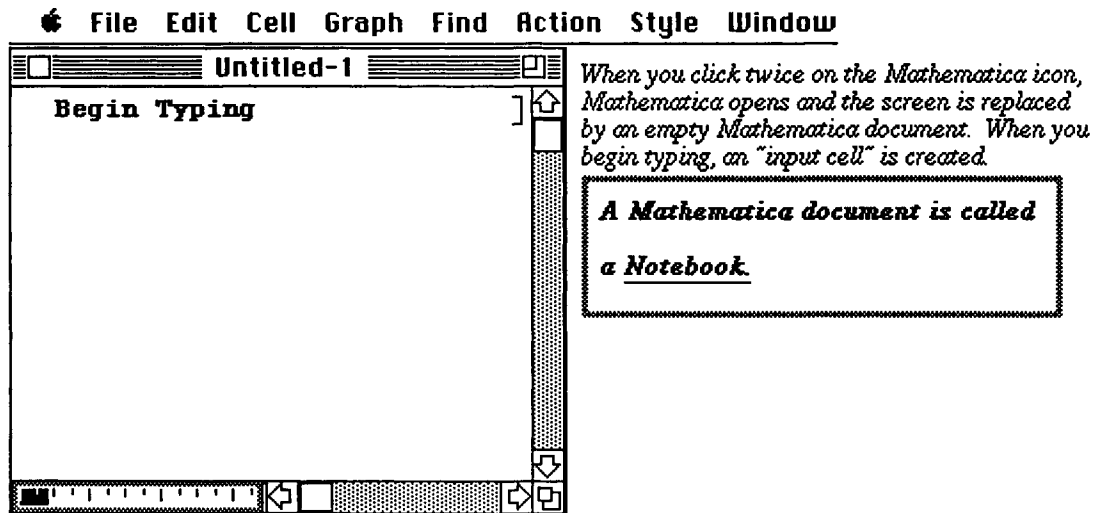


The *Mathematica* f folder can be opened by clicking twice on its icon. After opening the *Mathematica* f folder, start *Mathematica* by double clicking on the icon labeled *Mathematica*. These steps are illustrated below:



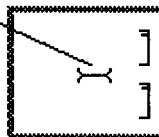
After double-clicking on the *Mathematica* icon, an empty *Mathematica* document appears; the *Mathematica* session can be initiated by typing anything. When you begin typing, *Mathematica* automatically creates an **input cell** for you. If an input cell contains a *Mathematica* command, the command is evaluated by pressing **ENTER** or **Shift-Return**.

In general, the **ENTER** key and **RETURN** key are not the same. The **ENTER** key is used to evaluate *Mathematica* commands; the **RETURN** key gives a new line.

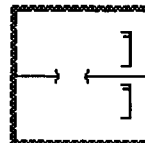


In order to create a new **input cell** move the cursor below the original cell so that the cursor is horizontal. When the cursor is horizontal, click the mouse once:

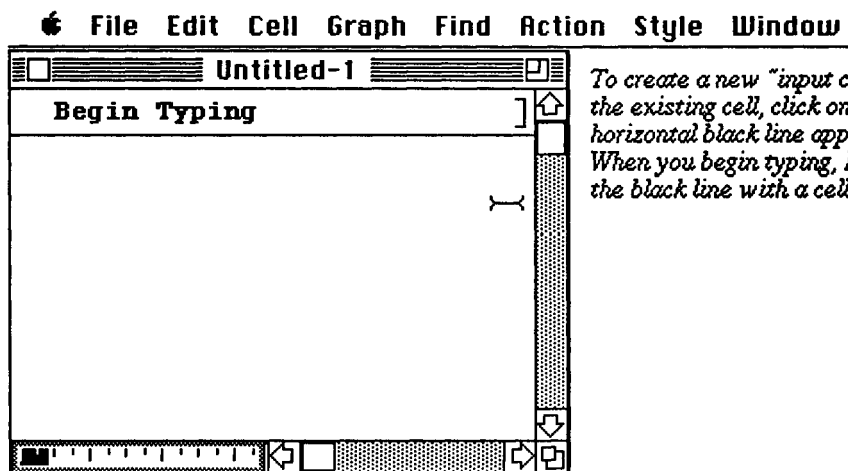
The cursor is horizontal whenever it is between two cells:



When the cursor is horizontal and the mouse is clicked once, a black line appears across the document window:



A horizontal black line appears after clicking the horizontal cursor once. Additional typing causes *Mathematica* to replace this line with a new **input cell** containing the most recently typed information.

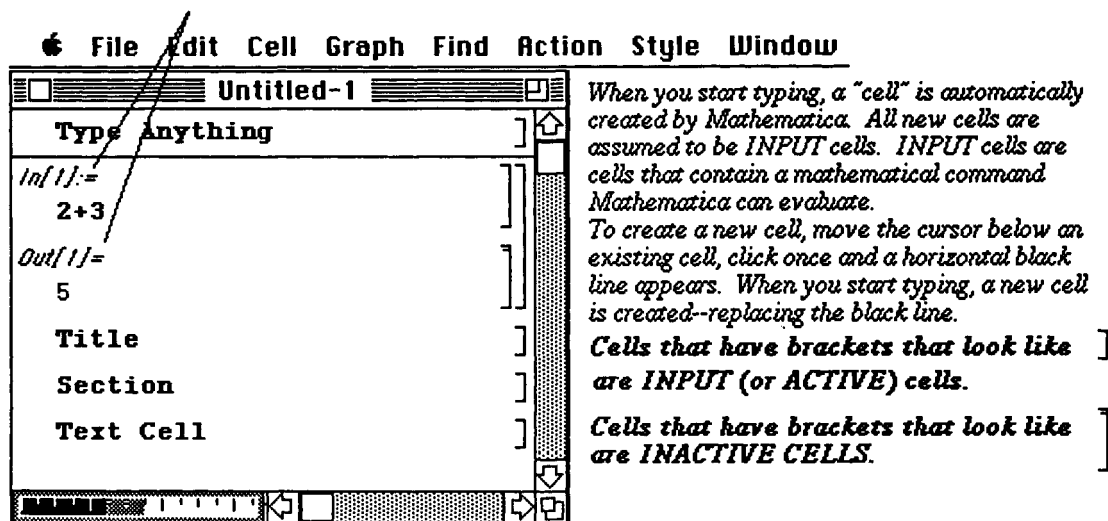


To create a new "input cell", move the cursor below the existing cell, click once. Notice that a horizontal black line appears. When you begin typing, Mathematica replaces the black line with a cell to hold your text.

■ 1.2 Introduction to the Basic Types of Cells, Cursor Shapes, and Evaluating Commands

In the following example, $2+3$ is a *Mathematica* command. The input cell containing $2+3$ can be evaluated by pressing **ENTER** after the command has been typed.

Do NOT type "In(1)" and "Out(1)". Mathematica automatically numbers the calculations for you.



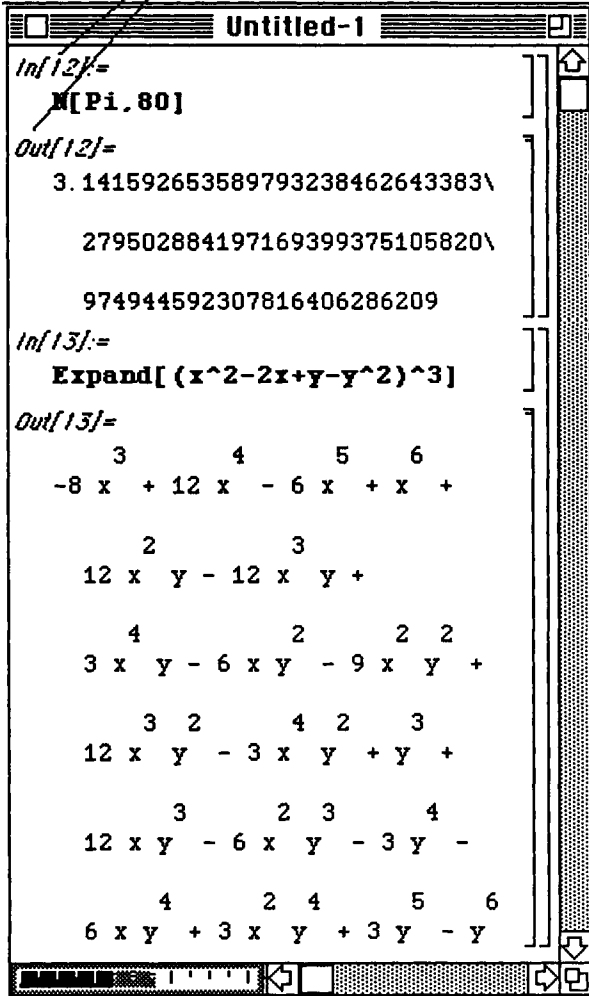
Inactive cells are cells that cannot be evaluated by *Mathematica*. Inactive cells include output cells, graphics cells, and text cells. **Output cells** are cells that contain the results of calculations performed by *Mathematica*; **graphics cells** are cells that contain two- or three-dimensional graphics produced by *Mathematica*; and **text cells** are cells that contain explanations or other written material that cannot be evaluated by *Mathematica*.

To verify that you are able to evaluate input cells correctly, carefully type and **ENTER** each of the following commands:

Notice that every *Mathematica* command begins with capital letters and the argument is enclosed by square brackets "[]".

Do NOT type In() or Out(); Mathematica automatically keeps track of the sequence of performed calculations for you.

File Edit Cell Graph Find Action Style Window



Be sure to type each command EXACTLY as it appears. Pay close attention to square brackets and capital letters.

To execute a Mathematica command, press ENTER. To obtain a new line within an existing cell, press RETURN.

`N[Pi, 80]`

Computes the value of π to 80 digits of accuracy.

`Expand[(x^2-2x+y-y^2)^3]`

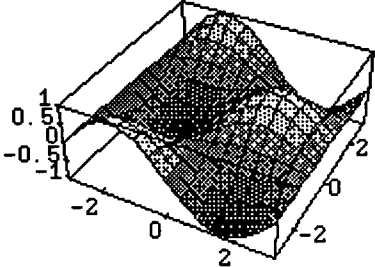
Computes the product $(x^2 - 2x + y - y^2)^3$.

The arrow " \rightarrow " in the following example is obtained by typing the minus key "-" followed by the greater than key ">".

File Edit Cell Graph Find Action Style Window

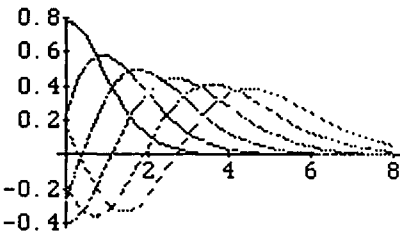
Untitled-1

```
In[14]:=
Plot3D[Sin[x]Cos[y],
{x,-Pi,Pi},{y,-Pi,Pi}]
```



```
Out[14]=
-SurfaceGraphics-
```

```
In[15]:=
table1=Table[BesselJ[x,n],
{n,1,6}];
table2=Table[GrayLevel[j/10]
{j,0.5}];
Plot[Release[table1],
{x,0,8},PlotStyle->
table2]
```



```
Out[15]=
-Graphics-
```

Be sure to type each command EXACTLY as it appears. In particular, pay close attention to capital letters, square brackets, and braces. To obtain a new line within a cell, press RETURN; to evaluate a Mathematica command, or input, press ENTER.

```
Plot3D[Sin[x]Cos[y],
{x,-Pi,Pi},{y,-Pi,Pi}]
```

graphs the function

$f(x,y) = \sin(x)\cos(y)$ on the interval $[-\pi,\pi] \times [-\pi,\pi]$.

BesselJ[x,n] denotes the Bessel function of the first kind,

$$J_n(x) = \sum_{j=0}^{\infty} \frac{(-1)^j}{j!\Gamma(1+j+n)} \left(\frac{x}{2}\right)^{2j+n} \text{ where}$$

$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$ is the Gamma function.

```
table1=Table[BesselJ[x,n],
{n,1,6}];
```

```
table2=Table[GrayLevel[j/10]
{j,0.5}];
```

```
Plot[Release[table1],
{x,0,8},PlotStyle->
table2]
```

creates and graphs, in different shades of gray, a table of Bessel functions of the first kind. This example shows that several Mathematica commands can be combined into a single input cell and executed.

Remember: To execute a command, press ENTER; To obtain a new line, press RETURN.

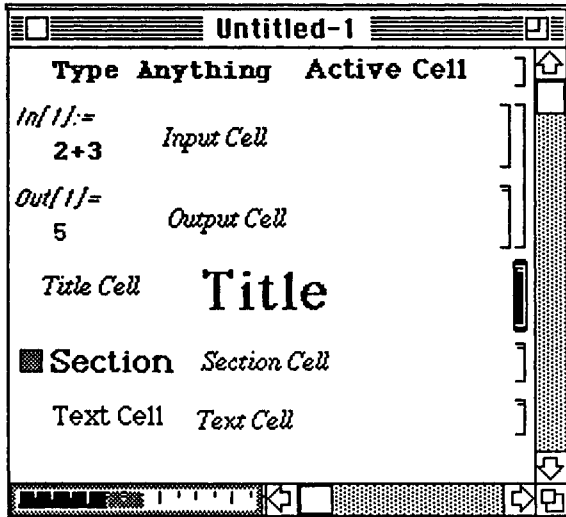
Often when using a notebook, users need to convert active cells to inactive cells. This may be accomplished as follows:

□ **To convert Active Cells to Inactive Cells:**

- 1) Use the mouse to click on the cell bracket of the cell to be modified. The cell bracket will become highlighted.
- 2) Go to **Style** and select **Cell Style**.
- 3) Use the mouse and cursor to choose the desired cell style.

Notice how the cells from the first example have been modified; the Title Cell is highlighted.

⌘ File Edit Cell Graph Find Action Style Window



This cell was changed to a Title Cell;

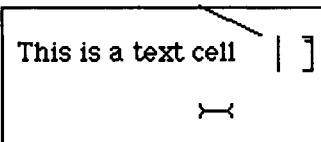
This cell was changed to a Section Cell; and

This cell was changed to a Text Cell.

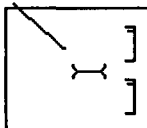
As the cursor is moved within a *Mathematica* notebook, the cursor changes shape. The shape depends on whether (a) the cursor is within an active or inactive cell or (b) the cursor is between two cells.

Cursor Shapes:

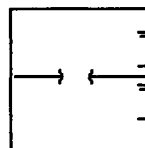
When you click within a text cell, the cursor is vertical. You can then type within the text cell



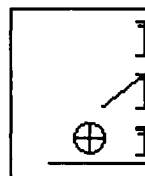
When you are between two cells, the cursor is horizontal.



When you click between two cells, a horizontal black line appears:



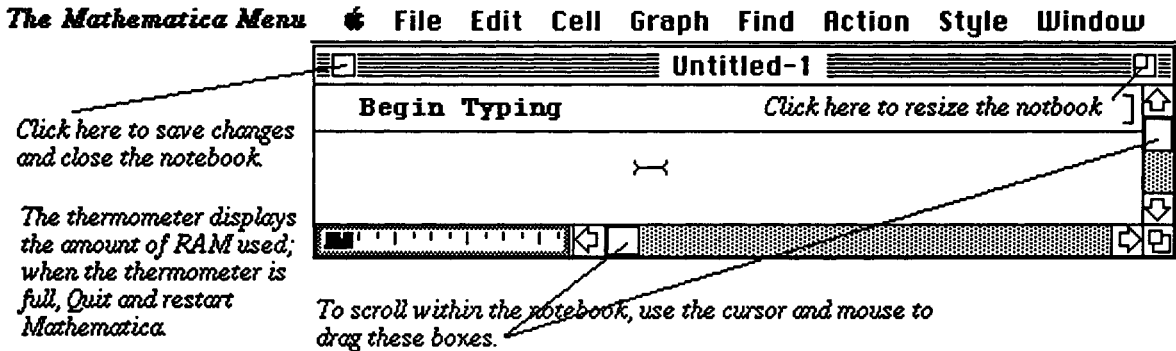
When you are within a graphics cell, a bullseye appears. You cannot write inside a graphics cell



■ 1.3 Introduction to the *Mathematica* Menu

After *Mathematica* has started, the *Mathematica* Menu appears at the top of the screen. The purpose of this section is to introduce the most frequently used operations from the Menu. The Menu will be described in more detail in Chapter 10.

- The Menu discussed here is as it appears in Version 1.2. The Version 2.0 Menu is somewhat different from the Version 1.2 Menu. For a discussion of the Version 2.0 Menu, see Chapter 10.



To use the Menu, use the mouse to move the cursor to either **File**, **Edit**, **Cell**, **Graph**, **Find**, **Action**, **Style**, or **Window**. We briefly describe several of the features available under **File**, **Edit**, **Style**, and **Window**.

Use the mouse to move the cursor to **FILE** in order to create a new Mathematica notebook, Open an existing Mathematica notebook, Save changes to a notebook, Print a notebook, or Quit Mathematica.

File	
New	<i>Creates a new Mathematica Notebook</i>
Open...	<i>Opens an existing Mathematica Notebook</i>
Save	<i>Saves (but does not close) the open Mathematica Notebook</i>
Save As...	
Save As Other...	
Show Page Breaks	<i>Marks where page breaks will occur</i>
Show Keywords	
Show Names	
Page Setup...	<i>Use to specify type of printer and paper used</i>
Printing Options...	<i>Use to modify margins and page numbering</i>
Print...	<i>Prints the open Mathematica Notebook</i>
Print Selection...	<i>Prints highlighted cells</i>
Quit	<i>Saves changes to Mathematica Notebooks then quits Mathematica</i>

To take advantage of the standard Macintosh editing commands (Cut, Copy, Paste) select EDIT. One can also divide a cell into two cells or merge two (or more) cells of the same type into a single cell. The various Mathematica settings will be discussed later.

Edit

Undo/Can't Undo
Cut
Copy
Paste
Clear
Paste and Discard
Convert Clipboard
Select All Cells
Nesting
Divide Cell
Merge Cells
Settings

Highlights all cells

Divides a single cell into two cells

Merges highlighted cells of the same type into a single cell

Contains various startup and display settings for Mathematica

To modify highlighted text or cells, use the mouse to move the cursor to STYLE. Fonts, faces, sizes, color and cell style can be modified.

Style

Font
Face
Size
Color
Format
Cell Style
Uniform Style
Default Styles
All Default Styles

Use to change highlighted text to different fonts

Convert highlighted text to italics, bold, or underline

Change size of highlighted text

Change color of highlighted text

Format

Change cell style of highlighted cells

WINDOW lists all open notebooks, several options for viewing several open notebooks simultaneously, and contains lists of the various Mathematica defaults and styles which will be discussed in detail later.

Window

Stack Windows
Tile Windows Wide
Tile Windows Tall
Network Window
Defaults
Styles
Clipboard
(Open Files)

Various ways of viewing several open notebooks simultaneously.

Mathematica displays a list of the open notebooks

■ **Preview:**

In order for the *Mathematica* user to take full advantage of the capabilities of this software, an understanding of its syntax is imperative. The goal of *Mathematica by Example* is to introduce the reader to the *Mathematica* commands and sequences of commands most frequently used by beginning users. Although all of the rules of *Mathematica* syntax are far too numerous to list here, knowledge of the following five rules equips the beginner with the necessary tools to start using the *Mathematica* program with little trouble.

■ **Remember these Five Basic Rules of Mathematica Syntax**

- **1. The ARGUMENTS of functions are given in square brackets.**
- **2. The NAMES of built-in functions have their first letters capitalized.**
- **3. Multiplication is represented by a space.**
- **4. Powers are denoted by a ^.**
- **5. If you get no response or an incorrect response, you have entered or executed the command incorrectly.**

Chapter 2

Mathematical Operations on Numbers, Expressions and Functions in *Mathematica*

- Chapter 2 introduces the essential commands of *Mathematica*. Basic operations on numbers, expressions, and functions are introduced and discussed.
- Commands introduced and discussed in this chapter from Version 1.2 are:

Operations:

+
-
*
^
/

Constants:

E
I
Pi

Built-In Functions:

N[number]
number // **N**
N[number,digits]
Abs[number]
Sqrt[number]
Exp[number]
Sin[number]
Cos[number]
Tan[number]
ArcCos[number]
ArcSin[number]
ArcTan[number]
Log[b]
Log[a,b]
Mod[a,b]
Prime[n]

Operations on Equations:

Solve
NRoots
FindRoot

Operations on Expressions and Functions:

Simplify[expression]
Factor[expression]
Expand[expression]
Modulus->p
Together[expression]
Apart[expression]
Numerator[fraction]
Denominator[fraction]
Cancel[expression]
Clear[functions]
Compose[function1, function2, . . . ,x]
Nest[function,n,x]

Evaluation:

expression /. variable->number
Out[n]

@

Graphics:

Plot[f[x],{x,a,b},options] or **Plot**[[f[x], g[x], . . .],{x,a,b},options]

Options:

PlotStyle
DisplayFunction
AspectRatio
Framed
Ticks
AxesLabel
PlotLabel
GrayLevel[number]
RGBColor[number(1),number(2),number(3)]
PlotRange->{a,b}
Axes->{a,b}
Show[graphics,options]

- Commands introduced and discussed in this chapter from Version 2.0 are:

Operations on Expressions and Functions:

Composition[function1,function2,...,functionn][x]

ComplexExpand[expression]

PolynomialMod[poly,p]

Graphics:

GraphicsArray[{{graph1.1,graph1.2,...,graph1.n},
 {graph2.1,...,graph2.n},...,
 {graphn.1,...,graphn.n}}]

Rectangle[{xmin,ymin},{xmax,ymax},graphics]

Options:

Background

GridLines

Frame

DefaultFont

PlotLabel→FontForm

- Application: Locating intersection points of graphs of functions

■ 2.1 Numerical Calculations and Built-In Functions

■ Numerical Calculations and Built-In Constants

The basic arithmetic operations (addition, subtraction, multiplication, and division) are performed in the natural way with *Mathematica*. Whenever possible, *Mathematica* gives an exact answer and reduces fractions:

"a plus b" is entered as $a+b$;

"a minus b" is entered as $a-b$;

"a times b" is entered as either $a*b$ or $a\ b$ (note the space between a and b); and

"a divided by b" is entered as a/b . Executing the command a/b results in a reduced fraction.

Do NOT type "In" and "Out". Mathematica automatically numbers the calculations for you.

File Edit Cell Graph Find Action Style Window

Untitled-1

In[14]:=

121+542

Out[14]=

663

In[15]:=

3231-9876

Out[15]=

-6645

In[16]:=

-23*76

Out[16]=

-1748

In[17]:=

22361 832748 387281

Out[17]=

7211589719761868

In[18]:=

467/31

Out[18]=

467

--

31

Mathematica computes basic operations on numbers in the usual way.

Mathematica assumes all cells are INPUT cells. INPUT cells are cells that contain a command that Mathematica can execute. To execute a command, press ENTER, or equivalently, Shift-RETURN. In general, the RETURN key gives you a new line; the ENTER key evaluates a Mathematica command.

*The symbol * denotes multiplication. However, a space between two expressions also denotes multiplication.*

OUTPUT cells are not ACTIVE cells. They cannot be evaluated since they do not contain a command Mathematica can evaluate.

Mathematica will usually give exact answers.

The symbol / denotes division. Instead of yielding a decimal approximation, Mathematica gives the exact fraction as output.

a^b , "a raised to the bth power", is entered as a^b .

$\sqrt{a} = a^{1/2}$ can be evaluated as either $a^{(1/2)}$ or `Sqrt [a]`; $\sqrt[3]{a} = a^{1/3}$ can be evaluated by $a^{(1/3)}$.

File Edit Cell Graph Find Action Style Window

```

RoutineCalculation
In[1]:=
(-5)^121
Out[1]=
-3761581922631320025499956919\
11118616901972978167068006\
88280054600909352302551269\
53125
In[2]:=
(-5)^(1/9)
Out[2]=
1/9
(-5)
    
```

When Mathematica computes $(-5)^{121}$ it gives an exact number.

The \ indicates that the output continues onto the next line.

However, when Mathematica computes $\sqrt[9]{-5} = (-5)^{1/9}$ the result is an irrational number.

Notice that *Mathematica* gives an exact answer whenever possible. For a variety of reasons, however, numerical approximations of results are either more meaningful or more desirable. The command used to obtain a numerical approximation of the number **a**, is **N[a]** or equivalently

a // N. The command to obtain a numerical approximation of **a** to **n** digits of precision is **N[a, n]**.

The exact values computed in the previous window are approximated numerically below:

```

In[3]:=
  N[(-5)^(1/9)]
Out[3]=
  -1.19581
In[4]:=
  N[(-5)^121]
Out[4]=
           84
  -3.76158 10
In[43]:=
  Sqrt[233]
Out[43]=
  Sqrt[233]
In[44]:=
  Sqrt[233] // N
Out[44]=
  15.2643

```

To numerically approximate an expression, use the command **N[expression]** or **expression // N**

N[(-5)^121] converts $(-5)^{121}$ to scientific notation.

To obtain a numerical approximation of

$$\sqrt{233} = (233)^{1/2}$$

ENTER **Sqrt[233] // N** OR

N[Sqrt[233]].

Note that Sqrt[number] produces the same output as (number)^(1/2).

Mathematica has built-in definitions of many commonly used constants. In particular,

e is denoted by **E**; π is denoted by **Pi**; and $i = \sqrt{-1}$ is denoted by **I**.

File Edit Cell Graph Find Action Style Window

```

RoutineCalculation
In[5]:=
N[E, 50]
Out[5]=
2.718281828459045235360287471\
3526624977572470937

In[6]:=
N[Pi, 25]
Out[6]=
3.141592653589793238462643

In[25]:=
Sqrt[-9]
Out[25]=
3 I

In[27]:=
(1-I)^4
Out[27]=
-4

In[28]:=
(3+I)/(4-I)
Out[28]=
11 7 I
-- + ---
17 17
    
```

E denotes the constant e .

N[E, 50] yields a fifty digit approximation of e .

Pi denotes the constant π .

N[Pi, 25] calculates a twenty-five digit approximation of π .

The symbol **I** denotes $i = \sqrt{-1}$.

This command computes $(1-i)^4$

This writes the complex number $\frac{3+i}{4-i}$ in standard form.

■ Built-In Functions

Mathematica recognizes numerous built-in functions. These include the exponential function, **Exp**[*x*]; the absolute value function, **Abs**[*x*]; the trigonometric functions **Sin**[*x*], **Cos**[*x*], **Tan**[*x*], **Sec**[*x*], **Csc**[*x*], and **Cot**[*x*]; and the inverse trigonometric functions **ArcCos**[*x*], **ArcSin**[*x*], **ArcTan**[*x*], **ArcSec**[*x*], **ArcCsc**[*x*], and **ArcCot**[*x*]. Notice that each of these functions is capitalized and uses square brackets.

(Note that the inverse trigonometric functions include two capital letters!) If both of these requirements are not met, then *Mathematica* will not recognize the built-in function and undesirable results will be obtained.

□ The Absolute Value, Exponential and Logarithmic Functions

Calculations involving the functions **Abs**[*x*], **Exp**[*x*], and **Log**[*x*] appear in the following windows. Notice that in order to obtain a numerical value of **Exp**[*x*], a numerical approximation must be requested by either the command **N**[**Exp**[*x*]] or **Exp**[*x*]/**N**. Otherwise, the exact value is given which, in many cases, is not as useful as the numerical approximation.

File Edit Cell Graph Find Action Style Window

<pre> In[7]:= Exp[-5] Out[7]= -5 E In[8]:= Exp[-5] // N Out[8]= 0.00673795 In[2]:= Abs[-5] Out[2]= 5 In[3]:= Abs[14] Out[3]= 14 </pre>	<p>To compute $\frac{1}{e^5} = e^{-5}$ ENTER either Exp[-5] or equivalently E⁽⁻⁵⁾.</p> <p>Exp[-5] // N numerically approximates the irrational number $\frac{1}{e^5} = e^{-5}$; the identical result would be produced by the commands N[Exp[-5]] OR N[E⁽⁻⁵⁾].</p> <p>Abs[-5] computes -5 .</p> <p>Abs[14] computes 14 .</p>
---	--

In addition to real numbers, the function **Abs[x]** can be used to find the absolute value of the complex number $a+bi$, where $\text{Abs}[a+bi] = \text{Sqrt}[a^2+b^2]$.

```
In[5]:=
  Abs[3-4I]
Out[5]=
  5
In[7]:=
  Abs[(3+2I)/(2-9I)]
Out[7]=
  Sqrt[13]
  -----
  Sqrt[85]
```

Abs[3-4I] computes $|3-4i|$.

Abs[(3+2I)/(2-9I)] computes $\left| \frac{3+2i}{2-9i} \right|$.

Log[x] computes the natural logarithm of x which is usually denoted as either $\text{Ln}(x)$ or $\text{Log}_e(x)$:

```
LogsandExponents
In[8]:=
  Log[E]
Out[8]=
  1
In[9]:=
  Log[E^3]
Out[9]=
  3
In[10]:=
  Exp[Log[Pi]]
Out[10]=
  Pi
```

Log[E]
computes $\text{Ln}(e) = 1$.

Log[E^3]
computes $\text{Ln}(e^3) = 3\text{Ln}(e) = 3$.

Exp[Log[Pi]]
computes $e^{\text{Ln}(\pi)} = \pi$.

Log [a, b] computes $\text{Log}_b(a) = \frac{\text{Ln}(a)}{\text{Ln}(b)}$:

```

In[11]:=
  Log[3, 9]
Out[11]=
  2

In[12]:=
  Log[2, 10]
Out[12]=
  Log[10]
  -----
  Log[2]

In[13]:=
  N[Log[2, 10], 10]
Out[13]=
  3.321928095
  
```

Log[3, 9]
computes $\text{Log}_3(9) = 2$.

Log[2, 10]
computes $\text{Log}_2(10) = \frac{\text{Ln}(10)}{\text{Ln}(2)}$.

N[Log[2, 10], 10]
computes the numerical value of
 $\text{Log}_2(10) = \frac{\text{Ln}(10)}{\text{Ln}(2)}$ to ten decimal places.

□ Trigonometric Functions

Examples of typical operations involving the trigonometric functions **Sin[x]**, **Cos[x]**, and **Tan[x]** are given below. (Although not illustrated in the following examples, the functions **Sec[x]**, **Csc[x]**, and **Cot[x]** are used similarly.) Notice that *Mathematica* yields the exact value for trigonometric functions of some angles, while a numerical approximation must be requested for others.

```

RoutineCalculation
In[1]:=
  Cos[Pi/4]
Out[1]=
  Sqrt[2]
  -----
  2
In[2]:=
  Sin[Pi/3]
Out[2]=
  Sqrt[3]
  -----
  2
In[3]:=
  Tan[3 Pi/4]
Out[3]=
  -1
In[4]:=
  Cos[Pi/12]
Out[4]=
  Pi
  Cos[--]
  12

```

Mathematica gives exact values of the standard trigonometric functions. If the value is not well known, it is necessary to request a numerical approximation.

Notice that every built-in Mathematica function begins with a capital letter and the argument is enclosed in square brackets.

Since the numerical value of $\text{Cos}\left[\frac{\pi}{12}\right]$ is not well-known, a numerical approximation must be requested.

Even though *Mathematica's* built-in functions cannot compute exact values of

$\text{Cos}\left(\frac{\pi}{12}\right)$ and $\text{Sin}\left(\frac{-9\pi}{8}\right)$, numerical approximations can be obtained by entering

N[Cos [Pi/12]] or **Sin [-9Pi/8] // N.**

```

In[5]:=
  N[Cos [Pi/12]]
Out[5]=
  0.965926

In[6]:=
  Sin [-9 Pi/8]
Out[6]=
  -9 Pi
  Sin[-----]
      8

In[7]:=
  Sin [-9 Pi/8] // N
Out[7]=
  0.382683
  
```

Similarly, to obtain a numerical value of $\text{Sin}\left[\frac{-9\pi}{8}\right]$ a numerical approximation must be requested.

□ Inverse Trigonometric Functions

Commands involving the inverse trigonometric functions are similar to those demonstrated in the earlier section on trigonometric functions. Again, note the two capital letters in each of the inverse trigonometric functions. The (built-in) inverse trigonometric functions are:

- (i) `ArcCos[x]`; (ii) `ArcCoth[x]`; (iii) `ArcSec[x]`; (iv) `ArcSinh[x]`;
 (v) `ArcCosh[x]`; (vi) `ArcCsc[x]`; (vii) `ArcSech[x]`; (viii) `ArcTan[x]`;
 (ix) `ArcCot[x]`; (x) `ArcSch[x]`; (xi) `ArcSin[x]`; and (xii) `ArcTanh[x]`.

```

RoutineCalculation
In[2]:=
  ArcCos[1/2]
Out[2]=
  Pi
  --
  3
In[3]:=
  ArcSin[-1]
Out[3]=
  -Pi
  ---
  2
In[4]:=
  ArcTan[1]
Out[4]=
  Pi
  --
  4
    
```

Notice that the inverse trigonometric functions are built-in Mathematica functions. When possible, exact values are given.

In most instances, a numerical approximation must be requested:

```

In[5]:=
  ArcSin[1/3] // N
Out[5]=
  0.339837
In[6]:=
  N[ArcCos[2/3]]
Out[6]=
  0.841069
In[7]:=
  ArcTan[100] // N
Out[7]=
  1.5608
    
```

Since `ArcSin[1/3]` is not well known, a numerical approximation is obtained.

Notice that `N[ArcCos[2/3]]` gives the same numerical approximation to `ArcCos[2/3]` as `ArcCos[2/3] // N` if it were evaluated.

■ 2.2 Expressions and Functions

■ Basic Algebraic Operations on Expressions

Mathematica performs standard algebraic operations on mathematical expressions. For example, the command **Factor**[*expression*] factors *expression*; **Expand**[*expression*] multiplies *expression*; **Together**[*expression*] writes *expression* as a single fraction.

The screenshot shows a Mathematica notebook window titled "OperationsonExpressions". It contains three input-output pairs:

- Example 1:**

In[27]:= **Factor**[12x²+27 x y-84y²]

Out[27]= 3 (4 x - 7 y) (x + 4 y)

Factor[12x²+27 x y-84y²] factors the polynomial 12x²+27xy-84y².
- Example 2:**

In[28]:= **Expand**[(x+y)² (3x-y)³]

Out[28]=

$$27 x^5 + 27 x^4 y - 18 x^3 y^2 - 10 x^2 y^3 + 7 x y^4 - y^5$$

Expand[(x+y)² (3x-y)³] computes the product (x+y)²(3x-y)³.
- Example 3:**

In[29]:= **Together**[2/x² - x²/2]

Out[29]=

$$\frac{4 - x^4}{2x^2}$$

Together[2/x² - x²/2] writes the expression $\frac{2}{x^2} - \frac{x^2}{2}$ as a single fraction.

A callout box with a dotted border contains the text: "Don't forget the space between the x and the y. Remember: Multiplication of two expressions is denoted by a space. Hence, xy is an expression xy while x y denotes x multiplied by y."

In general, a space is not needed between a number and a symbol to denote multiplication. That is, **3dog** means "3 times variable **dog**"; *Mathematica* interprets **3 dog** the same way. However, when denoting multiplication of two variables, either include a space or *: **cat dog** means "variable **cat** times variable **dog**", **cat*dog** means the same thing but **catdog** is interpreted as a variable **catdog**.

The command **Apart**[*expression*] computes the partial fraction decomposition of *expression*; **Cancel**[*expression*] factors the numerator and denominator of *expression* then reduces *expression* to lowest terms.

The screenshot shows a Mathematica notebook interface with two input-output pairs. The first pair shows the **Apart** command being used on the expression $1/((x-3)(x-1))$, resulting in a partial fraction decomposition. The second pair shows the **Cancel** command being used on the expression $(x^2-1)/(x^2-2x+1)$, resulting in a simplified fraction.

In[30]:=
Apart[1/((x-3)(x-1))]
Out[30]=

$$\frac{1}{2(-3+x)} - \frac{1}{2(-1+x)}$$

In[31]:=
Cancel[(x^2-1)/(x^2-2x+1)]
Out[31]=

$$\frac{1+x}{-1+x}$$

Apart[1/((x-3)(x-1))]
 performs the partial fraction decomposition
 on the expression $\frac{1}{(x-3)(x-1)}$.

Cancel[(x^2-1)/(x^2-2x+1)]
 simplifies the fraction $\frac{x^2-1}{x^2-2x+1}$
 by factoring and reducing to lowest terms.

■ Naming and Evaluating Expressions

In *Mathematica*, mathematical objects can be named. Naming objects is convenient: we can avoid typing the same mathematical expression repeatedly and named expressions can be referenced throughout a notebook.

Since every built-in *Mathematica* function begins with a capital letter, we will adopt the convention that every mathematical object we name will begin with a lower-case letter. Consequently, we will be certain to avoid any possible ambiguity with a built-in *Mathematica* object. An expression is named by using a single equals sign (=).

Expressions can be evaluated easily. To evaluate an expression we introduce the command /. . The command /. means "replace by". For example, the command $x^2 /. x \rightarrow 3$ means evaluate the expression

x^2 when $x = 3$.

The following example illustrates how to name an expression. In addition, *Mathematica* has several built-in functions for manipulating fractions:

- 1) **Numerator**[*fraction*] yields the numerator of a fraction; and
- 2) **Denominator**[*fraction*] yields the denominator of a fraction.

The naming of expressions makes the numerator and denominator easier to use in the following examples:

```

NamingExpressions
In[19]:=
fraction=(x^3+2x^2-x-2)/(x^3+x^2-4x-4)
Out[19]=
      2 3
-2 - x + 2 x + x
-----
      2 3
-4 - 4 x + x + x

In[20]:=
num=Numerator[fraction]
Out[20]=
      2 3
-2 - x + 2 x + x

In[21]:=
Factor[num]
Out[21]=
(-1 + x) (1 + x) (2 + x)

In[22]:=
num /. x->2
Out[22]=
12

In[23]:=
den=Denominator[fraction]
Out[23]=
      2 3
-4 - 4 x + x + x

In[24]:=
Factor[den]
Out[24]=
(-2 + x) (1 + x) (2 + x)

In[25]:=
den /. x->3
Out[25]=
20

```

The expression $\frac{x^3+2x^2-x-2}{x^3+x^2-4x-4}$

is named **fraction**.

Numerator[fraction] yields the numerator of **fraction**; the numerator is named **num**.

Factor[num] factors **num**.

num /. x->2 evaluates **num** when $x=2$.

Denominator[fraction] yields the denominator of **fraction**; the denominator is named **den**.

Factor[den] factors **den**.

den /. x->3 evaluates **den** when $x=3$.

Mathematica can also evaluate and perform standard algebraic operations on named expressions:

NamingExpressions		
<pre>In[26]:= Cancel[fraction]</pre>	<p>Cancel[fraction] <i>factors the numerator and denominator of fraction then simplifies.</i></p>	
<pre>Out[26]= -1 + x ----- -2 + x</pre>		
<pre>In[27]:= fraction /. x->4</pre>		<p>fraction /. x->4 <i>evaluates fraction when x=4.</i></p>
<pre>Out[27]= 3 - 2</pre>		
<pre>In[28]:= fraction /. x->-3</pre>	<p>fraction /. x->-3 <i>evaluates fraction when x=-3.</i></p>	
<pre>Out[28]= 4 - 5</pre>		
<pre>In[29]:= Apart[fraction]</pre>	<p>Apart[fraction] <i>performs the partial fraction decomposition on fraction.</i></p>	
<pre>Out[29]= 1 1 + ----- -2 + x</pre>		

Every *Mathematica* object can be named; even graphics and functions can be named with *Mathematica*.

■ Defining and Evaluating Functions

It is important to remember that functions, expressions, and graphics can be named anything that is not the name of a built-in *Mathematica* function or command. Since every built-in *Mathematica* function begins with a capital letter, every user-defined function or expression in this text will be defined using lower case letters. This way, the possibility of conflicting with a built-in *Mathematica* command or function is completely eliminated. Also, since definitions of functions are frequently modified, we introduce the command **Clear**. **Clear[expression]** clears all definitions of **expression**. Consequently, we are certain to avoid any ambiguity when we create a new definition of a function. When you **first** define a function, you must always enclose the argument in square brackets and place an underline **after** the argument on the left-hand side of the equals sign in the definition of the function.

□ Example:

Use *Mathematica* to define $f(x) = x^2$, $g(x) = \sqrt{x}$, and $h(x) = x + \sin(x)$.

The screenshot shows a Mathematica notebook titled "DefiningFunctions" with the following content:

```

In[10]:=
Clear[f,g,h]
f[x_]=x^2

Out[10]=
2
x

In[11]:=
g[x_]=Sqrt[x]

Out[11]=
Sqrt[x]

In[12]:=
h[x_] := x + Sin[x]

In[13]:=
?h

h
h/: h[x_] := x + Sin[x]

```

Callouts and annotations:

- Clear[f,g,h]** clears all prior definitions of $f, g,$ and h . Consequently, we are sure to avoid any ambiguity if $f, g,$ and h have been used previously in the notebook.
- $f[x_]=x^2$ defines $f(x)$ to be the function $f(x) = x^2$.
- Notice the underline ("_") on the left-hand side of the definition of $f(x)$ does NOT appear on the right-hand side. The underline MUST be included on the left-hand side of the equals sign and NOT included on the right-hand side.
- $g[x_]=Sqrt[x]$ defines $g(x)$ to be the function $g(x) = \sqrt{x}$.
- $h[x_] := x + Sin[x]$ defines $h(x)$ to be the function $h(x) = x + \sin(x)$.
- Notice that the `:=` prevents Mathematica from showing the definition of $h(x)$ after it is entered; nevertheless, the command `?h` shows the definition of $h(x)$.
- Don't forget to include the underline ("_") on the left-hand side of the equals sign in the definition of a function. Remember to ALWAYS include arguments of functions in square brackets.

When you evaluate a function, type **functionname**[point] ENTER. Notice that functions can be evaluated for any real number (in the function's domain):

□ Example:

Using the definitions of f, g, and h from above, compute f(2), g(4) and h($\pi/2$).

<pre>In[14]:= f[2]</pre>	<pre>]]</pre>	<p><i>f[2] evaluates the function f at x=2.</i></p>
<pre>Out[14]= 4</pre>	<pre>]]</pre>	
<pre>In[19]:= g[4]</pre>	<pre>]]</pre>	<p><i>g[4] evaluates the function g at x=4.</i></p>
<pre>Out[19]= 2</pre>	<pre>]]</pre>	
<pre>In[20]:= h[Pi/2]</pre>	<pre>]]</pre>	<p><i>h[Pi/2] evaluates the function h at x=$\frac{\pi}{2}$.</i></p>
<pre>Out[20]= Pi 1 + -- 2</pre>	<pre>]]</pre>	

Moreover, *Mathematica* can symbolically evaluate and manipulate functions.

□ Example:

Several examples follow which involve the function $f(x) = x^2$ defined above.

<pre>In[23]:= f[a-b^2] Out[23]= 2 2 (a - b) In[24]:= Expand[f[a-b^2]] Out[24]= 2 2 4 a - 2 a b + b In[34]:= (f[x+h]-f[x])/h Out[34]= 2 2 -x + (h + x) ----- h In[35]:= Simplify[(f[x+h]-f[x])/h] Out[35]= h + 2 x</pre>	<p>$f[a-b^2]$ evaluates $f(a-b^2)$.</p> <p>Expand[f[a-b²]] computes $f(a-b^2)$ and then expands the resulting product.</p> <p>$(f[x+h]-f[x])/h$ computes the quotient $\frac{f(x+h)-f(x)}{h}$.</p> <p>Notice that RETURN gives a new line; while ENTER (or SHIFT-RETURN) evaluates an input cell.</p> <p>On the other hand, Simplify[(f[x+h]-f[x])/h] computes and simplifies $\frac{f(x+h)-f(x)}{h}$.</p>
---	--

Many different types of functions can be defined using *Mathematica*. An example of a function f of two variables is illustrated below.

Additional ways of defining functions will be discussed in later parts of this text.

□ Example:

Define $f(x,y) = 1 - \sin(x^2 + y^2)$. Compute $f(1,2)$, $f\left(2\sqrt{\pi}, \frac{3}{2}\sqrt{\pi}\right)$, $f(0,a)$, and $f(a^2 - b^2, b^2 - a^2)$.

DefiningFunctions	
<code>In[19]:=</code> <code>f[x_,y_]=1-Sin[x^2+y^2]</code>	<code>f[x_,y_]=1-Sin[x^2+y^2]</code> defines the function of two variables $f(x,y) = 1 - \sin(x^2 + y^2)$.
<code>Out[19]=</code> $1 - \sin[x^2 + y^2]$	
<code>In[23]:=</code> <code>f[1,2]</code>	<code>f[1,2]</code> computes $f(1,2)$.
<code>Out[23]=</code> $1 - \sin[5]$	
<code>In[24]:=</code> <code>f[2 Sqrt[Pi], 3/2 Sqrt[Pi]]</code>	<code>f[2 Sqrt[Pi], 3/2 Sqrt[Pi]]</code> computes $f\left(2\sqrt{\pi}, \frac{3}{2}\sqrt{\pi}\right)$.
<code>Out[24]=</code> $1 - \frac{\text{Sqrt}[2]}{2}$	
<code>In[25]:=</code> <code>f[0,a]</code>	<code>f[0,a]</code> computes $f(0,a)$.
<code>Out[25]=</code> $1 - \sin[a^2]$	

Evaluating $f(a^2 - b^2, b^2 - a^2)$ is done the same way as in the previous examples:

<code>In[26]:=</code> <code>f[a^2-b^2,b^2-a^2]</code>	<code>f[a^2-b^2,b^2-a^2]</code> computes $f(a^2 - b^2, b^2 - a^2)$.
<code>Out[26]=</code> $1 - \sin[(a^2 - b^2)^2 + (-a^2 + b^2)^2]$	

Vector-valued functions, such as g below, can also be defined:

□ Example:

Define the vector-valued function $g(x) = \{x^2, 1-x^2\}$; compute $g(1)$ and $g(\sin(b))$.

<pre>In[27]:= g[x_]= {x^2, 1-x^2} Out[27]= 2 2 {x , 1 - x } In[28]:= g[1] Out[28]= {1., 0}</pre>	<p>$g[x_]=\{x^2, 1-x^2\}$ defines the vector function $g(x) = \{x^2, 1-x^2\}$.</p> <p>$g[1]$ computes $g(1) = \{1^2, 1-1^2\}$.</p>
--	--

In any case, don't forget to include the underline " " after each variable on the left hand side of the definition of the function. Do not use the underline in any other case.

$g(\sin(b))$ is computed the same way:

<pre>In[30]:= g[Sin[b]] Out[30]= 2 2 {Sin[b] , 1 - Sin[b] }</pre>	<p>$g[\text{Sin}[b]]$ computes $g(\sin(b)) = \{\sin^2(b), 1 - \sin^2(b)\}$.</p>
--	---

□ Example:

Define the vector-valued function of two variables $h(x,y) = \{\cos(x^2 - y^2), \sin(y^2 - x^2)\}$.

```

In[31]:=
  h[x_, y_] = {Cos[x^2 - y^2],
               Sin[y^2 - x^2]}
Out[31]=
  {Cos[x^2 - y^2], Sin[-x^2 + y^2]}
In[32]:=
  h[1, 2]
Out[32]=
  {Cos[-3], Sin[3]}
In[33]:=
  h[Pi, -Pi]
Out[33]=
  {1, 0}
In[34]:=
  h[-Pi, Pi]
Out[34]=
  {1, 0}
In[35]:=
  h[Cos[a^2], Cos[1 - a^2]]
Out[35]=
  {Cos[Cos[a^2] - Cos[1 - a^2]],
   Sin[-Cos[a^2] + Cos[1 - a^2]]}

```

$h[x_, y_] = \{\text{Cos}[x^2 - y^2], \text{Sin}[y^2 - x^2]\}$
 defines the function
 $h(x,y) = \{\cos(x^2 - y^2), \sin(y^2 - x^2)\}$.
 Notice that h is a function of two variables that has a range consisting of ordered pairs. We will see that many types of functions can be defined with Mathematica.
 $h[1, 2], h[\text{Pi}, -\text{Pi}], h[-\text{Pi}, \text{Pi}]$, and $h[\text{Cos}[a^2], \text{Cos}[1 - a^2]]$ calculate $h(1,2), h(\pi, -\pi), h(-\pi, \pi)$, and $h(\cos(a^2), \cos(1 - a^2))$, respectively.

■ Additional Ways to Evaluate Functions and Expressions

Not only can a function $f[x]$ be evaluated by computing $f[a]$ where a is either a real number in the domain of f or an expression, functions and expressions can be evaluated using the command /. . . In general, to evaluate the function $f[x]$ when x is replaced by **expression**, the following two commands are equivalent and yield the same output:

- 1) $f[\text{expression}]$ replaces each variable in f by **expression**; and
- 2) $f[x] /. x \rightarrow \text{expression}$ replaces each variable x in $f[x]$ by **expression**.

□ Example:

```

DefiningFunctions
In[43]:=
Clear[f,g]
f[x_,y_]:=x^2+y^2
g[x_,y_] := {Sin[x^2-y^2],
             Cos[y^2-x^2]}

In[46]:=
f[1,2]

Out[46]=
5

In[47]:=
g[1,2]

Out[47]=
{Sin[-3], Cos[3]}

In[48]:=
f[x,y] /. x->1 /. y->2

Out[48]=
5

In[49]:=
g[x,y] /. x->1 /. y->2

Out[49]=
{Sin[-3], Cos[3]}

```

Before defining new functions f and g , first clear all prior definitions.

Then define

$$f(x,y) = x^2 + y^2; \quad g(x,y) = \{\sin(x^2 - y^2), \cos(y^2 - x^2)\}$$

$f[1,2]$ computes $f(1,2)$.

$g[1,2]$ computes $g(1,2)$.

$f[x,y] /. x \rightarrow 1 /. y \rightarrow 2$ computes $f(x,y)$, replaces x by 1, and then replaces y by 2.

Notice that the result is **EXACTLY** the same as $f[1,2]$

$g[x,y] /. x \rightarrow 1 /. y \rightarrow 2$ computes $g(x,y)$, replaces x by 1, and then replaces y by 2.

Notice that the result is **EXACTLY** the same as $g[1,2]$.

There are several other methods available for evaluating functions. However, depending on the situation, one method may prove to be more appropriate than others. Some of these methods are discussed here in order to make the reader aware of alternate approaches to function evaluation. In the example which follows, a function f is defined which maps a list of two elements, $\{a, b\}$, to the real number, a Modulo b using the built-in function Mod . If a and b are real numbers, $\text{Mod}[a, b]$ returns a modulo b . The typical approach to evaluating f at $\{a, b\}$ is to directly substitute $\{a, b\}$ into f with $f[\{a, b\}]$. However, two another approaches which yield the same result are $f@ \{a, b\}$ and $\{a, b\} // f$. These are demonstrated below with $\{5, 3\}$.

```

In[110]:=
Clear[f]
f[{a_, b_}] := Mod[a, b]

In[111]:=
f[{a, b}]

Out[111]=
Mod[a, b]

In[112]:=
f[{5, 3}]

Out[112]=
2

In[113]:=
f @ {5, 3}

Out[113]=
2

In[114]:=
{5, 3} // f

Out[114]=
2

```

Mod[a, b]
returns a Mod b.

All three of these commands compute $f(\{5, 3\}) = 2$.

■ Retrieving Unnamed Output

Although naming *Mathematica* objects is convenient, occasionally, one may want to use previous results in subsequent calculations even though these objects were not necessarily named. Fortunately, *Mathematica* provides two convenient ways to refer to previously generated output. First, the symbol `%` refers to the most recent output; `%%` refers to the second most recent output; `%%%` refers to the third most recent output and, in general `%%%...%` (*k*-times) refers to the *k*th most recent output. Second, `Out[n]`, where *n* is a positive integer, refers to the *n*th output.

Several examples are given below which illustrate these ideas. First, functions `f`, `g`, and `h` are defined. Then, these functions are evaluated using several different methods. The commands `f[%]` and `f[Out[30]]` given below yield the same output since both evaluate the function `f` at $x = .077$.

```

UsingOutput

In[25]:=
Clear[f,g,c,h]
f[x_]:=x^2
g[x_]:=N[Sqrt[x],2]
h[x_]:=N[Sin[x]+2Cos[x],2];

In[29]:=
g[2] computes g(2)
Out[29]=
1.4

In[30]:=
h[2] computes h(2)
Out[30]=
0.077

In[31]:=
f[%] computes f(.077)
Out[31]=
0.00592958

In[32]:=
f[Out[30]] computes f(.077)
Out[32]=
0.00592958

```

After clearing all prior definitions of `f`, `g`, `c`, and `h`, define $f(x) = x^2$, $g(x)$ to be the numerical value of \sqrt{x} to two decimal places, and $h(x)$ to be the numerical value of $\text{Sin}(x) + 2\text{Cos}(x)$ to two decimal places.

`%` refers to the previous output; `%%` refers to the second most recent output; `%%%` refers to the third most recent output; and, in general, `%%%...%` (*k*-times) refers to the *k*th most recent output.

These methods of retrieving output are useful as input is altered. For example, a new variable **c** is defined below in terms of **a** and **b**. The function **f** can then be evaluated at **c** in several ways which are demonstrated below. **g[*%*]** computes **g** at the previous output, **Out[[35]]**. Hence, **g[*%%*]** computes **g** at the fourth previous output, **Out[[32]]**. In the last example below, **h** is evaluated at the second previous output, **Out[[35]]**.

```

In[33]:=
  c=a+b
Out[33]=
  a + b
In[34]:=
  f[%]
Out[34]=
  (a + b)2
In[35]:=
  f[Out[33]]
Out[35]=
  (a + b)2
In[36]:=
  g[****]
Out[36]=
  0.077
In[37]:=
  h[%]
Out[37]=
  2. Cos[(a + b)2] + Sin[(a + b)2]
  
```

Since we have defined $c = a + b$, the same result would have been obtained if we entered either $f[c]$ or $f[Out[33]]$.

In this case, the same result could have been obtained by entering either $h[(a + b)^2]$ or $h[Out[35]]$.

■ **Composition of Functions**

Mathematica can easily perform the calculation $f[g[x]]$. However, when composing several different functions or repeatedly composing a function with itself, two additional commands are provided:

- 1) **Compose**[*f1*, *f2*, *f3*, . . . , *fn*, *x*] computes the composition $f1 \circ f2 \circ f3 \circ \dots \circ fn(x)$ where *f1*, *f2*, *f3*, ..., and *fn* are functions and *x* is an expression.
- o In Version 2.0, the function **Compose** is replaced by the function **Composition**. In Version 2.0, **Composition**[*f1*, *f2*, . . . , *fn*] [*x*] computes the composition $f1 \circ f2 \circ f3 \circ \dots \circ fn(x)$ where *f1*, *f2*, *f3*, ..., and *fn* are functions and *x* is an expression.

2) **Nest[f, x, n]** computes the composition

$$f \circ f \circ f \circ \dots \circ f(x)$$

(f composed with itself n times)

where f is a function, n is a positive integer, and x is an expression.

□ **Example:**

In the following example $f(x) = x^2$ and $h(x) = x + \sin(x)$.

DefiningFunctions

In[6]:= f[h[f[x]]]
Out[6]=
 $(x^2 + \sin^2(x))^2$

In[7]:= Compose[f, h, f, x]
Out[7]=
 $(x^2 + \sin^2(x))^2$

In[8]:= f[f[f[f[x]]]]
Out[8]=
16x

In[9]:= Nest[f, x, 4]
Out[9]=
16x

$f[h[f[x]]]$ computes $f(h(f(x)))$.

Compose[f, h, f, x]
also computes $f(h(f(x)))$.

$f[f[f[f[x]]]]$ computes $f(f(f(f(x))))$.
The same result could have been obtained by
evaluating **Compose[f, f, f, f, x]**.

However, for repeated compositions of the same function
the command **Nest** can be used.

Nest[f, x, 4] also computes $f(f(f(f(x))))$.

- o In Version 2.0 *Mathematica* displays output for EACH command as it is generated unless a semi-colon is included at the end of the command. Hence, in the following example, output is displayed for all except the last command:

o Example:

Let $f(x) = \text{Log}\left(\frac{2x+1}{x-1/2}\right)$, $g(x) = \text{Sin}(3x) - \text{Cos}(4x)$, $h(x) = x^2$ and $k(x) = h(g(x))$. Compute

and simplify k , compute $\text{Exp}[f(x)] = e^{f(x)}$, and write $e^{f(x+iy)} = \text{Exp}[f(x+iy)]$ in terms of its real and imaginary parts, assuming x and y are real.

```

Version2AlgSimplification
In[4]:=
Clear[f, g, h, k]
f[x_]=Log[(2x+1)/(x-1/2)]
g[x_]=Sin[3 x]-Cos[4 x]
h[x_]=x^2;

Out[2]=
Log[ $\frac{1 + 2 x}{-(\frac{1}{2}) + x}$ ]

Out[3]=
-Cos[4 x] + Sin[3 x]

```

In Version 2.0, output for each command is displayed unless a semi-colon is placed at the end of the command. Hence, in this case the definitions of f and g are shown, the definition of h is suppressed.

- o In Version 2.0, the command **Compose** has been replaced by the command **Composition**. Even though entering the command **Compose**[f, g, x] yields $f[g[x]]$, *Mathematica* issues a warning that **Compose** is an obsolete function, replaced by **Composition**.

- o Also notice that the option **Trig->True** has been added to the command **Expand**. The effect of the option **Trig->True** is to eliminate powers of Sines and Cosines in trigonometric expressions:

```

In[19]:=
  k[x_]=Compose[h,g,x]
  Compose::obsfn:
    Compose is an obsolete function,
    superseded by Composition.

Out[19]=
  (-Cos[4 x] + Sin[3 x])2

In[20]:=
  Expand[k[x]]

Out[20]=
  Cos[4 x]2 - 2 Cos[4 x] Sin[3 x] +
  Sin[3 x]2

In[21]:=
  Expand[k[x],Trig->True]

Out[21]=
  1 -  $\frac{\text{Cos}[6 x]}{2}$  +  $\frac{\text{Cos}[8 x]}{2}$  + Sin[x] -
  Sin[7 x]

```

Even though the command **Compose** is considered obsolete in Version 2.0, $h(g(x))$ is computed correctly.

Expands the terms of $k(x)$.

Expands the terms of $k(x)$ and eliminates powers of trigonometric functions.

- Version 2.0 also includes the new command **ComplexExpand**. If **expression** is a *Mathematica* expression in terms of $x+I y$, the command **ComplexExpand[expression]** rewrites **expression** in terms of its real and imaginary components, assuming that x and y are both real.

In order to compute $h(g(x))$ in Version 2.0, enter **Composition[h,g][x]**:

```

In[5]:=
  Composition[h,g][x]
Out[5]=
  (-Cos[4 x] + Sin[3 x])2
In[6]:=
  fraction=Composition[Exp,f][x]
Out[6]=
   $\frac{1 + 2 x}{-(\frac{1}{2}) + x}$ 
In[8]:=
  fraction /. x->x+I y // Simplify
Out[8]=
   $\frac{1 + 2 x + 2 I y}{-(\frac{1}{2}) + x + I y}$ 
In[9]:=
  ComplexExpand[
    fraction /. x->x+I y]
Out[9]=
  I (  $\frac{2 (-\frac{1}{2} + x) y}{\text{Abs}[-(\frac{1}{2}) + x + I y]^2}$  -
     $\frac{(1 + 2 x) y}{\text{Abs}[-(\frac{1}{2}) + x + I y]^2}$  ) +
     $\frac{(-\frac{1}{2} + x) (1 + 2 x)}{\text{Abs}[-(\frac{1}{2}) + x + I y]^2}$  +
     $\frac{2 y^2}{\text{Abs}[-(\frac{1}{2}) + x + I y]^2}$ 
  
```

Performs the same computation as **Compose[h,g,x]** in Version 1.2.

Computes $e^{f(x)}$ and names the result **fraction**.

replaces each x in **fraction** by $x + I y$ and simplifies the result.

Assuming x and y are real, **ComplexExpand** is used to rewrite $\frac{1+2x+2Iy}{-1/2+x+Iy}$ (above) in terms of its real and imaginary components.

100% ▾

■ 2.3 Mod Math

The command `Mod[a, b]` reduces the number `a` modulo `b`. If `p` is a polynomial, the command `Mod[p, b]` reduces the coefficients of `p` modulo `b`.

□ Example:

In the following example, the factors of $x^4 + x^3 + x^2 + x + 1$ modulo 5 are found and verified. A function `modexpand[poly, p]` which expands and factors the polynomial `poly` modulo `p` is then defined for later use.

Version 1.2 ModMath		
<code>In[27]:=</code>	<code>Factor[x^4+x^3+x^2+x+1, Modulus->5]</code>	<i>factors $x^4 + x^3 + x^2 + x + 1$ modulo 5.</i>
<code>Out[27]=</code>	$(4 + x)^4$	
<code>In[28]:=</code>	<code>mult=Expand[(4+x)^4]</code>	<i>expands $(x+4)^5$ and names the result <code>mult</code>.</i>
<code>Out[28]=</code>	$256 + 256x + 96x^2 + 16x^3 + x^4$	
<code>In[29]:=</code>	<code>Mod[mult, 5]</code>	<i><code>Mod[mult, 5]</code> reduces each coefficient of <code>mult</code> modulo 5.</i>
<code>Out[29]=</code>	$1 + x + x^2 + x^3 + x^4$	
<code>In[30]:=</code>	<code>modexpand[p_, m_] := Block[{poly}, poly=Expand[p]; Mod[poly, m]]</code>	<i><code>modexpand[p, m]</code> first expands the expression <code>p</code> and then reduces each coefficient modulo <code>m</code>. Notice that the variable <code>poly</code> is defined to be a local variable to the function <code>modexpand</code>.</i>

□ **Example:**

It is well known that if F is a field of characteristic p , p a prime number,

and $a, b \in F$, then $(a + b)^{p^m} = a^{p^m} + b^{p^m}$. Illustrate this fact when $m = 1$ for the first five prime numbers.

We proceed by using the user-defined command `modexpand` from above and the built-in command `Prime`. `Prime[i]` returns the i th prime number.

The command `Table` is discussed in more detail in Chapters 4 and 5.

```

In[31]:=
Table[
  modexpand[(x+a)^Prime[i],Prime[i]],
  {i,1,5}] // TableForm

Out[31]//TableForm=
  2      2
a  +  x

  3      3
a  +  x

  5      5
a  +  x

  7      7
a  +  x

 11     11
a  +  x

```

*expands $(x+a)^p$
and reduces modulo p
for the first five prime
numbers.*

- In Version 2.0, **Mod**[*a*, *b*] reduces the number *a* modulo *b*. Notice that unlike prior versions of *Mathematica*, *a* must be a number. To reduce the coefficients of a polynomial *p* modulo *b*, use the command **PolynomialMod**[*p*, *b*]:

○ Example:

<pre>In[78]:= Factor[x^10+x^5+2, Modulus->5]</pre>	<pre>]</pre>	<p>factors $x^{10} + x^5 + 2$ modulo 5.</p>
<pre>Out[78]= 2 5 (2 + x + x)</pre>	<pre>]</pre>	
<pre>In[77]:= poly=Expand[(x^2+x+1)^5]</pre>	<pre>]</pre>	<p>expands $(x^2 + x + 1)^5$ and names the result poly.</p>
<pre>Out[77]= 2 3 1 + 5 x + 15 x + 30 x + 4 5 6 45 x + 51 x + 45 x + 7 8 9 10 30 x + 15 x + 5 x + x</pre>	<pre>]</pre>	
<pre>In[80]:= Mod[13, 5]</pre>	<pre>]</pre>	<p>Mod[13, 5] computes 13 modulo 5.</p>
<pre>Out[80]= 3</pre>	<pre>]</pre>	
<pre>In[81]:= Mod[poly, 5]</pre>	<pre>]</pre>	<p>Unlike Version 1.2, Mod[poly, 5] does not reduce the polynomial poly modulo 5.</p>
<pre>Out[81]= Mod[1 + 5 x + 15 x + 30 x + 4 5 6 45 x + 51 x + 45 x + 7 8 9 10 30 x + 15 x + 5 x + x , 5]</pre>	<pre>]</pre>	

However, coefficients of a polynomial p can be reduced modulo n with the command `PolynomialMod[p, n]`:

```
In[82]:=
  PolynomialMod[poly, 5]

Out[82]=
      5    10
  1 + x  + x
```

PolynomialMod[poly, 5]
reduces poly modulo 5.

Hence, the previous definition of `modexpand` must be altered to include `PolynomialMod` In Version 2.0. This command is then illustrated by creating a table similar to that in the previous example for the prime numbers 13, 17, 19, 23, and 29. Note that this table also includes the prime number as well as the reduced polynomial.

```
In[87]:=
  modexpand[p_, m_] :=
    PolynomialMod[Expand[p], m]

In[89]:=
  modexpand[(x^2+x+1)^11, 11]

Out[89]=
      11    22
  1 + x  + x

In[90]:=
  Table[
    {Prime[i],
     modexpand[(x^2+x+b)^Prime[i], Prime[i]]},
    {i, 6, 10}] // TableForm

Out[90]:=TableForm=
      13    13    26
  13  b  + x  + x

      17    17    34
  17  b  + x  + x

      19    19    38
  19  b  + x  + x

      23    23    46
  23  b  + x  + x

      29    29    58
  29  b  + x  + x
```

100% ▾ ↩

■ 2.4 Graphing Functions and Expressions

One of the best features of *Mathematica* is its graphics capabilities. In this section, we discuss methods of graphing functions and several of the options available to help graph functions. The command used to plot real-valued functions of a single variable is `Plot`. The form of the command to graph the function $f[x]$ on the domain $[a,b]$ is `Plot[f[x], {x, a, b}]`. To plot the graph of $f[x]$ in various shades of gray or colors, the command is `Plot[f[x], {x, a, b}, PlotStyle->GrayLevel[w]]` where w is a number between 0 and 1. `PlotStyle->GrayLevel[0]` represents black; `PlotStyle->GrayLevel[1]` represents a white graph.

If a color monitor is being used, the command is

`Plot[f[x], {x, a, b}, PlotStyle->RGBColor[r, g, b]]` where r , g , and b are numbers between 0 and 1. `RGBColor[1, 0, 0]` represents red, `RGBColor[0, 1, 0]` represents green, and `RGBColor[0, 0, 1]` represents blue.

□ Example:

Use *Mathematica* to define and graph $f(x)=\sin(x)$ on the interval $[-2\pi, 2\pi]$ and $g(x)=e^{-x^2}$ on the interval $[-1,1]$.

GraphingFunctions

```

In[8]:=
Clear[f,g]
f[x_]:=Sin[x]

In[9]:=

Out[9]=
-Graphics-

In[10]:=
g[x_]=Exp[-x^2]
Plot[g[x],{x,-1,1}]

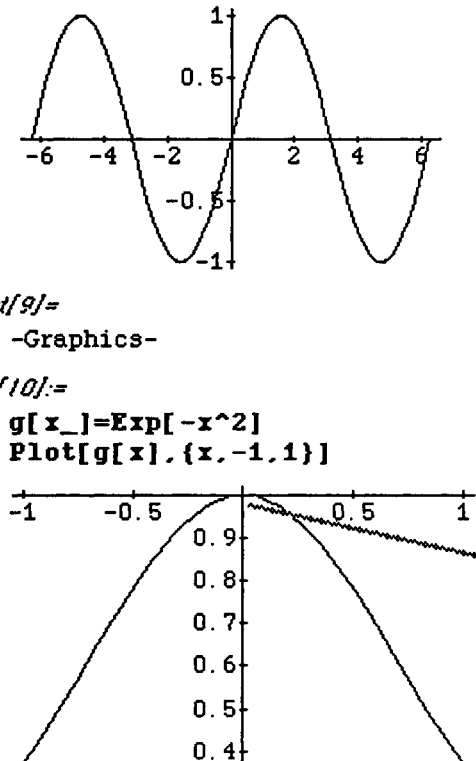
Out[10]=
-Graphics-
                    
```

Notice that whenever we define functions, we first clear any existing prior definitions of them to avoid any possible chance of ambiguity later.

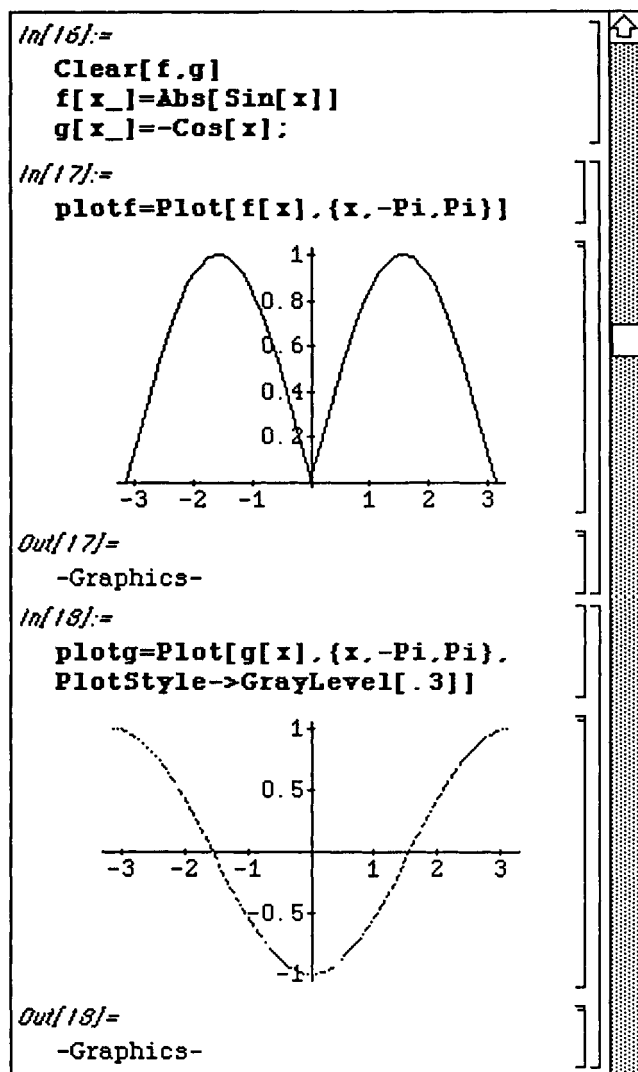
Plot[f[x],{x,-2Pi,2Pi}] graphs the function $f(x) = \sin(x)$ on the interval $[-2\pi, 2\pi]$.

In this case, $g(x) = e^{-x^2}$ is both defined and graphed in a single command.

Notice that Mathematica has placed the axes so that the intersection point of the two axes is the point (0,1).



Graphs of functions, like expressions, can be named. This is particularly useful when one needs to refer to the graph of particular functions repeatedly or to display several graphs on the same axes.



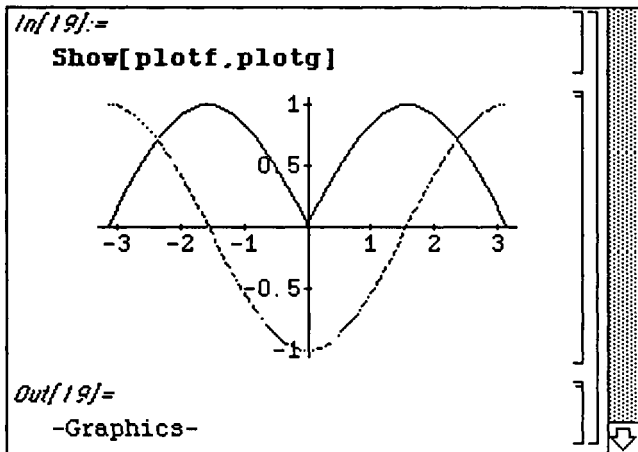
In this example, define
 $f(x)=|\sin(x)|$ and $g(x)=-\cos(x)$.

`plotf` is a graph of $f(x)$ on the
interval $[-\pi, \pi]$.

`plotg` is a graph of $g(x)$ on the
interval $[-\pi, \pi]$.

The option
`PlotStyle->GrayLevel[.3]`
specifies that the color of the graph of
 $g(x)$ be a shade of gray.

The command used to display several graphs on the same axes is **Show**. To show two graphs named **graph1** and **graph2**, the command entered is **Show[graph1, graph2]**. This command is shown below using **plotf** and **plotg** from above:



The command **Show[plotf, plotg]** shows **plotf** and **plotg** simultaneously.

More generally, the commands **Plot** and **Show** have many options. To implement the various options, the form of the command **Plot** is **Plot[f[x], {x, a, b}, options]**; the form of the command **Show** is **Show[graphs, options]**. The option **DisplayFunction->Identity** prevents the graph from being shown; the option **DisplayFunction->\$DisplayFunction** causes the display of a graph which previously was suppressed. For example, one can create several graphs without displaying any of them, and then display all of them simultaneously:

□ Example:

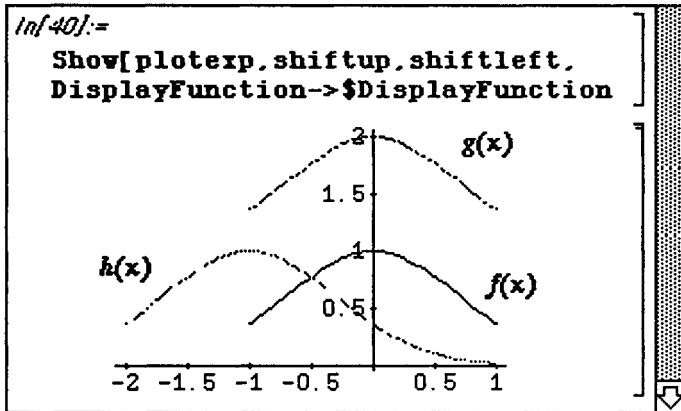
Let $f(x) = e^{-x^2}$, $g(x) = e^{-x^2} + 1 = f(x) + 1$, and $h(x) = e^{-(x-1)^2} = f(x-1)$.

Graph f , g , and h on the intervals $[-1,1]$, $[-1,1]$, and $[-2,1]$, respectively. Show the graphs of all three functions simultaneously.

<pre>In[36]:= Clear[f,g,h] f[x_]=Exp[-x^2] g[x_]=f[x]+1 h[x_]=f[x+1]; In[37]:= plotexp=Plot[f[x],{x,-1,1}, DisplayFunction->Identity] Out[37]= -Graphics- In[38]:= Show[plotexp, DisplayFunction->\$DisplayFunction]</pre>	<p>We begin by clearing any prior definitions of f, g, and h and then define</p> $f(x) = e^{-x^2}, \quad g(x) = f(x) + 1 = e^{-x^2} + 1, \quad \text{and}$ $h(x) = f(x+1) = e^{-(x+1)^2}.$ <p><code>plotexp</code> is a graph of $f(x)$ on the interval $[-1,1]$. The option <code>DisplayFunction->Identity</code> causes no display of <code>plotexp</code>.</p> <p>However, <code>plotexp</code> can be viewed by entering this command.</p>
	<p>In general, the graphics option <code>DisplayFunction->Identity</code> allows one to create graphics but not view them until necessary.</p>
<pre>In[39]:= shiftup=Plot[g[x],{x,-1,1}, PlotStyle->GrayLevel[.2], DisplayFunction->Identity] shiftright=Plot[h[x],{x,-2,1}, PlotStyle->GrayLevel[.4], DisplayFunction->Identity]</pre>	<p><code>shiftright</code> is a graph of $g(x)$; the option <code>DisplayFunction->Identity</code> prevents the graph from being shown.</p> <p><code>shiftright</code> is a graph of $h(x)$; the option <code>DisplayFunction->Identity</code> prevents the graph from being shown.</p>

Even though `shiftright` and `shiftright` are not shown, they may be viewed along with `plotexp`, using the `Show` command together with the option `DisplayFunction->$DisplayFunction`.

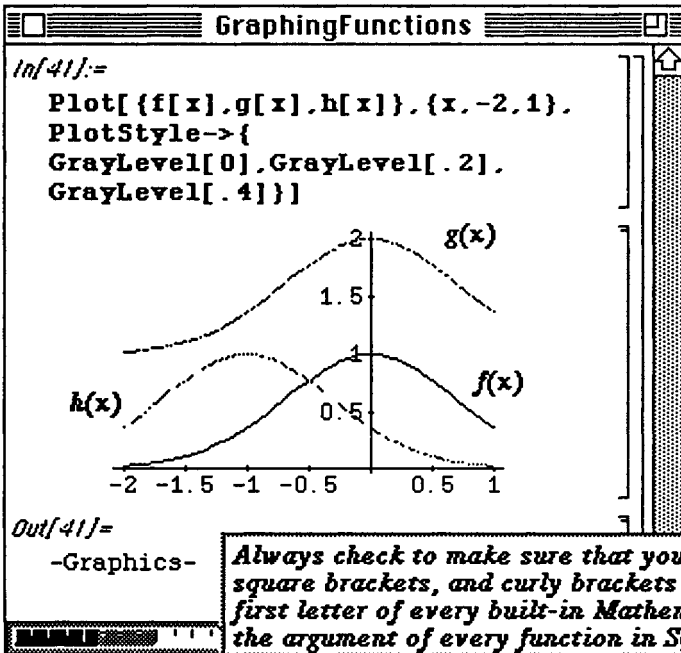
Note that no graphs would be displayed if the `DisplayFunction->$DisplayFunction` option were omitted from the following `Show` command:



However this command shows the graphs of $f(x)$ (in `graylevel[0]`), $g(x)$ (in `graylevel[.2]`), and $h(x)$ (in `graylevel[.4]`).

Note: The labels $f(x)$, $g(x)$, and $h(x)$ were added later.

The `Plot` command can also be used to `Plot` several functions simultaneously. To display the graphs of the functions $f[x]$, $g[x]$, and $h[x]$ on the domain $[a,b]$ on the same axes, enter commands of the form `Plot[{f[x],g[x],h[x]},{x,a,b},options]`. This command can be generalized to include more than three functions.



Here we plot $f(x)$, $g(x)$, and $h(x)$ on the interval $[-2,1]$.

Notice that the color of the graph of $f(x)$ is `GrayLevel[0]`; the color of the graph of $g(x)$ is `GrayLevel[.2]`; and the color of the graph of $h(x)$ is `GrayLevel[.4]`.

Always check to make sure that you have nested parentheses, square brackets, and curly brackets correctly; capitalized the first letter of every built-in Mathematica function; and included the argument of every function in SQUARE BRACKETS.

■ Other Available Options

Additional **Plot** options include:

1) **AspectRatio**->**number**

This makes the ratio of the length of the x-axis to the y-axis **number**. The default value is **1/GoldenRatio**. **GoldenRatio** is a built-in *Mathematica* constant (like **E** and **Pi**)

with value $\frac{1+\sqrt{5}}{2}$ (approximately 1.61803).

2) **Framed**->**True**

This draws a frame around the graph; the default value is **False**--no frame is drawn.

- In Version 2.0, the option **Framed** is replaced by the option **Frame**. Hence, if you are using Version 2.0, including **Frame**->**True** instructs *Mathematica* to place a frame around the graph.

3) **Ticks**->**None** or **Ticks**->{{**x-axis ticks**},{**y-axis ticks**}}

This specifies that either no tick marks be placed on either axis OR tick marks be placed on the x-axis at **x-axis ticks** and on the y-axis at **y-axis ticks**.

4) **AxesLabel**->{"**x-axis label**","**y-axis label**"}

This labels the x-axis **x-axis label** and the y-axis **y-axis label**. For example, the command **Plot[f[x], {x, xmin, xmax, AxesLabel->{"jane", "mary"}}]** graphs the function **f[x]** on the interval [**xmin**, **xmax**]; and labels the x-axis **jane** and the y-axis **mary**. The default for the option is that no labels are shown.

5) **PlotLabel**->{"**name**"}

This centers **name** above the graph. The default for the option is that the graph is not labeled.

6) **Axes**->{**x-coordinate**,**y-coordinate**}

This option specifies that the x-axis and y-axis intersect at the point {**xcoordinate**,**ycoordinate**}.

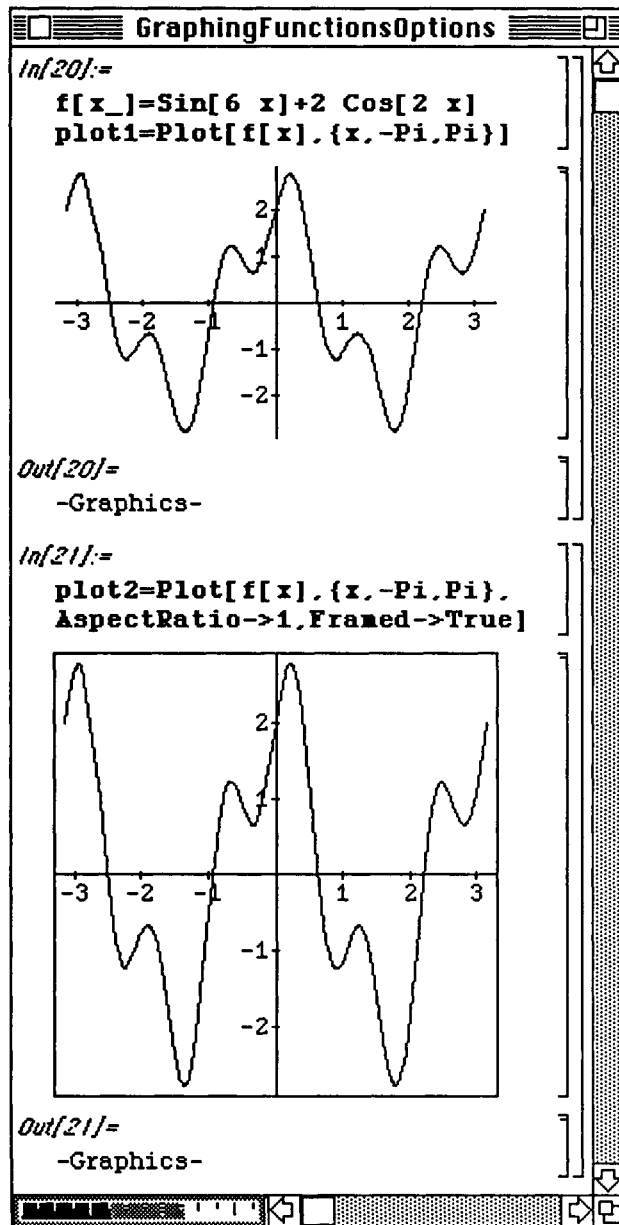
- In Version 2.0, **Axes** has been redefined. The option **Axes**->**False** specifies that the graph is to be drawn without axes; the option **AxesOrigin**->{**x-coordinate**,**y-coordinate**} places the axes so they intersect at the point {**x-coordinate**,**y-coordinate**}.

7) **PlotRange**->{**y-minimum**,**y-maximum**}

specifies the range displayed on the final graph to be the interval [**y-minimum**,**y-maximum**]; **PlotRange**->**All** attempts to show the entire graph.

□ Example:

These graphing options are illustrated below:



To illustrate the various features of the `Plot` command, we define $f(x) = \sin(6x) + 2\cos(2x)$ and graph $f(x)$ on the interval $[-\pi, \pi]$.

Notice that the resulting graph is named `plot1`; This will allow us to use the graph later.

`plot2` is also a graph of $f(x)$ on the interval $[-\pi, \pi]$. Adding the option

`AspectRatio->1` specifies that the ratio of the length of the x-axis to the length of the y-axis is 1; the option `Framed->True` encloses the graphics cell in a box (or frame).

In Version 2.0, the option `Framed` has been replaced by the option `Frame`.

plot3 is a graph of $f(x)$. The option

Ticks→**None** specifies that no tick marks are placed on either the x -axis or y -axis; the

option **AxesLabel**→
{"x-axis", "y-axis"}
specifies that the x -axis is marked **x-axis**

and the y -axis is marked **y-axis**.

When working with the Plot command, be sure to begin with a CAPITAL letter and enclose the entire command in square brackets.

For **plot4** the option

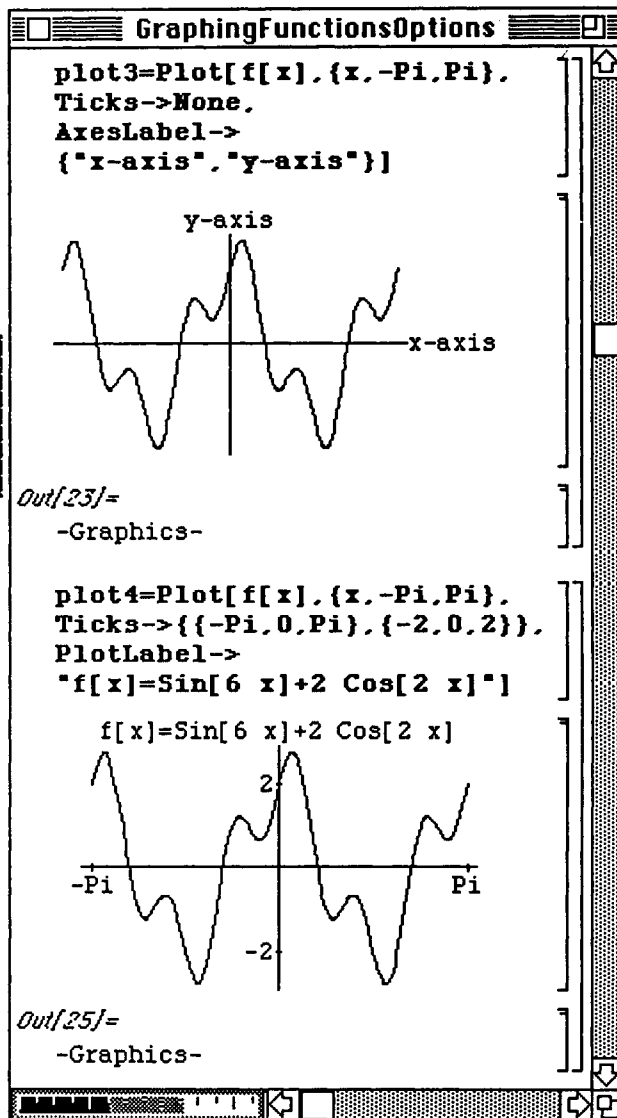
Ticks→{ $\{-\pi, 0, \pi\}$, $\{-2, 0, 2\}$ }

specifies that tick marks be placed at $x=-\pi$ and $x=\pi$ on the x -axis and $y=-2$ and $y=2$ on the y -axis; the option

PlotLabel→
"f[x]=Sin[6 x]+2 Cos[2 x]"

specifies that the top of the graph is marked

$f[x]=\text{Sin}[6 x]+2 \text{Cos}[2 x]$.



● Graphing Features and Options of Version 2.0

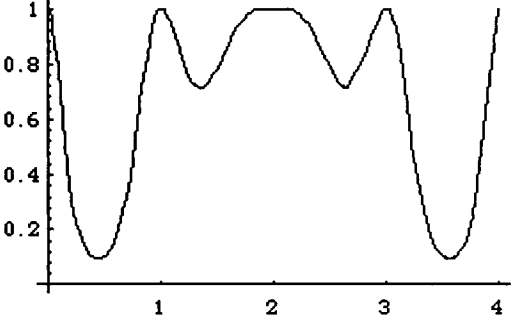
Version 2.0 of *Mathematica* offers several plotting options which are not available or differ from those in Version 1.2. In the first example below, the fact that a semi-colon must follow a command in Version 2.0 in order that it be suppressed is illustrated. (In Version 1.2, only the output of the last command in a single input cell is given even if semi-colons are not used.) After defining the function f , the graph of f is plotted and called `plotf`. Since a semi-colon follows the definition of f , the formula for f is not given in the output. Also shown below is the `GridLines` option in the `Plot` command. Notice in `feature1`, `GridLines->Automatic` causes horizontal and vertical gridlines to be shown on the graph.

GraphingOptions

```

In[3]:=
Clear[f]
f[x_]=Exp[-(x-2)^2 (Sin[Pi x])^2];
plotf=Plot[Exp[-(x-2)^2 (Sin[Pi x])^2],
{x,0,4}]

```



Notice that with Version 2.0, a semi-colon must be placed at the end of each command in order to suppress the resulting output. In this case, after defining

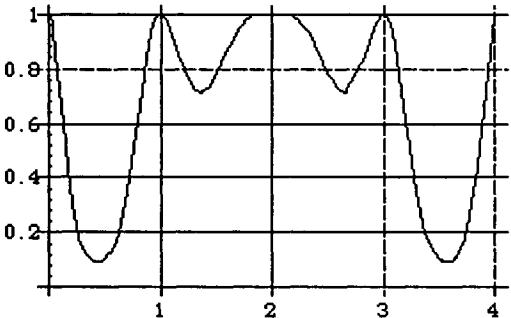
$$f(x) = e^{-(x-2)^2 \sin^2(\pi x)}$$

we graph $f(x)$ on $[0,4]$ and name the resulting graph `plotf`.

```

Out[3]=
-Graphics-
In[4]:=
feature1=Show[plotf,GridLines->Automatic]

```



The option

GridLines->Automatic

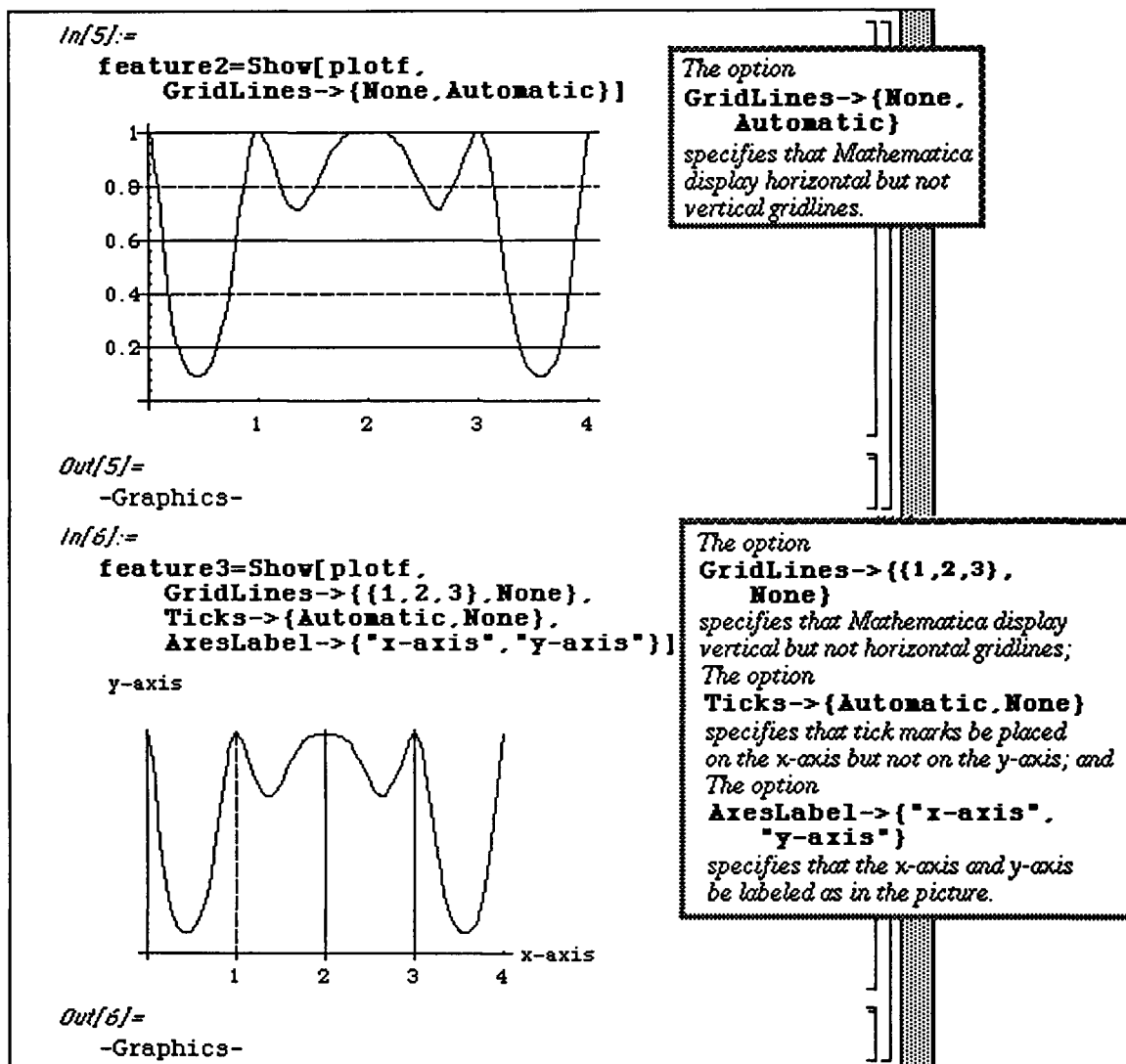
instructs Mathematica to display vertical and horizontal gridlines.

```

Out[4]=
-Graphics-

```

The **GridLines** option can be altered slightly. The following examples illustrate how one type of gridline is requested. In **feature2**, **GridLines**→{None, Automatic} specifies that only horizontal gridlines be displayed while in **feature3**, **GridLines**→{{1, 2, 3}, None} gives vertical gridlines at $x = 1, 2,$ and 3 . Also in **feature3**, **Ticks**→{Automatic, None} causes tick marks to be placed on the x -axis but none on the y -axis. Finally in **feature3**, the x and y axes are labeled with the option **AxesLabel**→{"x-axis", "y-axis"}.



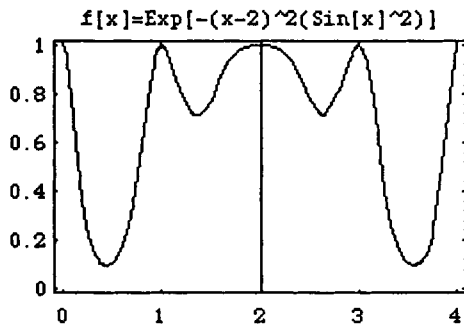
Several other **Plot** options are shown in the examples below. In **feature4**, **AxesOrigin**→{ x_0, y_0 } is illustrated. This causes the major axes to be drawn in such a way that they meet at the point { x_0, y_0 }. Another option is **Frame** which is demonstrated in both examples. **Frame**→True encloses the graph in a frame.

o In Version 2.0, **AxesOrigin** replaces **Axes** from Version 1.2 and **Frame** replaces **Framed**.

Note the tick marks which accompany the frame in **feature4**. In **feature5**, however, the **FrameTicks->None** option prohibits the marking of ticks on the frame. Also notice the **PlotLabel** option which appears in each **Plot** command. In **feature4**, the label is given in quotation marks. This causes the function within the quotations to be printed exactly as it appears in the **PlotLabel** option. Since the label does not appear in quotations in **feature5**, the label is given in mathematical notation.

```
In[7]:=
```

```
feature4=Show[plotf,Ticks->None,
  AxesOrigin->{2,0},
  PlotLabel->"f[x]=Exp[-(x-2)^2(Sin[x]^2)]",
  Frame->True]
```

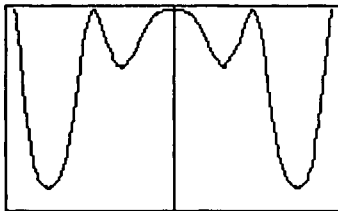


```
Out[7]=
```

```
-Graphics-
```

```
In[8]:=
```

```
feature5=Show[plotf,Ticks->None,
  AxesOrigin->{2,0},
  PlotLabel->Exp[-(x-2)^2(Sin[x]^2)],
  Frame->True,FrameTicks->None,
  Ticks->{None, Automatic}]
```

$$\text{Exp}\left[-\left(-2+x\right)^2\left(\text{Sin}\left[x\right]\right)^2\right]$$


```
Out[8]=
```

```
-Graphics-
```

The option

AxesOrigin->
{2,0}

instructs Mathematica to display the axes so that they intersect at the point (2,0);

The option

Frame->True instructs Mathematica to place a frame around the resulting graph;

The option

PlotLabel is used to label the graph.

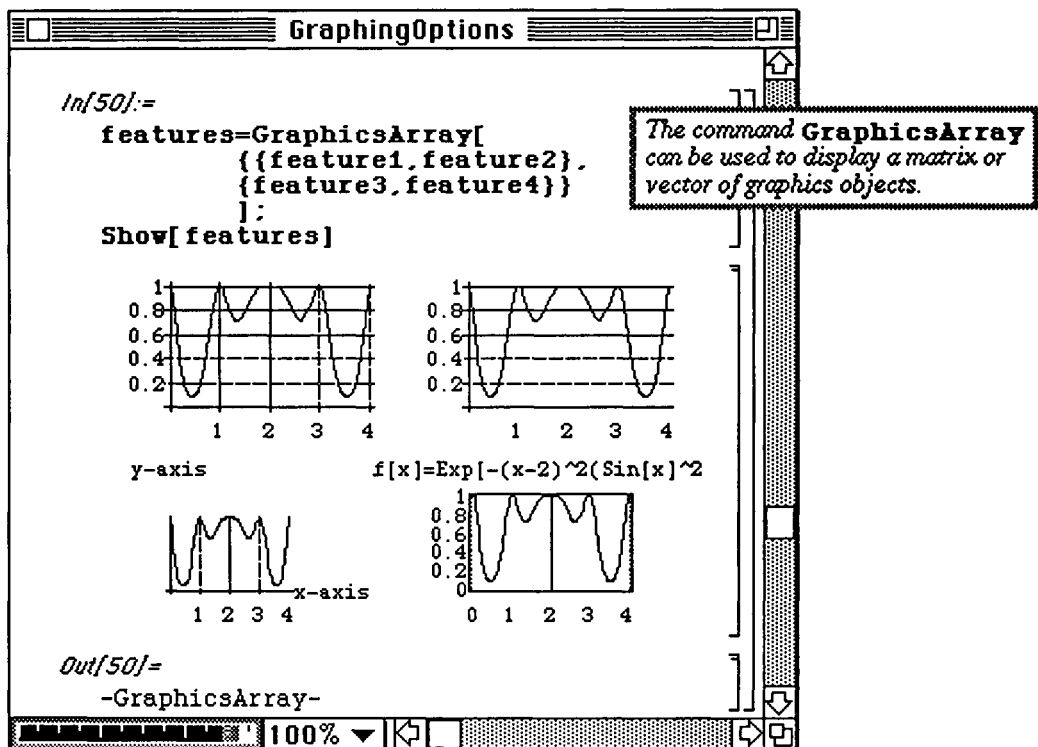
In this case, since

Exp[-(x-2)^2(Sin[x]^2)] is not contained in quotation marks (as in the previous example), the result is displayed in traditional mathematical notation.

● Displaying Several Graphs with Version 2.0

The plots given in `feature1`, `feature2`, `feature3`, and `feature4` are viewed below in a single graphics cell with the `GraphicsArray` option.

`GraphicsArray[{feature1, feature2}, {feature3, feature4}]` produces an array of graphics objects called `features` which is viewed in pairs with `Show[features]`, where `features=GraphicsArray[{feature1, feature2}, {feature3, feature4}]`. In general, `GraphicsArray` can be used to visualize any $m \times n$ array of graphics objects.



The command `Rectangle[{x0,y0},{x1,y1}]` creates the graphics primitive for a filled rectangle with sides along the lines $x = x_0$, $y = y_0$, $x = x_1$, and $y = y_1$. Hence, other *Mathematica* commands must be used to visualize the rectangles represented by `Rectangle[{x0,y0},{x1,y1}]`. Visualization is accomplished with `Show` and `Graphics` as illustrated below.

GraphingOptions

```
In[72]:=
Show[Graphics[ {
  Rectangle[ {0,0} . {1,1} ],
  Rectangle[ {1,1} . {1.5,1.5} ],
  Rectangle[ {0,1,1} . {.4,1.5} ],
  Rectangle[ {.5,1.1} . {.9,1.5} ],
  Rectangle[ {1.1,0} . {1.4,.5} ],
  Rectangle[ {1.1,.55} . {1.4,.9} ]
}]]
```

*Notice how the command **Rectangle** can be used to create and display several rectangles.*

corresponds to
Rectangle[{1,1} . {1.5,1.5}]

corresponds to
Rectangle[{1.1,.55} . {1.4,.5}]

corresponds to
Rectangle[{1.1,0} . {1.4,.5}]

Out[72]=
 -Graphics-

100%

Rectangle[{x0,y0},{x1,y1}] can be used in conjunction with other graphics cells to produce graphics of a particular size. The command **Show**[Rectangle[{x0,y0},{x1,y1},plot]] displays **plot** within the rectangle determined with **Rectangle**[{x0,y0},{x1,y1}]. This is illustrated below with rectangles from the previous example as well as earlier plots.

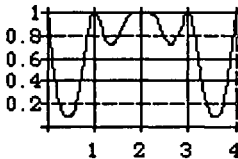
GraphicsOptions

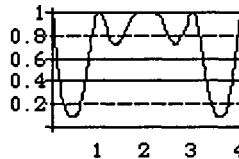
Graphics cells can be shown in rectangles of varying dimensions and then combined into a single graphics cell and displayed.


```

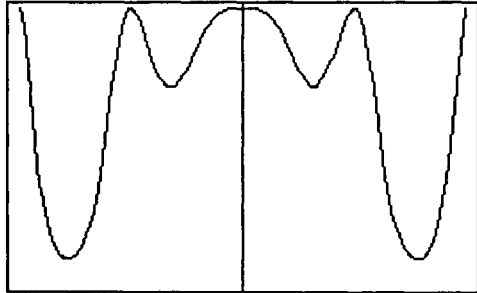
In[74]:=
Show[Graphics[ {
  Rectangle[ {0,0}, {1,1}, feature5],
  Rectangle[ {1,1}, {1.5,1.5}, plotf],
  Rectangle[ {0,1.1}, {4,1.5}, feature1],
  Rectangle[ {0.5,1.1}, {0.9,1.5}, feature2],
  Rectangle[ {1.1,0}, {1.4,5}, feature3],
  Rectangle[ {1.1,.55}, {1.4,.9}, feature4]
}]]


```








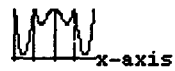
$$E^{-(((-2 + x)^2 \sin[x]^2)}$$


$$f[x]=Exp[-(x-2)^2(Sin[x]^2)]$$


y-axis



x-axis



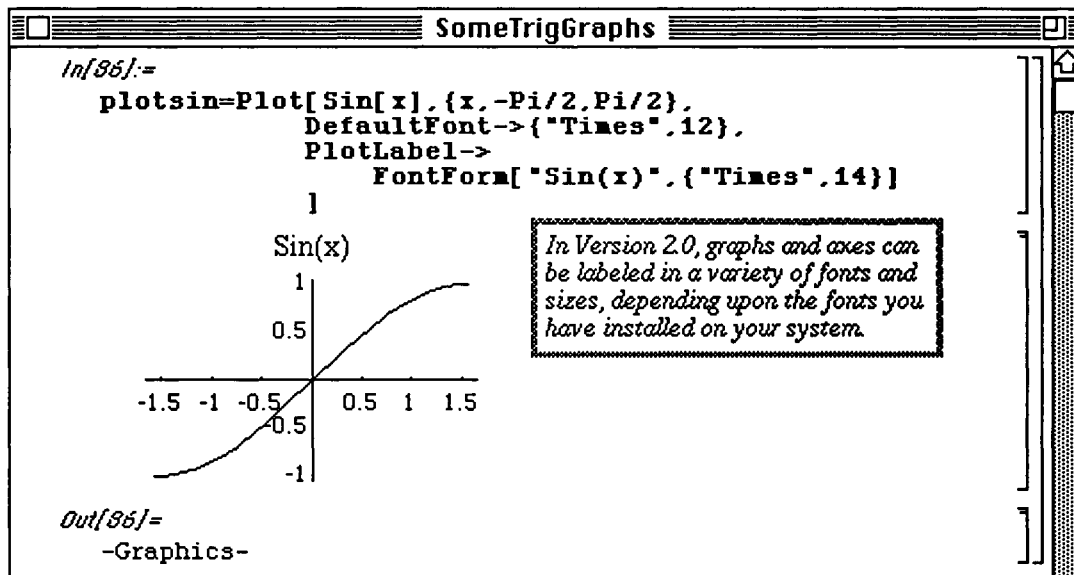
Out[74]=
-Graphics-

100%
⏪
⏩

○ Labeling Graphs in Version 2.0

In addition to the above features, graphs created with Version 2.0 can be labeled in a variety of ways. For example, in the following example the options `DefaultFont->{"font", size}` and `PlotLabel->FontForm["label", {"font", size}]`, where `font` and `size` is a font available on your computer and `label` is the desired graphics label, are used to create several trigonometric graphs.

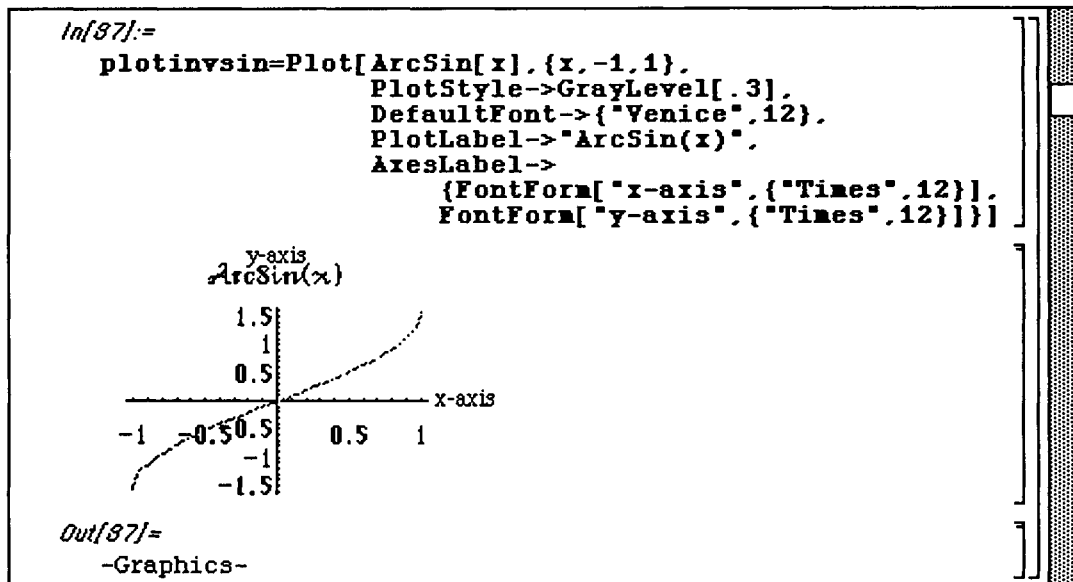
The numbering of the tick marks of `plotsin` are in size 12 Times font; the graph is labeled in size 14 Times font:



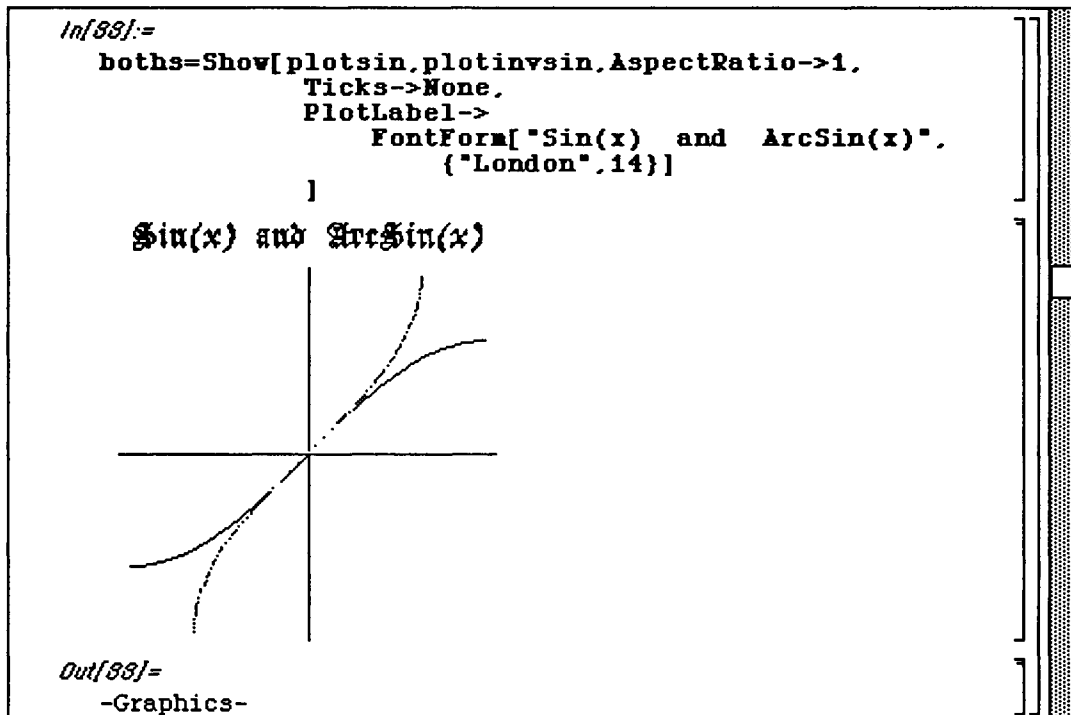
Similarly, the axes can also be labeled in different fonts and sizes using the option

```
AxesLabel->{
  FontForm["x-axis label", {"font", size}],
  FontForm["y-axis label", {"font", size}]
}
```

In the following example, the function $\text{ArcSin}(x) = \text{Sin}^{-1}(x)$ is graphed on the interval $[-1,1]$. The axes are labeled "x-axis" and "y-axis" in size 12 Times font. The graph is labeled "ArcSin (x)" in size 12 Venice font since the **DefaultFont** is chosen to be size 12 Venice font:



Naturally, many different options can be combined together. In the following window, the previous two graphs are displayed. The option `Ticks->None` specifies that no tick marks are to be drawn on either axis; the graph is labeled "Sin(x) and ArcSin(x)" in size 14 London font:



In the next example we graph $\text{Cos}(x)$ and $\text{ArcCos}(x)$. The option `DisplayFunction->Identity` is used so the graphs are not immediately displayed. Instead, these three graphs are shown simultaneously with the three previous graphs, `plotsin`, `plotinvsin`, and `boths`, as a graphics array:

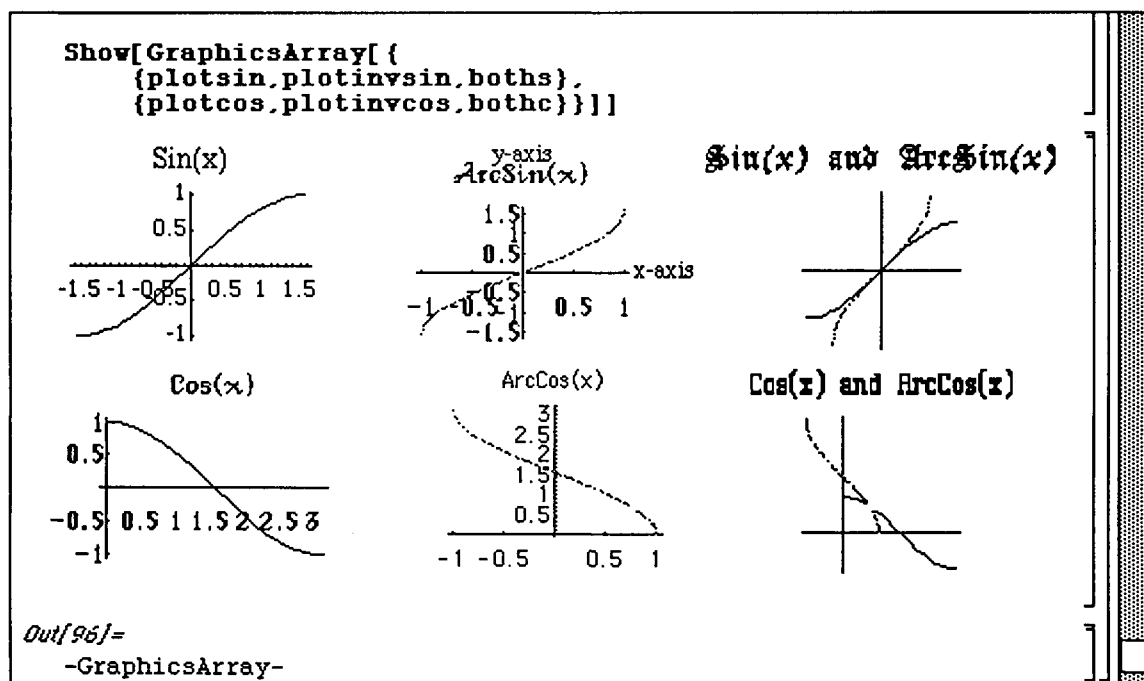
```
In[96]:=
plotcos=Plot[Cos[x],{x,0,Pi},
             DefaultFont->{"Yenice",12},
             PlotLabel->"Cos(x)",
             DisplayFunction->Identity];

plotinvcos=Plot[ArcCos[x],{x,-1,1},
               DefaultFont->{"Einstein",10},
               PlotLabel->"ArcCos(x)",
               PlotStyle->GrayLevel[.3],
               DisplayFunction->Identity];

bothc=Show[plotcos,plotinvcos,AspectRatio->1,
           Ticks->None,
           PlotLabel->FontForm["Cos(x) and ArcCos(x)",
                               {"Athens",12}]]];

Show[GraphicsArray[{
  {plotsin,plotinvsin,boths},
  {plotcos,plotinvcos,bothc}}]]]
```

All six graphs are then displayed as a graphics array, illustrating the various options we have used:



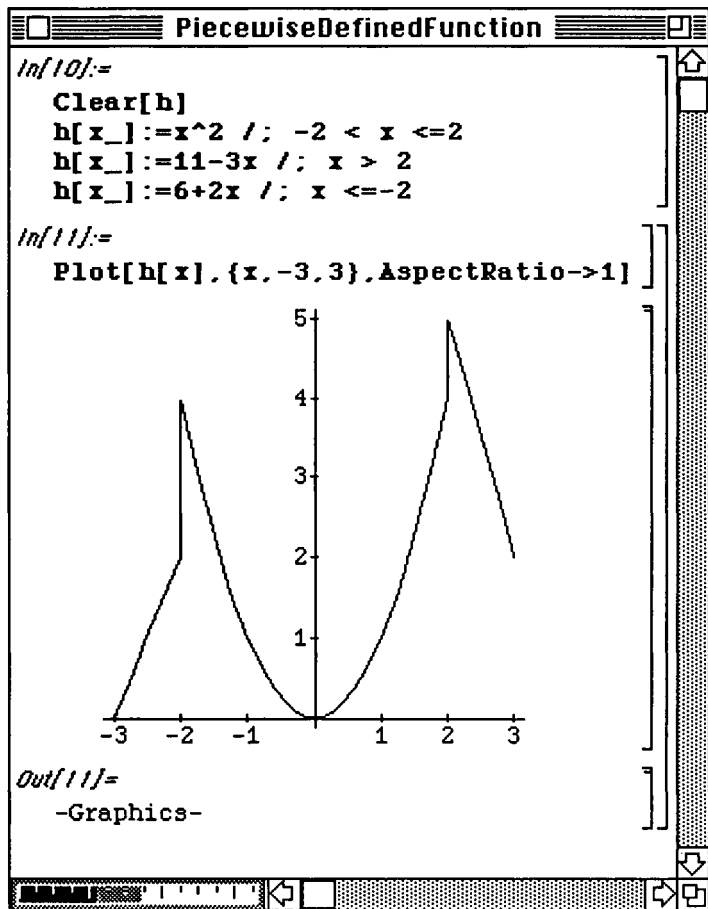
■ Piecewise Defined Functions

Piecewise defined functions may also be defined and graphed with *Mathematica*. In the following example, $h(x)$ is defined in three "pieces". Notice that $/;$ designates the definition of $h(x)$ for different domain values.

□ Example:

Use *Mathematica* to graph $h(x)$ on the interval $[-3,3]$ if $h(x) = \begin{cases} 6+2x & \text{for } x \leq -2 \\ x^2 & \text{for } -2 < x \leq 2 \\ 11-3x & \text{for } x > 2 \end{cases}$

Not that \leq represents a less than or equal to symbol.



After clearing all prior definitions of h , define

$$h(x) = \begin{cases} 6+2x & \text{for } x \leq -2 \\ x^2 & \text{for } -2 < x \leq 2 \\ 11-3x & \text{for } x > 2 \end{cases}$$

Piecewise-defined functions are graphed the same way as other functions.

Functions can be defined recursively. For example, if the function $f[x]$ is defined on the interval $[a,b]$, then f can be defined for $x > b$ with $f[x_]:=f[x-(b-a)]$ /; $x>b$. Two examples are given below. Functions of this type are useful in the study of Fourier series.

○ Example:

(A) If $-1 \leq x \leq 1$, let $f(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1; \\ -1 & \text{for } -1 \leq x < 0. \end{cases}$ If $x > 1$, define $f(x)$ recursively by $f(x) = f(x-2)$. Use *Mathematica* to define f and graph f on the interval $[0,6]$.

(B) If $-1 \leq x \leq 3$, let $h(x) = \begin{cases} \frac{1}{2}x & \text{for } -1 \leq x \leq 2; \\ 1 & \text{for } 2 \leq x < 3 \end{cases}$

$h(x) = h(x-4)$. Use *Mathematica* to define h and graph h on the interval $[0,12]$.

○ Version 2.0 was used in the following solution to illustrate the Version 2.0 graphics option

Background->GrayLevel[n], where n is between 0 and 1. If using Version 2.0, the functions are defined the exact same way; however, the option **Background->GrayLevel[n]** is not available in Version 1.2:

<pre> In[84]:= Clear[f] f[x_]:=1 /; 0<=x<=1 f[x_]:=-1 /; -1<=x<0 f[x_]:=f[x-2] /; x > 1 In[86]:= graphf=Plot[f[x],{x,0,6},DisplayFunction->Identity]; graphftwo=Plot[f[x],{x,0,6}, Background->GrayLevel[.2], PlotStyle->{{Thickness[.01],GrayLevel[1]}}. DisplayFunction->Identity]; In[90]:= Clear[h] h[x_]:=1/2x /; -1<=x<=2 h[x_]:=1 /; 2<=x<=3 h[x_]:=h[x-4] /; 3<x In[92]:= graphh=Plot[{h[x],1/2 h[x-1]},{x,0,12}, PlotStyle->{GrayLevel[0],GrayLevel[.3]}. DisplayFunction->Identity]; graphhtwo=Plot[{h[x],1/2 h[x-1]},{x,0,12}, Background->GrayLevel[0], PlotStyle->{{GrayLevel[1]},{GrayLevel[.8]}}. DisplayFunction->Identity]; </pre>	<p>After clearing all prior definitions of f, f is defined by $f(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq 1 \\ -1 & \text{for } -1 \leq x < 0 \end{cases}$</p> <p>Then $f(x) = f(x-2)$ for $x > 1$ defines f recursively for x greater than one.</p>
	<p>graphf graphftwo are different graphs of f illustrating Plot options available in Version 2.0.</p>

In Version 2.0, arrays of graphics cells can be visualized with the command `GraphicsArray`. Since `graphf`, `graphftwo`, `graphh`, and `graphhtwo` are graphics cells, `{{graphf, graphftwo}, {graphh, graphhtwo}}` is an array of graphics cells. All four can be viewed in a single graphics cell using the command `GraphicsArray`:

```
In[93]:=
Show[GraphicsArray[{{graphf, graphftwo},
{graphh, graphhtwo}}]]
```

```
Out[93]=
-GraphicsArray-
```

Sometimes it is useful to have *Mathematica* remember functional values it computes. For example, this is particularly useful when defining functions recursively as in the previous examples. In general, to define a function f to remember the values it computes enter the definition in the form $f[x_]:=f[x]=\text{mathematical expression}$.

□ Example:

Use *Mathematica* to define $k(x) = \text{Exp}[-(x-2)^2] \text{Cos}(\pi(x-2))$. For $0 \leq x \leq 4$, define $g(x) = k(x)$ and for $x > 4$, define g recursively by $g(x) = g(x-4)$. Graph g on the interval $[0,16]$.

In the following example, notice that k is defined so that *Mathematica* remembers the values of $k(x)$ it computes. Since g is defined recursively in terms of k , evaluation of g for values of x greater than four proceeds quickly since the corresponding k -values have already been computed and remembered:

Mathematica will remember the values of $k(x)$ it computes.

```

In[33]:=
Clear[g,k]
k[x_]:=k[x]=N[Exp[-(x-2)^2 Abs[Cos[Pi (x-2)]]]]
In[39]:=
g[x_]:=k[x] /; 0<=x<=4
g[x_]:=g[x-4] /; 4<x
graphg=Plot[g[x],{x,0,16}]
    
```

*Since $k(x)$ is defined so that *Mathematica* remembers the values of $k(x)$ it computes, evaluation of $g(x)$ will be much faster since g is defined recursively in terms of k .*

Out[39]=
-Graphics-

Time: 68.43 seconds 100%

■ 2.5 Exact and Approximate Solutions of Equations

■ Exact Solutions of Equations

Mathematica can solve many equations exactly. For example, *Mathematica* can find exact solutions to systems of equations and exact solutions to polynomial equations of degree four or less. Since a single equals sign (=) is used to name objects in *Mathematica*, equations in *Mathematica* are of the form

left-hand side==right-hand side. The "double-equals" sign (==) between the left hand side and right hand side specifies that the object is an equation. For example, to represent the equation $3x+7=4$ in *Mathematica*, type **`3x+7==4`**. The command **`Solve[lhs==rhs, x]`** solves the equation $lhs=rhs$ for x .

If the **only** unknown in the equation $lhs=rhs$ is x and *Mathematica* does not need to use inverse functions to solve for x , then the command **`Solve[lhs==rhs]`** solves the equation $lhs=rhs$ for x . Hence, to solve the equation $3x+7=4$, both the command **`Solve[3x+7==4]`** and **`Solve[3x+7==4, x]`** produce the same result.

Notice that the representation of equations in *Mathematica* involves replacing the traditional single equals sign by a double equals sign:

□ Example:

Use *Mathematica* to find exact solutions of the equations $3x+7=4$, $\frac{x^2-1}{x-1}=0$, and $x^3+x^2+x+1=0$.

The screenshot shows the **SolvingEquations** window with the following content:

Example 1:
In[1]:=
Solve[3x+7==4]
Out[1]=
 {{x -> -1}}

*The command **Solve[3x+7==4]** solves the linear equation $3x+7=4$ for x .*

Example 2:
In[2]:=
Solve[(x^2-1)/(x-1)==0]
Out[2]=
 {{x -> -1}}

***Solve[(x^2-1)/(x-1)==0]** solves the rational equation $\frac{x^2-1}{x-1}=0$ for x .*

Example 3:
In[3]:=
Solve[x^3+x^2+x+1==0]
Out[3]=
 {{x -> 0}, {x -> $\frac{-1 + \text{Sqrt}[-3]}{2}$ },
 {x -> $\frac{-1 - \text{Sqrt}[-3]}{2}$ }}

***Solve[x^3+x^2+x+1==0]** solves the cubic equation $x^3+x^2+x+1=0$ exactly. In general, *Mathematica* will find the exact roots of any polynomial equation with degree four or less.*

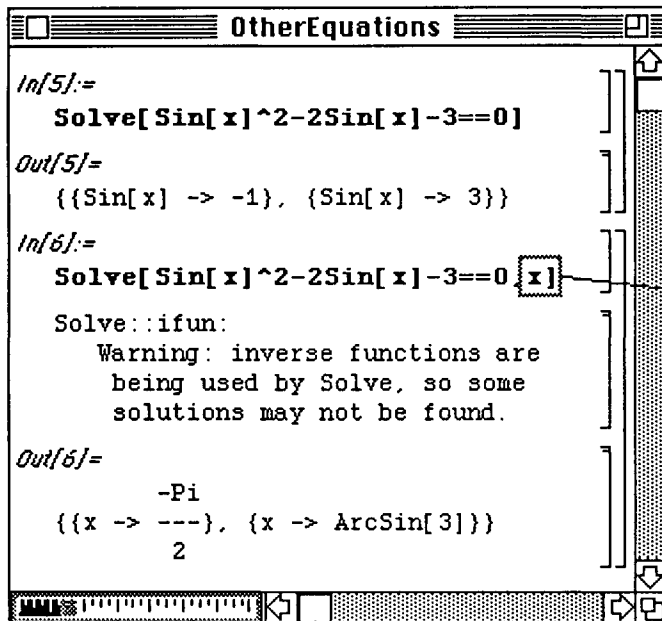
Warning: *Don't forget the "double-equals" when you solve equations!*

As stated above, the exception to the above rule is when using the command **Solve** to find solutions of equations where inverse functions must be used:

□ Example:

Solve $\sin^2(x) - 2\sin(x) - 3 = 0$.

When the command `Solve[Sin[x]^2-2Sin[x]-3==0]` is entered, *Mathematica* solves the equation for `Sin[x]`. However, when the command `Solve[Sin[x]^2-2Sin[x]-3==0,x]` is entered, *Mathematica* attempts to solve the equation for `x`. In this case, *Mathematica* succeeds in finding one solution:



*Even though the only variable in the equation $\sin^2(x) - 2\sin(x) - 3 = 0$ is `x`, *Mathematica* will use inverse functions to solve $\sin^2(x) - 2\sin(x) - 3 = 0$ only if `x` is included in the **Solve** command.*

Solve[{lhs1=rhs1, lhs2==rhs2}, {x, y}] solves a system of two equations for x and y. **Solve**[{lhs1==rhs1, lhs2==rhs2}] attempts to solve the system of equations for all unknowns. In general, **Solve** can find the solutions to a system of linear equations. In fact, if the systems to be solved are inconsistent or dependent, *Mathematica*'s output will tell you so.

□ Example:

Use *Mathematica* to solve each of the systems of equations (i) $\begin{cases} 3x - y = 4 \\ x + y = 2 \end{cases}$; and

(iii) $\begin{cases} 2x - 3y + 4z = 2 \\ 3x - 2y + z = 0 \\ x + y - z = 1 \end{cases}$

<pre style="font-family: monospace;">In[4]:= Solve[{3x-y==4, x+y==2}, {x, y}] Out[4]= {{x -> 3/2, y -> 1/2}}</pre>	<p>Solve[{3x-y==4, x+y==2}, {x, y}] solves the system in two unknowns</p> $\begin{cases} 3x - y = 4 \\ x + y = 2 \end{cases} \text{ for } x \text{ and } y.$
<p><i>An equation is always represented by the form expression1==expression2. You must remember to include the "double-equals" between the left-hand side and the right-hand side of an equation.</i></p>	
<pre style="font-family: monospace;">In[8]:= Solve[{2x-3y+4z==2, 3x-2y+z==0, x+y-z==1}, {x, y, z}] Out[8]= {{x -> 7/10, y -> 9/5, z -> 3/2}}</pre>	<p>Solve[{2x-3y+4z==2, 3x-2y+z==0, x+y-z==1}, {x, y, z}] solves the system of three equations</p> $\begin{cases} 2x - 3y + 4z = 2 \\ 3x - 2y + z = 0 \\ x + y - z = 1 \end{cases} \text{ for } x, y, \text{ and } z.$

Although *Mathematica* can find the exact solution to every polynomial equation of degree four or less, exact solutions to some equations that *Mathematica* can solve may not be meaningful. In those cases, *Mathematica* can provide approximations of the exact solutions using either the **N**[expression] or the **expression // N** command:

Remember that *Mathematica* denotes $\sqrt{-1}$ by **I**.

□ Example:

Approximate the values of x that satisfy the equation (i) $x^4 - 2x^2 = 1 - x$; and (ii) $1 - x^2 = x^3$.

```

SolvingEquations

In[22]:=
N[Solve[x^4-2x^2==1-x]]
Out[22]=
{{x -> 1.34509}, {x -> -1.71064},
 {x -> 0.182777 + 0.633397 I},
 {x -> 0.182777 - 0.633397 I}}

In[23]:=
Solve[1-x^2==x^3,x] // N
Out[23]=
{{x -> 0.754878},
 {x -> -0.877439 + 0.744862 I},
 {x -> -0.877439 - 0.744862 I}}

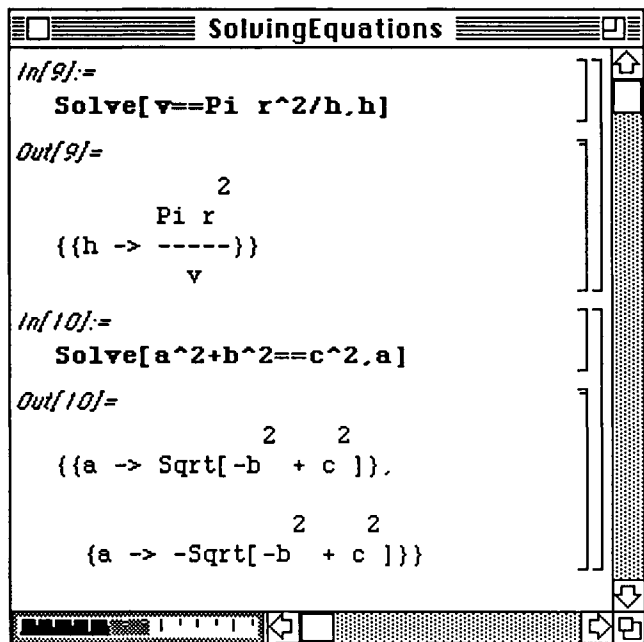
```

N[Solve[x⁴-2x²==1-x]] solves the equation $x^4 - 2x^2 = 1 - x$ and then provides approximations of the roots

Solve[1-x²==x³,x] // N solves the equation $1 - x^2 = x^3$ and then provides approximations of the roots

*In general, the commands **N[operation]** and **operation // N** yield the same output.*

Mathematica can also solve equations involving more than one variable for one of the other variables in terms of other unknowns.



For example, here Mathematica solves

the equation $v = \frac{\pi r^2}{h}$ for h .

Or, in this case, Mathematica solves the

equation $a^2 + b^2 = c^2$ for a .

■ Numerical Approximation of Solutions of Equations

When solving an equation is either impractical or impossible, *Mathematica* provides two functions to approximate roots of equations: **FindRoot** and **NRoots**. **NRoots** numerically approximates the roots of any polynomial equation. **FindRoot** attempts to approximate a root to an equation provided that a "reasonable" guess of the root is given. **FindRoot**[*lhs==rhs*, {*x*, *firstguess*}] searches for a numerical solution to the equation *lhs==rhs*, starting with *x=firstguess*. (*firstguess* can be obtained by using the **Plot** command.) Thus, **FindRoot** works on functions other than polynomials. Moreover, to locate more than one root, **FindRoot** must be used several times. **NRoots** is easier to use when trying to approximate the roots of a polynomial.

□ Example:

Approximate the solutions of $x^5 + x^4 - 4x^3 + 2x^2 - 3x - 7 = 0$.

```

In[15]:=
  NRoots[
    x^5+x^4-4x^3+2x^2-3x-7==0, x]
Out[15]=
  x == -2.74463 || x == -0.880858 ||
  x == 0.41452 - 1.19996 I ||
  x == 0.41452 + 1.19996 I ||
  x == 1.79645
In[19]:=
  FindRoot[
    x^5+x^4-4x^3+2x^2-3x-7==0, {x, 1.8}]
Out[19]=
  {x -> 1.79645}
  
```

To obtain approximations of all solutions to the equation $x^5 + x^4 - 4x^3 + 2x^2 - 3x - 7 = 0$ use **NRoots**.

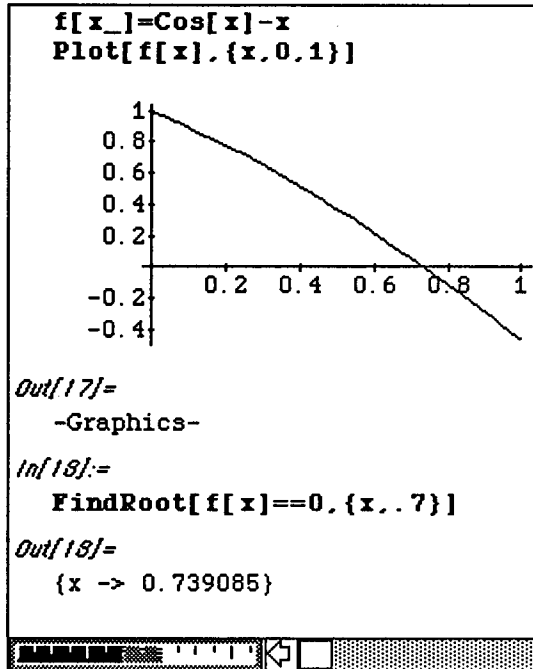
The symbol || means "Or."

FindRoot may also be used to approximate each root of the equation $x^5 + x^4 - 4x^3 + 2x^2 - 3x - 7 = 0$.

1.8 is our "first guess."

□ Example:

In order to approximate the roots of the equation $\text{Cos}(x)-x=0$, **FindRoot** must be used since $\text{Cos}(x)-x=0$ is not a polynomial equation.



To approximate the positive solutions to the equation $\text{Cos}(x)-x=0$, first define and graph $f(x)=\text{Cos}(x)-x$. Notice that $f(x)$ is zero near 0.7. Then Enter **FindRoot[f[x]==0,{x,.7}]** to approximate the root near 0.7.

In general, **FindRoot** will yield a single approximation to a solution of an equation provided that a "good" first approximation has been given; **NRroots** will give approximations of all roots of a polynomial equation.

● Approximating Solutions of Equation in Version 2.0

In addition to the commands **FindRoot** and **NRoots**, Version 2.0 contains the command **NSolve** which can also be used to approximate roots of some equations.

○ Example:

If $h(x) = x^3 - 8x^2 + 19x - 12$ and $k(x) = \frac{1}{2}x^2 - x - \frac{1}{8}$, use *Mathematica* Version 2.0 to compute approximations of the solution of $h(x) = k(x)$ using (i) **NRoots**; and (ii) **NSolve**.

```

In[20]:=
Clear[h,k]
h[x_]=x^3-8x^2+19x-12
k[x_]=1/2x^2-x-1/8
vals1=NRoots[h[x]==k[x],x]

Out[18]=
-12 + 19 x - 8 x^2 + x^3

Out[19]=
-(1/8) - x + x^2/2

Out[20]=
x == 0.904363 ||
x == 2.66088 || x == 4.93476

In[21]:=
vals2=NSolve[h[x]==k[x],x]

Out[21]=
{{x -> 0.904363},
 {x -> 2.66088},
 {x -> 4.93476}}

```

In Version 2.0, the command **NSolve** may also be used to numerically compute solutions to polynomial equations.

Notice that the difference between the result of using the command **NRoots** and **NSolve** is the form of the final output. These differences will be discussed in detail in Chapter 9.

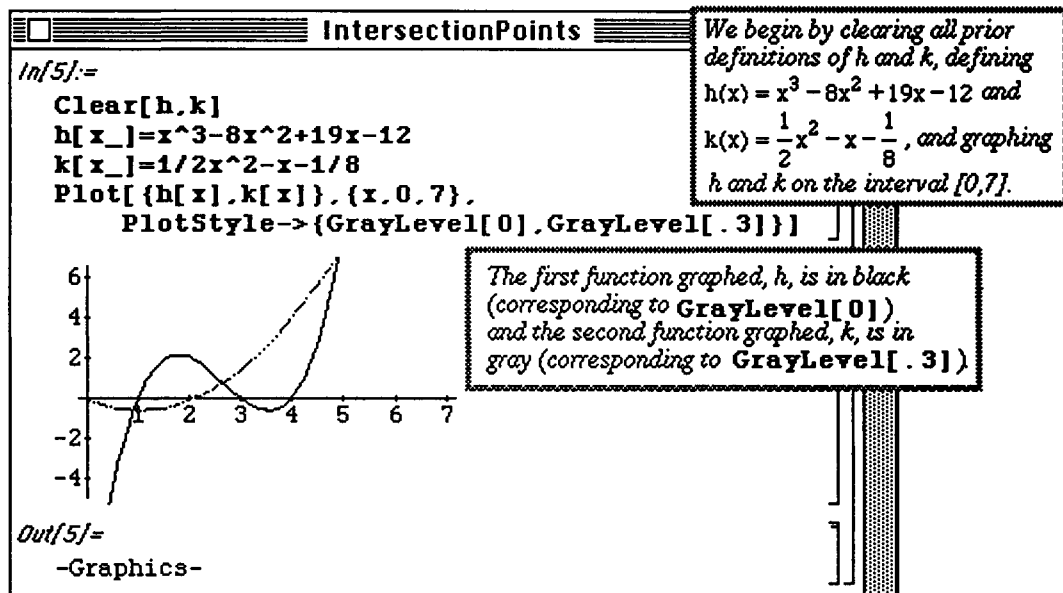
Even though three commands are entered simultaneously, Version 2.0 generates output for each command as it is evaluated.

■ **Application:** Intersection Points of Graphs of Functions

■ (A)

Locate the points where the graphs of $h(x) = x^3 - 8x^2 + 19x - 12$ and $k(x) = \frac{1}{2}x^2 - x - \frac{1}{8}$ intersect.

Notice that the x -coordinates of the intersection points satisfy the equation $h(x) = k(x)$. Consequently, to locate the intersection points, it is sufficient to solve the equation $h(x) = k(x)$. Although this step is not necessary to solve the problem, we first graph h and k and notice that h and k intersect three times.



Since $h(x) = k(x)$ is a polynomial equation of degree three, *Mathematica* can compute exact values of all three roots. However, the roots are complicated so we approximate the solutions. Moreover, since $h(x) = k(x)$ is a polynomial equation we use the command `NRroots[h[x]==k[x],x]`:

o Notice that in Version 2.0, `NSolve[h[x]==k[x],x]` produces the same result.

In the following example, the exact solutions of the equation $h(x)=k(x)$ are computed with the command `Solve[h[x]==k[x]]`. Notice that the resulting solution is expressed as a **list**. Lists are discussed in detail in Chapters 4 and 5. Nevertheless, the results of the command `Solve[lhs==rhs]` attempts to solve the equation $lhs=rhs$ for all variables that appear in the equation. The solutions, if any, are displayed as a list. In general, the command `Solve[lhs==rhs][[1]]` yields the first element of the list of solutions, `Solve[lhs==rhs][[2]]` yields the second element of the list of solutions, and `Solve[lhs==rhs][[j]]` yields the j th element of the list of solutions.

```
In[6]:=
```

```
NRroots[h[x]==k[x],x]
```

```
Out[6]=
```

```
x == 0.904363 || x == 2.66088 || x == 4.93476
```

```
In[7]:=
```

```
Solve[h[x]==k[x]]
```

```
Out[7]=
```

```
{ {x ->  $\frac{17}{6} + \frac{49}{36 \left( \frac{151}{432} + \frac{\text{Sqrt}[-16585]^{1/3}}{48 \text{Sqrt}[3]} \right)}$  +  $\frac{151}{432} + \frac{\text{Sqrt}[-16585]^{1/3}}{48 \text{Sqrt}[3]}$  }, {x ->  $\frac{17}{6} + \frac{49}{36 \left( \frac{151}{432} + \frac{\text{Sqrt}[-16585]^{1/3}}{48 \text{Sqrt}[3]} \right)}$  } }
```

Computes numerical approximations of all solutions to the polynomial equation $h(x)=k(x)$. Be sure to use "double-equals" signs when working with equations.

The command `Solve[h[x]==k[x]]` computes the exact solutions of the polynomial equation $h(x)=k(x)$. The command `Solve[h[x]==k[x]][[1]]` yields the exact value of the first solution; the command `Solve[h[x]==k[x]][[2]]` yields the exact value of the second solution; and the command `Solve[h[x]==k[x]][[3]]` yields the exact value of the third solution. Since the degree of the polynomial equation $h(x)=k(x)$ is 3, there are at most three distinct solutions.

■ (B)

Locate the points where the graphs of $f(x) = e^{-(x/4)^2} \cos\left(\frac{x}{\pi}\right)$ and $g(x) = \sin\left(x^{3/2}\right) + \frac{5}{4}$ intersect.

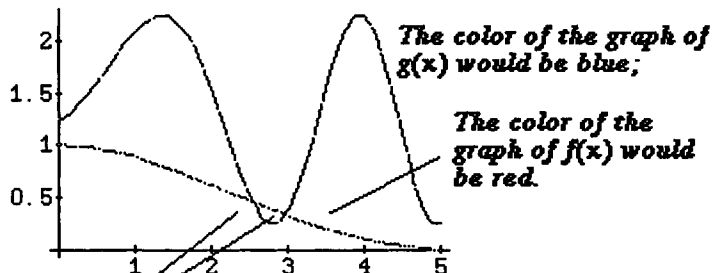
Notice that the x-coordinates of the intersection points satisfy the equation $f(x)=g(x)$. Consequently, to locate the intersection points, it is sufficient to solve the equation $f(x)=g(x)$. Since this problem does not involve polynomials, we must first graph f and g and notice that they intersect twice.

To locate the intersection points, first clear any prior definitions of f and g , define f and g , and graph:

Don't forget to place the underline on the left-hand side of the equals sign and enclose the arguments of all functions in square brackets.

In order to approximate the intersection points, we will use **FindRoot** to approximate solutions to the equation $f(x)=g(x)$.

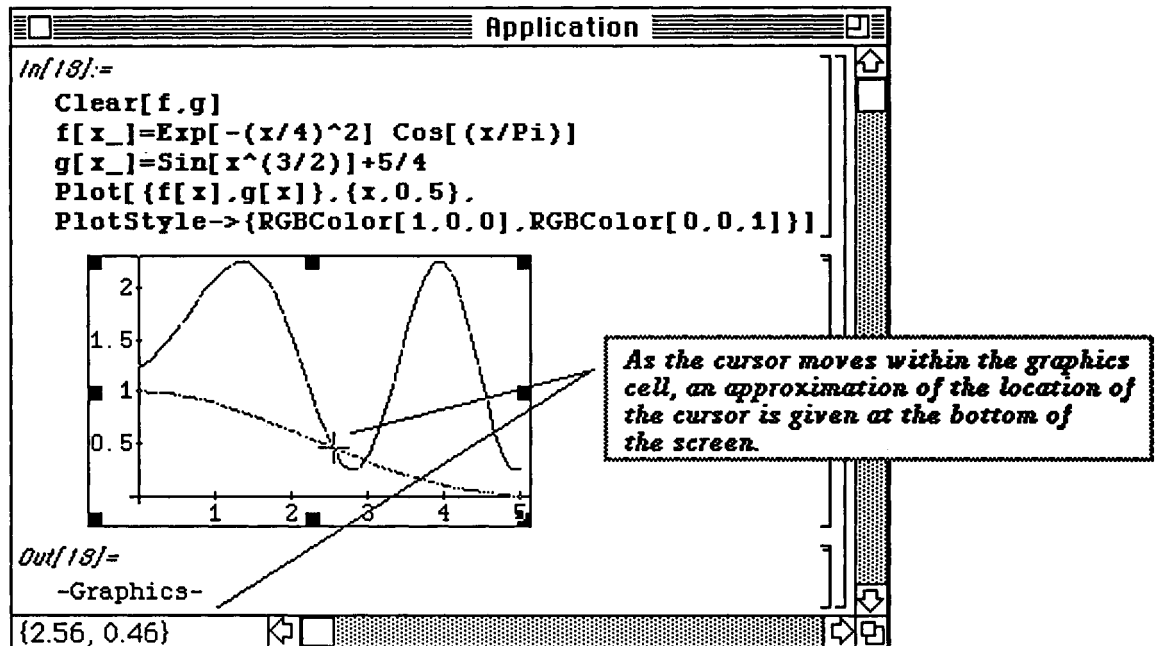
```
Clear[f,g]
f[x_]=Exp[-(x/4)^2] Cos[x/Pi]
g[x_]=Sin[x^(3/2)]+5/4
Plot[{f[x],g[x]},{x,0,5},
PlotStyle->{RGBColor[1,0,0],RGBColor[0,0,1]}]
```



Intersection points of the graphs of f and g

Mathematica cannot solve $f(x)=g(x)$ exactly. Since $f(x)=g(x)$ is NOT a polynomial equation, the command **NRroots** cannot be used to numerically approximate the roots. However, we can use the command **FindRoot** to approximate each root provided we have a "good" initial approximation of the root. To obtain a "good" initial approximation of each root:

- 1) Move the cursor within the graphics cell and click once. Notice that a box appears around the graph:
- 2) Press and hold down the Open-Apple key; as you move the cursor within the graphics cell, notice that the thermometer at the bottom of the screen has changed to ordered pairs approximating the location of the cursor within the graphics cell:



When the cursor is near the intersection point, we see that the x-coordinate of the point is approximately 2.56. Hence, we will use 2.56 as our initial approximation in the **FindRoot** command.

An approximation of the second intersection point is similarly obtained.

We then use **FindRoot** twice to compute an approximation of each solution:

The screenshot shows a Mathematica application window titled "Application". The main content area displays the following input and output:

```

In[25]:=
  FindRoot[f[x]==g[x].{x,2.59}]
Out[25]=
  {x -> 2.54105}
In[26]:=
  f[2.54105] // N
Out[26]=
  0.461103
In[27]:=
  FindRoot[f[x]==g[x].{x,2.98}]
Out[27]=
  {x -> 2.9746}
In[28]:=
  f[2.9746] // N
Out[28]=
  0.336066
  
```

Three callout boxes provide additional context:

- The first callout box (top right) states: *FindRoot[f[x]==g[x].{x,2.59}] computes an approximation of the solution to the equation $f(x)=g(x)$ near the value 2.59.*
- The second callout box (middle right) states: *FindRoot[f[x]==g[x].{x,2.98}] computes an approximation of the solution to the equation $f(x)=g(x)$ near the value 2.98.*
- The third callout box (bottom right) states: *We conclude that one intersection point is approximately (2.54105, 46113) and the other intersection point is approximately (2.9746, 336066).*

The application window includes a standard Mac OS-style title bar with a close button on the right and a status bar at the bottom with various utility icons.

Chapter 3 Calculus

- Chapter 3 introduces *Mathematica*'s built-in calculus commands. The examples used to illustrate the various commands are similar to examples routinely done in first-year calculus courses.
- Commands introduced and discussed in this chapter from Version 1.2 are:

The command `Chop[smallnumber]` produces zero when $|\text{smallnumber}| \leq 10^{-10}$

Other commands include:

Limits

`Limit[expression, x->a]`

Differential Calculus

`f'[x]`

`f''[x]`

`f'''[x]`

`D[f[x],x]`

`D[f[x],{x,n}]`

`Dt[equation,x]`

Integral Calculus

`Integrate[expression,x]`

`Integrate[expression, {x,a,b}]`

`NIntegrate[expression, {x,a,b}]`

Graphics

`ContourPlot[f[x,y],{x,a,b},{y,c,d}]`

`Plot3D[f[x,y],{x,a,b},{y,c,d},options]`

Options

`PlotPoints->n`

`Shading->False`

`Boxed->False`

`PlotLabel->"text"`

`AxesLabel->{"x-axis text","y-axis text","z-axis text"}`

`Ticks->None`

`Ticks->{{x-axis tick marks},{y-axis tick marks},{z-axis tick marks}}`

`Axes->None`

`BoxRatios->{a,b,c}`

`Mesh->False`

`DisplayFunction->Identity`

Multi-Variable Calculus

`D[f[x,y],x]`

`D[f[x,y],y]`

`D[f[x,y],{x,n}]`

`D[f[x,y],{y,n}]`

`D[f[x,y],x,y]`

`Derivative[n,m][f][x,y]`

`Integrate[f[x,y],{x,a,b},{y,c,d}]`

`NIntegrate[f[x,y],{x,a,b},{y,c,d}]`

Series Calculus

`Series[f[x],{x,a,n}]`

`Normal[Series[f[x],{x,a,n}]]`

`LogicalExpand[series1==series2]`

- Commands and options discussed in this chapter from Version 2.0 include:

Graphics

ImplicitPlot[equation, {x, xmin, xmax}, {y, ymin, ymax}]

Version 2.0 Graphics Options Include:

ContourSmoothing	}	ContourPlot	Show	GraphicsArray	} Plot3D
ContourShading					
Contours					

Other Options:
Direction -> ±1 } **Limit**

- Applications in this chapter include:

- Differential Calculus

- Locating Horizontal Tangent Lines
- Graphing Functions and Tangent Lines
- Maxima and Minima

- Integral Calculus

- Area between Curves
- Volumes of Solids of Revolution
- Arc Length

- Series

- Approximating the Remainder
- Computing Series Solutions to Differential Equations

- Multi-Variable Calculus

- Classifying Critical Points
- Tangent Planes
- Volume

3.1 Computing Limits

One of the first topics discussed in calculus is that of limits. *Mathematica* uses the command **Limit[expression, x->a]** to find the limit of **expression** as **x** approaches the value **a**, where **a** can be a finite number, positive infinity (**Infinity**), or negative infinity (**-Infinity**). The "**->**" is obtained by typing a minus sign "**-**" followed by a greater than sign "**>**".

□ Example:

Use *Mathematica* to compute (i) $\lim_{x \rightarrow -3} \frac{3x^2 + 4x - 15}{13x^2 + 32x - 21}$; (ii) $\lim_{x \rightarrow 0} \frac{\sin(x)}{x}$; (iii) $\lim_{x \rightarrow \infty} \frac{50 - 17x^2}{200x + 3x^2}$;

and (iv) $\lim_{x \rightarrow \infty} \frac{3 - x^2}{4 - 1000x}$.

The screenshot shows the Mathematica interface for the Limits window. It displays four input-output pairs for the Limit function, with corresponding mathematical expressions shown to the right of the window.

Input	Output	Mathematical Expression
<code>In[5]:= Limit[(3x^2+4x-15)/(13x^2+32x-21), x->-3]</code>	<code>Out[5]= 7/23</code>	$\lim_{x \rightarrow -3} \frac{3x^2 + 4x - 15}{13x^2 + 32x - 21}$
<code>In[6]:= Limit[Sin[x]/x, x->0]</code>	<code>Out[6]= 1</code>	$\lim_{x \rightarrow 0} \frac{\sin(x)}{x}$
<code>In[7]:= Limit[(50-17x^2)/(200x+3x^2), x->Infinity]</code>	<code>Out[7]= 17/(-3)</code>	$\lim_{x \rightarrow \infty} \frac{50 - 17x^2}{200x + 3x^2}$
<code>In[8]:= Limit[(3-x^2)/(4-1000x), x->Infinity]</code>	<code>Out[8]= Infinity</code>	$\lim_{x \rightarrow \infty} \frac{3 - x^2}{4 - 1000x}$

The **Limit** command can also be used along with **Simplify** to assist in determining the derivative of a function by using the definition of the derivative. This is illustrated in the following example. (This example also shows that an expression can be assigned any name, as long as that name is not a built-in *Mathematica* function or constant.)

Remember: Since every built-in *Mathematica* object begins with a capital letter, we have adopted the convention that all user-defined objects will be named using lower-case letters.)

□ Example:

Let $g(x) = x^3 - x^2 + x + 1$. Use *Mathematica* to compute and simplify (i) $\frac{g(x+h) - g(x)}{h}$;

and (ii) $\lim_{h \rightarrow 0} \frac{g(x+h) - g(x)}{h}$.

```

Limits
In[17]:=
Clear[g]
g[x_]=x^3-x^2+x+1
Out[17]=
      2      3
1 + x - x + x
In[19]:=
dog=Simplify[(g[x+h]-g[x])/h]
Out[19]=
      2      2
1 - h + h - 2 x + 3 h x + 3 x
In[20]:=
Limit[dog,h->0]
Out[20]=
      2
1 - 2 x + 3 x

```

dog is the simplified
expression $\frac{g(x+h) - g(x)}{h}$.

Limit[dog,h->0]
computes $\lim_{h \rightarrow 0} \frac{g(x+h) - g(x)}{h}$.

The next example illustrates how several *Mathematica* commands can be combined in a single statement to obtain the desired result.

□ Example:

Let $f(x) = \frac{1}{\sqrt{x}} + \sqrt{x}$. Use *Mathematica* to compute and simplify (i) $\frac{f(x+h)-f(x)}{h}$;

and (ii) $\lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h}$.

Remember that *Mathematica* denotes \sqrt{x} by `Sqrt[x]`. Hence, f is defined by

`f[x_]=1/Sqrt[x]+Sqrt[x]`. Entering `f[x_]=x^(-1/2)+x^(1/2)` would yield the same result.

<pre>In[9]:= Clear[f] f[x_]=1/Sqrt[x]+Sqrt[x] Out[9]= 1 ----- + Sqrt[x] Sqrt[x] In[11]:= quotient=Together[(f[x+h]-f[x])/h] Out[11]= 3/2 (Sqrt[x] + h Sqrt[x] + x - Sqrt[h + x] - x Sqrt[h + x]) / (h Sqrt[x] Sqrt[h + x]) In[13]:= Limit[quotient,h->0] Out[13]= -1 Sqrt[x] ----- + ----- 2 Sqrt[x] 2 x</pre>	<p>quotient is the expression</p> $\frac{f(x+h)-f(x)}{h}$ <p>Limit[quotient,h->0] calculates $\lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h}$.</p>
---	--

Note that the square brackets must be properly nested in order to correctly perform the combined operations. Also note that **Simplify** can be used to express the result in a more reasonable form:

Notice that several Mathematica commands can be combined. Be sure that square brackets are nested appropriately.

yields a simplified version of $\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$.

`In[14]:=`
`Simplify[Limit[quotient, h->0]]`
`Out[14]=`

$$\frac{-1 + x}{2x}$$

Sometimes Version 1.2 yields surprising results. For example, if f is an unknown function, `Limit[f[x], x->a]` yields $f[a]$.

In other cases, the command `Limit` returns results that do not make sense:

Version 1.2 evaluates the limit even when it encounters an unknown function and in some cases yields results that don't make sense.

`In[14]:=`
`Clear[f]`
`?f`
`f`
`In[15]:=`
`Limit[f[x], x->a]`
`Out[15]=`
 $f[a]$
`In[16]:=`
`Limit[Abs[x]/x, x->0]`
`Out[16]=`
 $Abs'[0]$

• Computing Limits with Version 2.0

In Version 2.0, the command `Limit` does not evaluate when it encounters an unknown function unless the option `Analytic->True` is included.

Version2.0Limits

```

In[20]:=
  Clear[f]
  ?f
  Global`f

In[21]:=
  Limit[f[x], x->a]

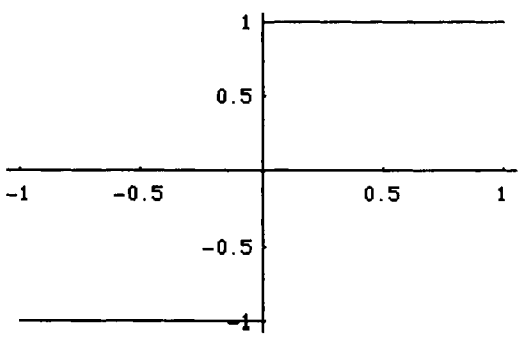
Out[21]=
  Limit[f[x], x -> a]

In[22]:=
  Limit[f[x], x->a, Analytic->True]

Out[22]=
  f[a]

In[23]:=
  Plot[Abs[x]/x, {x, -1, 1}]

Out[23]=
  -Graphics-
  
```



Since `Clear[f]` removes all prior definitions of `f`, `f[x]` is an undefined function.

In Version 2.0, `Limit` does NOT evaluate when it encounters an unknown function unless the option `Analytic->True` is included.

In any case, `Limit` is unable to evaluate many well-known limits.

For example, even though

$\lim_{x \rightarrow 0} \frac{|x|}{x}$ does not exist,

$\lim_{x \rightarrow 0^-} \frac{|x|}{x} = -1$ and $\lim_{x \rightarrow 0^+} \frac{|x|}{x} = 1$.

In addition, in some cases, the options `Direction->1` or `Direction->-1` may help in computing limits. The command `Limit[f[x], x->a, Direction->-1]` computes

$\lim_{x \rightarrow a^+} f[x]$; the command `Limit[f[x], x->a, Direction->1]` computes $\lim_{x \rightarrow a^-} f[x]$.

The screenshot shows three input-output pairs in Mathematica:

- `In[24]:= Limit[Abs[x]/x, x->0]`
`Out[24]= Limit[Abs[x], x -> 0]`
- `In[25]:= Limit[Abs[x]/x, x->0, Direction->-1]`
`Out[25]= Limit[Abs[x], x -> 0, Direction -> -1]`
- `In[26]:= Limit[Abs[x]/x, x->0, Direction->1]`
`Out[26]= Limit[Abs[x], x -> 0, Direction -> 1]`

Annotations to the right of the screenshot:

- Next to the first two outputs: *Mathematica can neither compute*
- Next to the second output: $\lim_{x \rightarrow 0^+} \frac{|x|}{x} = 1$
- Next to the third output: *nor can Mathematica compute*
- Next to the third output: $\lim_{x \rightarrow 0^-} \frac{|x|}{x} = -1$

However, Version 2.0 does correctly compute $\lim_{x \rightarrow 0^+} \frac{1}{x}$ and $\lim_{x \rightarrow 0^-} \frac{1}{x}$.

The screenshot shows two input-output pairs in Mathematica:

- `In[13]:= Limit[1/x, x->0, Direction->-1]`
`Out[13]= Infinity`
- `In[14]:= Limit[1/x, x->0, Direction->1]`
`Out[14]= -Infinity`

Annotations to the right of the screenshot:

- Next to the first output: *computes* $\lim_{x \rightarrow 0^+} \frac{1}{x}$
- Next to the second output: *computes* $\lim_{x \rightarrow 0^-} \frac{1}{x}$

■ 3.2 Differential Calculus

■ Calculating Derivatives of Functions and Expressions

If we are given a differentiable function $f(x)$, *Mathematica* can compute the derivative of $f(x)$ in at least two ways once $f(x)$ has been properly defined using *Mathematica* :

- 1) The command $f' [x]$ computes the derivative of $f[x]$ with respect to x .
- 2) The command $D[f[x], x]$ computes the derivative of $f[x]$ with respect to x .
- 3) The command $D[f[x], \{x, n\}]$ computes the n th derivative of $f[x]$ with respect to x .
- 4) The command $D[\text{expression}, \text{variable}]$ computes the derivative of **expression** with respect to **variable**.

Other ways *Mathematica* can compute derivatives of functions and expressions are discussed in **Section 3.6**.

□ Example:

For example, in order to compute the derivative of $(7x-3)^3(5-4x^2)^2$, we may either directly compute the derivative of $(7x-3)^3(5-4x^2)^2$ or we may define $h(x) = (7x-3)^3(5-4x^2)^2$ and compute $h'(x)$.

Derivative	
<code>In[1]:=</code>	<code>D[(7x-3)^3(5-4x^2)^2, x]</code>
<code>Out[1]=</code>	$-16x(-3+7x)^3(5-4x^2) + 21(-3+7x)^2(5-4x^2)^2$
<code>In[2]:=</code>	<code>Clear[h]</code>
	<code>h[x_]=(7x-3)^3(5-4x^2)^2;</code>
<code>In[3]:=</code>	<code>h'[x]</code>
<code>Out[3]=</code>	$-16x(-3+7x)^3(5-4x^2) + 21(-3+7x)^2(5-4x^2)^2$

`D[(7x-3)^3(5-4x^2)^2, x]`
computes the derivative (with respect to x) of the expression
 $(7x-3)^3(5-4x^2)^2$.

A semi-colon placed at the end of a command suppresses the output. Remember that RETURN gives a new line; ENTER evaluates Mathematica input.

`Clear[h]`
first clears all previous definitions of h and then defines
 $h(x) = (7x-3)^3(5-4x^2)^2$.

`h'[x]` *computes the derivative of h.*

Notice that both $h'[x]$ and $D[h[x], x]$ produce the same result.

The screenshot shows a Mathematica notebook window titled "Derivative". It contains three input-output pairs:

Input 4: `D[h[x], x]`

Output 4:
$$-16x^3(-3 + 7x) + (5 - 4x)^2 + 21(-3 + 7x)^2(5 - 4x)^2$$

Input 5: `Factor[h'[x]]`

Output 5:
$$(-3 + 7x)^2(-5 + 4x)^2(-105 - 48x + 196x^2)$$

Input 6: `Factor[D[(7x-3)^3(5-4x^2)^2, x]]`

Output 6:
$$(-3 + 7x)^2(-5 + 4x)^2(-105 - 48x + 196x^2)$$

$D[h[x], x]$ also computes the derivative of $h(x)$ with respect to x .

Remember that *Mathematica* commands can be nested; be sure that square-brackets are nested correctly.

`Factor[h'[x]]` first computes then factors the derivative of $h(x)$.

The exact same result is obtained by executing the command

`Factor[D[(7x-3)^3(5-4x^2)^2, x]]`.

■ Graphing Functions and Derivatives

Moreover, since $f[x]$ is a function of x , $f'[x]$ can be graphed. The following example shows how to compute the derivative of a function and then plot the original function and its derivative simultaneously :

□ Example:

Let $f(x) = x\sin^2(x)$. Compute f' and graph both f and f' on the same axes.

Derivatives

```

Inf 7]:=
  Clear[f]
  f[x_]=x (Sin[x])^2
Out 7]=
      2
x Sin[x]
Inf 8]:=
  f'[x]
Out 8]=
      2
2 x Cos[x] Sin[x] + Sin[x]
Inf 9]:=
  D[f[x],x]
Out 9]=
      2
2 x Cos[x] Sin[x] + Sin[x]
Inf 10]:=
  Plot[{f[x],f'[x]},{x,-Pi,Pi},
  PlotStyle->{GrayLevel[0],GrayLevel[.3]}]
Out 10]=
-Graphics-
```

First clear prior definitions of f . In this example, define $f(x)$ to be $f(x) = x\sin^2(x)$.

$f'[x]$ computes
 $f'(x) = \frac{d}{dx}f(x) = D_x(f(x))$.

$D[f[x],x]$ also computes
 $f'(x) = \frac{d}{dx}f(x) = D_x(f(x))$.

Use **Plot** to graph $f(x)$ (in black) and $f'(x)$ (in gray) on the same axes.

■ Computing Higher Order Derivatives

The command `D[f[x], {x, n}]` computes the n th derivative of $f[x]$ with respect to x :

$$f^{(n)}(x) = \frac{d^n f(x)}{dx^n}.$$

Using the same definition of f as above, the following calculations compute the second, third, and fourth derivatives of f :

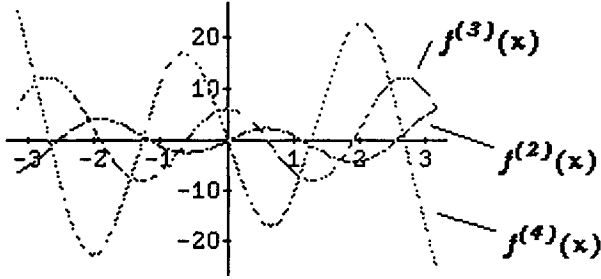
<pre>In[2]:= second=D[f[x], {x, 2}] Out[2]= 2 2 x Cos[x] + 4 Cos[x] Sin[x] - 2 2 x Sin[x]</pre>	<pre>second=D[f[x], {x, 2}]</pre>	<p>computes $f''(x) = \frac{d^2}{dx^2}f(x)$ and names it second.</p>
<pre>In[3]:= third=D[f[x], {x, 3}] Out[3]= 2 6 Cos[x] - 8 x Cos[x] Sin[x] - 2 6 Sin[x]</pre>	<pre>third=D[f[x], {x, 3}]</pre>	<p>computes $f^{(3)}(x) = \frac{d^3}{dx^3}f(x)$ and names it third.</p>
<pre>In[4]:= fourth=D[f[x], {x, 4}] Out[4]= 2 -8 x Cos[x] - 32 Cos[x] Sin[x] + 2 8 x Sin[x]</pre>	<pre>fourth=D[f[x], {x, 4}]</pre>	<p>computes $f^{(4)}(x) = \frac{d^4}{dx^4}f(x)$ and names it fourth.</p>

These higher order derivatives can be graphed together. Recall that any expression in *Mathematica* can be assigned a name. The names `plotsecond`, `plotthird`, and `plotfourth` are given to the graphs of the second, third, and fourth derivatives of the function $f(x)$, respectively. By naming these graphs individually, the **Show** command can be used to plot the three graphs at one time. Notice the relationship of the behavior of the graphs of these three derivatives.

Remember, $f^{(3)}(x)$ is the derivative of $f^{(2)}(x)$, so $f^{(3)}(x) > 0$ when $f^{(2)}(x)$ is increasing and $f^{(3)}(x) < 0$ when $f^{(2)}(x)$ is decreasing.

Also, $f^{(4)}(x)$ is the second derivative of $f^{(2)}(x)$, so $f^{(4)}(x) > 0$ when $f^{(2)}(x)$ is concave up and $f^{(4)}(x) < 0$ when $f^{(2)}(x)$ is concave down.

Notice how **GrayLevel** is used to distinguish between the three curves; the option **DisplayFunction->Identity** suppresses the resulting graph; and the option **DisplayFunction->\$DisplayFunction** is used in the Show command to display the graphs which were suppressed initially with the option **DisplayFunction->Identity**:

<pre>In[21]:= plotsecond=Plot[second, {x, -Pi, Pi}, PlotStyle->GrayLevel[.2], DisplayFunction->Identity] Out[21]= -Graphics-</pre>	<pre>In[22]:= plotthird=Plot[third, {x, -Pi, Pi}, PlotStyle->GrayLevel[.4], DisplayFunction->Identity] Out[22]= -Graphics-</pre>	<pre>In[23]:= plotfourth=Plot[fourth, {x, -Pi, Pi}, PlotStyle->GrayLevel[.6], DisplayFunction->Identity] Out[23]= -Graphics-</pre>	<pre>In[25]:= Show[plotsecond, plotthird, plotfourth, DisplayFunction->\$DisplayFunction] Out[25]= -Graphics-</pre>	<p>plotsecond is the graph of second on the interval $[-\pi, \pi]$; the graph is not displayed.</p> <p>plotthird of third on the interval $[-\pi, \pi]$; the graph is not displayed.</p> <p>plotfourth of fourth on the interval $[-\pi, \pi]$; the graph is not displayed.</p> <p>To display all three graphs simultaneously, use the Show command.</p>
				

■ Locating Critical Points and Inflection Points

Since derivatives of functions are expressions, algebraic procedures can be performed on them. Hence, in addition to finding the roots of a function, $g(x)$, *Mathematica* can also be used to locate the critical points and inflection points of $g(x)$.

In order to observe the location of these points, the **Plot** command is used to graph $g(x)$, $g'(x)$, and $g''(x)$ simultaneously.

□ Example:

Let $g(x) = 2x^3 - 9x^2 + 12x$. Graph g , g' , and g'' on the interval $[-1, 4]$. Locate all critical points and inflection points

The screenshot shows the Mathematica interface with the following input and output:

```

In[7]:=
Clear[g]
g[x_]=2x^3-9x^2+12x
Out[7]=
      2      3
12 x - 9 x + 2 x

In[8]:=
g'[x]
Out[8]=
      2
12 - 18 x + 6 x

In[9]:=
g''[x]
Out[9]=
-18 + 12 x

In[10]:=
Plot[{g[x].g'[x].g''[x]},{x,-1,4},
PlotStyle->{GrayLevel[0].GrayLevel[.3].
GrayLevel[0]}]
Out[10]=
-Graphics-
  
```

The plot shows three curves on the interval $[-1, 4]$: $g(x)$ (solid black line), $g'(x)$ (dashed gray line), and $g''(x)$ (dotted black line). The x-axis ranges from -1 to 4, and the y-axis ranges from -15 to 20. The origin is marked with 0. The curves intersect at several points, including the origin (0,0).

First clear any prior definitions of g ; then define the function $g(x)$

$$g(x) = 2x^3 - 9x^2 + 12x.$$

$g'[x]$ computes $g'(x) = \frac{d}{dx}g(x)$.
Notice the same result would have been obtained by the command $D[g[x], x]$.

$g''[x]$ computes $g''(x) = \frac{d^2}{dx^2}g(x)$. The same result would have been obtained by the command $D[g[x], \{x, 2\}]$.

We use the **Plot** command to display the graphs of $g(x)$ (in black), $g'(x)$ (in gray), and $g''(x)$ (in black) on the same axes.

Solving each of the equations $g(x)=0$, $g'(x)=0$, and $g''(x)=0$ locates the roots of g , the x -coordinates of the critical points of g , and the x -coordinates of the inflection points of g . Since g is a polynomial with degree less than five, these three equations can be solved with the **Solve** command by entering **Solve[g[x]==0]**, **Solve[g'[x]==0]** and **Solve[g''[x]==0]**. Be sure to include the double-equals sign between the left- and right-hand sides of equations when using the **Solve** command:

<pre>In[12]:= Factor[g'[x]] Out[12]= 6 (-2 + x) (-1 + x)</pre>	<pre>Factor[g'[x]] factors g'(x).</pre>
<pre>In[13]:= Solve[g[x]==0] Out[13]= {{x -> 0}, {x -> $\frac{9 + \text{Sqrt}[-15]}{4}$}, {x -> $\frac{9 - \text{Sqrt}[-15]}{4}$}}</pre>	<pre>Solve[g[x]==0] solves the equation g(x)=0.</pre>
<pre>In[14]:= Solve[g'[x]==0] Out[14]= {{x -> 2}, {x -> 1}}</pre>	<pre>Solve[g'[x]==0] solves the equation g'(x)=0.</pre>
<pre>In[15]:= Solve[g''[x]==0] Out[15]= {{x -> $\frac{3}{2}$}}</pre>	<pre>Solve[g''[x]==0] solves the equation g''(x)=0.</pre>

We conclude that the critical points of g are $(1, g(1))$ and $(2, g(2))$; the inflection point is $(3/2, g(3/2))$.

A similar type of problem which can be solved using *Mathematica* is as follows :

□ **Example:**

Locate the values of x for which the line tangent to the graph

$$p(x) = \frac{1}{2}x^6 - 2x^5 - \frac{25}{2}x^4 + 60x^3 - 150x^2 - 180x - 25 \text{ is horizontal.}$$

Notice that the function $p(x)$ is a polynomial of degree 6, so $p'(x)$ is a polynomial of degree 5. Therefore, when determining the values of x such that $p'(x) = 0$, the command `NRoots` must be used instead of `Solve`. (Recall that `Solve` finds exact solutions of polynomial equations of degree four or less.) Some of the roots of $p'(x)$ are complex numbers. These values are ignored since we are only concerned with real numbers.

Derivatives

```

In[42]:=
  p[x_]:=x^6 /2-2x^5-25 x^4/2+60x^3-
  150x^2-180x-25
    
```

First define p(x).

```

Out[42]=
      2      3
-25 - 180 x - 150 x + 60 x -
      4      6
  25 x      .5 x
  ----- - 2 x + --
      2      2
    
```

Notice that $p(x)$ is defined using two lines and Mathematica accepts the definition. In general, Mathematica will "read" each line until the command makes sense. Consequently, since the first line does not define a function (because of the minus sign at the end), Mathematica reads the second line and the command makes sense.

```

In[45]:=
  Plot[{p[x].p'[x]},{x,-6,6},
  PlotStyle->{GrayLevel[0].GrayLevel[.3]}]
    
```

Graph p(x) (in black) and p'(x) (in gray) on the same axes.

```

Out[45]=
-Graphics-
    
```

```

In[46]:=
  NRoots[p'[x]==0,x]
    
```

```

Out[46]=
  x == -4.44315 || x == -0.459096 ||
  x == 1.55293 - 1.82277 I ||
  x == 1.55293 + 1.82277 I ||
  x == 5.12971
    
```

The values of x for which the line tangent to the graph of f at the point $(x, f(x))$ is horizontal are (approximately) -4.44315 , -0.459096 , and 5.12971 .

Even though $p''[x]$ is a polynomial of degree 4, the inflection points are found using the command **NRoots**. (These points can also be determined with the **Solve** command.)

- o If you are using Version 2.0, **NSolve** can also be used to approximate the solutions of the polynomial equation $p''(x)=0$ by entering **NSolve** [$p''[x]==0, x$].

The screenshot shows a Mathematica notebook window titled "Derivatives". It contains two input-output pairs. The first input is `In[47]:= NRoots[p''[x]==0, x]`, and the output is `Out[47]=` followed by five solutions for x: `x == -3.25388 ||`, `x == 0.969666 - 0.776932 I ||`, `x == 0.969666 + 0.776932 I ||`, and `x == 3.98122`. The second input is `In[50]:= Factor[p'''[x]]`, and the output is `Out[50]=` followed by the factored polynomial `60 (-3 + x) (-1 + x) (2 + x)`. To the right of the notebook, there are two text annotations. The first, in a dotted box, says: "Since $p''(x)$ is also a polynomial, the command **NRoots**[$p''[x]==0, x$] solves the equation $p''(x)=0$." The second, in a solid box, says: "When using Mathematica to solve equations, don't forget to include the 'double-equals' sign between the left-hand side and the right-hand side of the equation." Below the notebook, there is a third text annotation: **Factor**[$p'''[x]$] factors $p'''(x)$.

Up to this point in our example problems, we have only considered polynomial functions. This next example involves a function which is not a polynomial. Hence, the **FindRoot** command, which depends on an initial guess, must be employed.

□ Example:

Let $w(x) = 2\sin^2(2x) + \frac{5}{2}x\cos^2\left(\frac{x}{2}\right)$ on $(0, \pi)$. Locate the values of x for which the line tangent to the graph of w at the point $(w, w(x))$ is horizontal.

Derivatives

```
In[1]:=
Clear[w]
w[x_]:=2(Sin[2x])^2+5/2x(Cos[x/2])^2
Out[1]=
      x 2
5 x Cos[-]
----- + 2 Sin[2 x] 2
      2
```

```
In[5]:=
Plot[w[x], {x, 0, Pi}]
Out[5]=
-Graphics-
```

```
In[6]:=
Plot[w'[x], {x, 0, Pi},
PlotStyle->GrayLevel[.3]]
Out[6]=
-Graphics-
```

The screenshot shows a Mathematica notebook window titled "Derivatives". It contains the following content:

- In[1]:** `Clear[w]` and `w[x_]:=2(Sin[2x])^2+5/2x(Cos[x/2])^2`. **Out[1]:** The derivative of $w(x)$ is displayed as $\frac{5x^2 \cos^2(x/2)}{2} + 2 \sin^2(2x)$.
- In[5]:** `Plot[w[x], {x, 0, Pi}]`. **Out[5]:** A plot of $w(x)$ on the interval $[0, \pi]$. The x-axis ranges from 0 to 3, and the y-axis from 0 to 3.5. Three points on the curve are marked with horizontal tangent lines. A callout box points to these points with the text: "Tangent lines are horizontal at these three points."
- In[6]:** `Plot[w'[x], {x, 0, Pi}, PlotStyle->GrayLevel[.3]]`. **Out[6]:** A plot of the derivative $w'(x)$ on the interval $[0, \pi]$. The x-axis ranges from 0 to 3, and the y-axis from -4 to 6. The curve crosses the x-axis at three points. A callout box points to these points with the text: "Use these x-values as initial approximations in FindRoot."

First clear prior definitions of w and define w .

After defining w , graph w on the interval $[0, \pi]$. Notice that w has three points for which the tangent line is horizontal.

The x -coordinates of the points on the graph of w for which the tangent line is horizontal are the values of x for which $w'(x)$ is zero. Since the equation $w'(x)$ is neither an equation Mathematica can solve exactly nor a polynomial, to approximate the solutions to the equation $w'(x) = 0$, we will use

FindRoot.

In order to use **FindRoot** we must have initial approximations of the solutions to $w'(x) = 0$. We obtain initial approximations by graphing $w'(x)$.

After using the graph of $w'(x)$ to find the initial guesses, the x -values such that $w'(x) = 0$ can be approximated using **FindRoot**. These three calculations are given below using initial guesses $x=.863$, $x=1.63$, $x=2.25$, the values where $w'(x)$ appears to cross the x -axis.

```

Derivatives
In[7]:=
  FindRoot[w'[x]==0, {x, .863}]
Out[7]=
  {x -> 0.864194}
In[8]:=
  FindRoot[w'[x]==0, {x, 1.63}]
Out[8]=
  {x -> 1.62391}
In[9]:=
  FindRoot[w'[x]==0, {x, 2.25}]
Out[9]=
  {x -> 2.24489}

```

*In this case, we take our "initial guesses" in **FindRoot** to be .863, 1.63, and 2.25.*

■ Application: Graphing Functions and Tangent Lines

An equation of the line with slope m that passes through the point (a,b) is given by the relationship $y-b=m(x-a)$. To write a function that describes the line with slope m that passes through the point note that $y-b=m(x-a)$ is equivalent to the statement $y=m(x-a)+b$.

The following example illustrates how the tangent line to the graph of a function can be determined and plotted simultaneously with the function.

□ Example:

Let $h(x) = \sin(6x) + 2\cos(2x)$. Graph h , the line tangent to the graph of h when $x = \frac{\pi}{3}$, and the line tangent to the graph of h when $x = \frac{2\pi}{3}$ on the interval $[0, \pi]$.

```

Derivatives
In[7]:=
Clear[h, y1, y2]
h[x_] = Sin[6x] + 2Cos[2x]
Out[7]=
2 Cos[2 x] + Sin[6 x]
In[8]:=
h'[x]
Out[8]=
6 Cos[6 x] - 4 Sin[2 x]
In[9]:=
h'[Pi/3]
Out[9]=
6 - 2 Sqrt[3]
In[11]:=
h[Pi/3]
Out[11]=
-1
In[12]:=
h'[2Pi/3]
Out[12]=
6 + 2 Sqrt[3]
In[13]:=
h[2 Pi/3]
Out[13]=
-1

```

Begin by clearing all prior definitions of h , $y1$, and $y2$. Then define $h(x) = \sin(6x) + 2\cos(2x)$.

$h'[x]$ calculates $h'(x)$.

$h'[Pi/3]$ computes $h'\left(\frac{\pi}{3}\right)$ which is the slope of the line tangent to the graph of h at the point $\left(\frac{\pi}{3}, h\left(\frac{\pi}{3}\right)\right)$.

$h[Pi/3]$ computes $h\left(\frac{\pi}{3}\right)$.

$h'[2Pi/3]$ calculates $h'\left(\frac{2\pi}{3}\right)$ which is the slope of the line tangent to the graph of h at the point $\left(\frac{2\pi}{3}, h\left(\frac{2\pi}{3}\right)\right)$.

$h[2 Pi/3]$ calculates $h\left(\frac{2\pi}{3}\right)$.

Consequently, a linear function tangent to the graph of h at the point

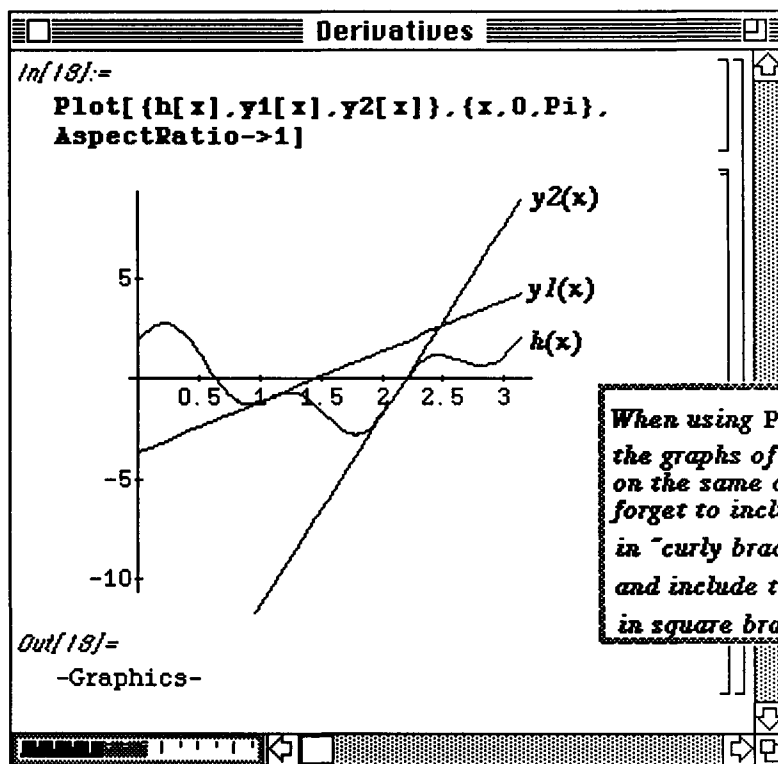
$\left(\frac{\pi}{3}, -1\right)$ is given by $y_1(x) = (6 - 2\sqrt{3})\left(x - \frac{\pi}{3}\right) - 1$ and a linear function tangent to the graph of

h at the point $\left(\frac{2\pi}{3}, -1\right)$ is given by $y_2(x) = (6 + 2\sqrt{3})\left(x - \frac{\pi}{3}\right) - 1$.

These are defined with *Mathematica* as follows:

<pre>In[15]:= Y1[x_]=(6-2Sqrt[3])(x-Pi/3)+-1 Out[15]= -1 + (6 - 2 Sqrt[3]) (----- + x) -Pi 3</pre>		<p>The graph of $Y1[x]$ is a line tangent to the graph of h at the point $\left(\frac{\pi}{3}, h\left(\frac{\pi}{3}\right)\right)$.</p>
<pre>In[16]:= Y2[x_]=h'[2Pi/3](x-2Pi/3)+h[2Pi/3] Out[16]= -1 + (6 + 2 Sqrt[3]) (----- + x) -2 Pi 3</pre>		<p>The graph of $Y2[x]$ is a line tangent to the graph of h at the point $\left(\frac{2\pi}{3}, h\left(\frac{2\pi}{3}\right)\right)$.</p>

The two lines $y_1[x]$ and $y_2[x]$ are plotted with the function $h[x]$ with the single command below. Note that the option **AspectRatio** \rightarrow 1 is used in the **Plot** command. If this option is not used with this particular function, then the graph is difficult to read (and the lines do not appear tangent to the curve).



Use **Plot** to display the graphs of $h[x]$, $y_1[x]$, $y_2[x]$ simultaneously.

If the option **AspectRatio** \rightarrow 1 had not been included, the graph of h would have been difficult to see.

When using **Plot** to display the graphs of several functions on the same axes, don't forget to include the functions in "curly brackets" ({ }) and include the entire **Plot** in square brackets.

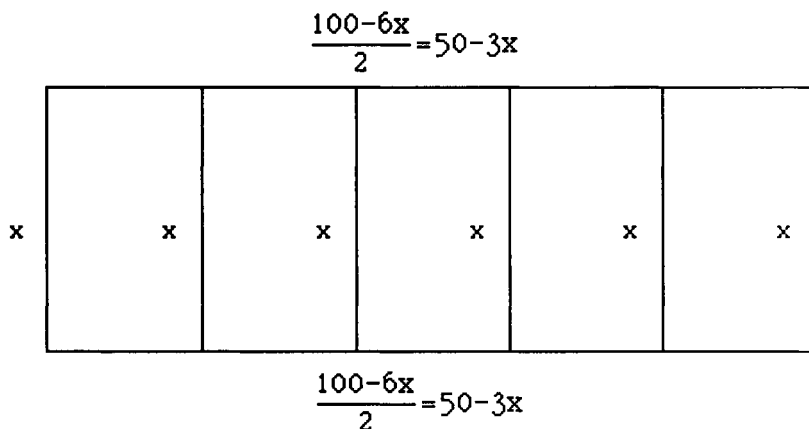
■ **Application: Maxima and Minima**

Mathematica can be used to solve maximization/minimization problems. An example of this type of problem is as follows :

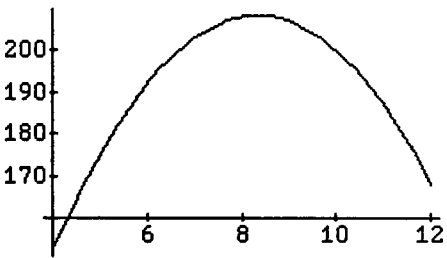
□ **Example:**

A farmer has 100 feet of fencing to construct five dog kennels by first constructing a fence around a rectangular region, and then dividing that region into five smaller regions by placing four fences parallel to one of the sides. What dimensions will maximize the total area?

First, let y denote the length across the top and bottom of the rectangular region and let x denote the vertical length. Then, since 100 feet of fencing are used, a relationship between x and y is given by the equation: $2y + 6x = 100$. Solving this equation for y , we obtain $y = 50 - 3x$ which is shown in the diagram below:



Since the area of a rectangle is $A = x y$, the function to be maximized is defined by entering **area[x]=x(50-3x)**. The value of x which maximizes the area is found by finding the critical value and observing the graph of **area[x]**.

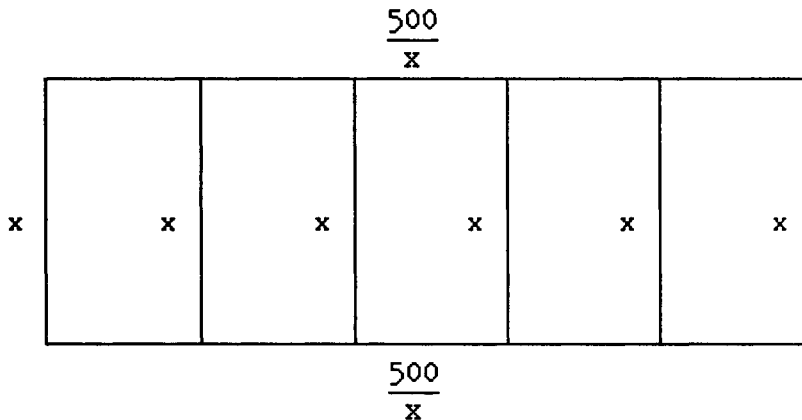
Maxima/Minima	
In[1]:= area[x]=x(50-3x)] <i>Define area as a function of x.</i>
Out[1]= (50 - 3 x) x	
In[2]:= area'[x]] <i>Compute the derivative.</i>
Out[2]= 50 - 6 x	
In[3]:= Solve[area'[x]==0]] <i>Find the values of x for which the derivative is zero.</i>
Out[3]= 25 {x -> --} 3	
In[5]:= Plot[area[x],{x,4,12}]] <i>Verify that the value of x for which the derivative is zero results in maximum area.</i>
	
Out[5]= -Graphics-	

The next problem is slightly different.

□ **Example:**

A farmer wants to construct five dog kennels of total area 500 square feet by first constructing a fence around a rectangular region, and then dividing that region into five smaller regions by placing four fences parallel to one of the sides. What dimensions will minimize the fencing used?

In this case, the total amount of fencing needed to construct the kennels is to be minimized using the constraint that the total area is 500 square feet. (In the first problem, we maximized area using a constraint on the perimeter.) Again, let y = length across the top and bottom of the rectangular region. Using the fact that $\text{area} = 500$, we have $x y = 500$ or $y = 500/x$:



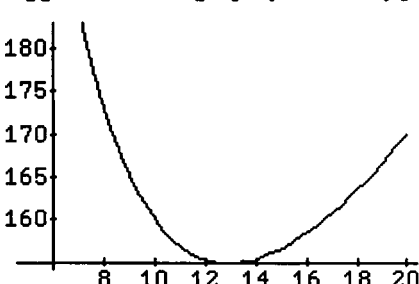
The perimeter of the rectangular region equals $2x + 2y$. Substituting for y , we define the function **perimeter** by entering **perimeter [x_] = 6x + 2(500/x)** which is to be minimized. The steps involved in solving this problem are shown below. (Note that only positive values of x are considered since x represents length.)

Maxima/Minima

```

In[9]:=
  perimeter[x_]=6x+2 500/x
Out[9]=
  1000
  ---- + 6 x
  x
In[10]:=
  perimeter'[x]
Out[10]=
  1000
  6 - ----
      2
      x
In[11]:=
  values=Solve[perimeter'[x]==0]
Out[11]=
  10 Sqrt[5]
  {{x -> -----},
   Sqrt[3]
  -10 Sqrt[5]
  {x -> -----}}
   Sqrt[3]
In[12]:=
  values // N
Out[12]=
  {{x -> 12.9099}, {x -> -12.9099}}
In[15]:=
  Plot[perimeter[x],{x,5,20}]

```



```

Out[15]=
  -Graphics-

```

Define perimeter as a function of x .

Compute the derivative.

Find the values of x for which the derivative is zero.

Notice that the solution to the problem must be positive.

A numerical approximation of the solution is 12.9099.

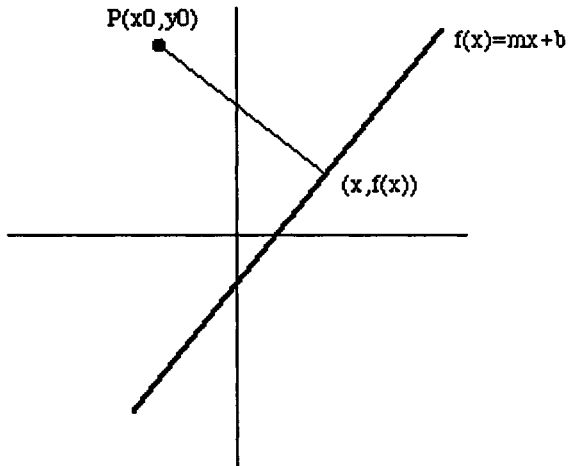
Verify that perimeter is a minimum

when $x = \frac{10\sqrt{5}}{\sqrt{3}} \approx 12.9099$.

Our final example illustrates *Mathematica's* ability to symbolically manipulate algebraic expressions.

□ **Example:**

Let $f(x)=mx+b$ and (x_0,y_0) be any point. Find the value of x for which the distance from (x_0,y_0) to $(x,f(x))$ is a minimum.



The distance from the point (x_0,y_0) to $(x,f(x))$ is given by the distance formula:

$$d = \sqrt{(x_0 - x)^2 + (y_0 - f(x))^2}$$

In order to determine the value of x which minimizes the distance between (x_0,y_0) and $(x, f(x))$, a function which determines this distance must first be defined. This is accomplished by defining the function **distance** by entering `distance[{a_, b_}, {c_, d_}]` which gives the distance between any two points (a,b) and (c,d) . Then the particular distance function for this problem is obtained by substituting the appropriate points (x_0,y_0) and $(x,f(x))$ into **distance**. The value of x that minimizes this function is obtained in the usual manner. (Notice how naming the distance function **expression** simplifies the solution of the problem.)

Maxima/Minima

```

In[38]:=
Clear[distance, f]
distance[ {a_, b_}, {c_, d_} ] =
Sqrt[ (a-c)^2+(b-d)^2 ]
Out[38]=
      2      2
Sqrt[ (a - c) + (b - d) ]
In[39]:=
f[ x_ ] = m x + b
Out[39]=
b + m x
In[40]:=
expression = distance[ {x0, y0}, {x, f[ x] } ]
Out[40]=
      2
Sqrt[ (-x + x0) +
      2
      (- (b + m x) + y0) ]
In[41]:=
equation = D[ expression, x ]
Out[41]=
(-2 (-x + x0) -
      2 m (- (b + m x) + y0) ) /
      2 Sqrt[ (-x + x0) +
      2
      (- (b + m x) + y0) ]
In[42]:=
Solve[ equation == 0, x ]
Out[42]=
      b m - x0 - m y0
      ( { x -> - (-----) } )
      2
      1 + m

```

distance[{a, b}, {c, d}] is a function that computes the distance between two points (a,b) and (c,d).

A non-vertical line can always be written in the form $f(x) = mx + b$ where m is the slope and the point $(0, b)$ is the y-intercept.

$f[x]$ is the line $m x + b$.

To denote "m times x" a space must be included between m and x.

expression is the distance between the points (x_0, y_0) and $(x, f(x))$.

Since expression contains several unknowns, the "x" must be included to indicate that the differentiation is to be calculated with respect to x.

equation is the derivative of expression with respect to x.

Since equation has more than one variable, the "x" must be included to indicate that the desired solution is for x.

Solve[equation == 0, x] finds the values of x (with respect to b, m, x0, and y0) for which equation is zero.

■ Application

As was previously stated, *Mathematica* can define many different types of functions. The following example is a function `plotderiv` which is a function that depends on a function and an interval. When given a function, $f(x)$, and an interval, $\{a,b\}$, `plotderiv` simultaneously plots $f(x)$ with `GrayLevel[0]` and $f'(x)$ with `GrayLevel[.3]`. (Remember, `GrayLevel[0]` indicates the darker curve.) Note that the labels f , f , f , and g' on the plots below were added later.

```

In[15]:=
plotderiv[f_,{a_,b_}]:=
Plot[{f[x],f'[x]},{x,a,b},
PlotStyle->
{GrayLevel[0],GrayLevel[.3]}]

In[17]:=
Clear[f,g]
f[x_]:=x^2
g[x_]:=x^3

In[18]:=
plotderiv[f,{-1,1}]

Out[18]=
-Graphics-

In[19]:=
plotderiv[g,{-1,1}]

Out[19]=
-Graphics-

```

The first plot shows the function $f(x) = x^2$ (solid curve) and its derivative $f'(x) = 2x$ (dashed line) on the interval $[-1, 1]$. The x-axis ranges from -1 to 1, and the y-axis ranges from -1 to 2. The second plot shows the function $g(x) = x^3$ (solid curve) and its derivative $g'(x) = 3x^2$ (dashed curve) on the interval $[-1, 1]$. The x-axis ranges from -1 to 1, and the y-axis ranges from -0.4 to 0.8.

`plotderiv` is defined to be a function that graphs a given function f and its derivative f' on an interval $[a,b]$.

To illustrate the function we use two elementary examples:

$$f(x) = x^2 \text{ and } g(x) = x^3.$$

`plotderiv[f,{-1,1}]` displays the graph of f and f' on the interval $[-1,1]$.

`plotderiv[g,{-1,1}]` displays the graph of g and g' on the interval $[-1,1]$.

■ 3.3 Implicit Differentiation

■ Computing Derivatives of Implicit Functions

If **equation** is an equation with variables **x** and **y**, *Mathematica* computes the implicit derivative of **equation** with the command **Dt [equation, x]** where **equation** is differentiated with respect to the variable **x**.

The expression **Dt [y, x]** encountered when using implicit differentiation represents the derivative of **y** with respect to **x**, dy/dx . (Hence, **Dt [x, y]** represents dx/dy .)

The built-in command **Dt** is versatile. Although here **Dt** is used to perform implicit differentiation,

Dt[expression, variable] computes the total derivative: $\frac{d(\text{expression})}{d \text{variable}}$; and

Dt[expression] computes the total differential $d(\text{expression})$.

The following examples demonstrate the use of the implicit differentiation command, **Dt [equation, x]** and show how this command can be used with **Solve** to obtain the desired derivative in a single command.

□ Example:

Find $\frac{dy}{dx}$ for: (A) $x^3 + y^3 = 1$; (B) $\cos(x^2 - y^2) = y \cos(x)$; and (C) $3y^4 + 4x - x^2 \sin(y) - 4 = 0$.

ImplicitDifferentiation

In[41]:= `Dt[x^3+y^3==1,x]` Dt[y,x] represents $\frac{dy}{dx}$.

Out[41]=

$$3x^2 + 3y^2 \text{Dt}[y,x] == 0$$

In[42]:= `Solve[Dt[x^3+y^3==1,x],Dt[y,x]]` Solve[Dt[x^3+y^3==1,x],Dt[y,x]] computes $\frac{dy}{dx}$ for $x^3 + y^3 = 1$.

Out[42]=

$$\left\{ \left\{ \text{Dt}[y,x] \rightarrow -\left(\frac{x^2}{y^2} \right) \right\} \right\}$$

In[43]:= `Solve[Dt[Cos[x^2-y^2]==y Cos[x],x],Dt[y,x]]`

Out[43]=

$$\left\{ \left\{ \text{Dt}[y,x] \rightarrow -\left(\frac{y \sin[x] - 2x \sin[x - y^2]}{-\cos[x] + 2y \sin[x - y^2]} \right) \right\} \right\}$$

computes $\frac{dy}{dx}$ for $\cos(x^2 - y^2) = y \cos(x)$.

In[44]:= `Solve[Dt[3y^4+4x-x^2 Sin[y]-4==0,x],Dt[y,x]]`

Out[44]=

$$\left\{ \left\{ \text{Dt}[y,x] \rightarrow -\left(\frac{4 - 2x \sin[y]}{12y^3 - x^2 \cos[y]} \right) \right\} \right\}$$

computes $\frac{dy}{dx}$ for $3y^4 + 4x - x^2 \sin(y) - 4 = 0$.

■ Other Methods to Compute Derivatives of Implicit Functions

The same results as above can be obtained if y is declared to be a function of x . Hence instead of entering

the equation $x^3 + y^3 = 1$ as $x^3 + y^3 = 1$, enter it as $x^3 + \underline{y[x]}^3 = 1$.

y is a
function
of x .

OtherImplicit

In[11]:= $D[x^3 + y[x]^3 == 1, x]$

Out[11]= $3x^2 + 3y[x]^2 y'[x] == 0$

In[12]:= $Solve[D[x^3 + y[x]^3 == 1, x], Y'[x]]$

Out[12]= $\{(Y'[x] \rightarrow -\frac{x}{y[x]^2})\}$

In[13]:= $Solve[D[Cos[x^2 - y[x]^2] == y[x]Cos[x], x], Y'[x]]$

Out[13]= $\{(Y'[x] \rightarrow -\frac{-2x \sin[x^2 - y[x]^2] + \sin[x] y[x]}{-\cos[x] + 2 \sin[x^2 - y[x]^2] y[x]})\}$

$D[x^3 + y[x]^3 == 1, x]$ computes the derivative of the equation $x^3 + (y(x))^3 = 1$ with respect to x . We assume y is a function of x .

solves the equation resulting from the command $D[x^3 + y[x]^3 == 1, x]$ for $y'(x)$.

solves the equation resulting from the command $D[Cos[x^2 - y[x]^2] == y[x]Cos[x], x]$ for $y'(x)$.

● Graphing Implicit Functions with Version 2.0

The Version 2.0 package `ImplicitPlot.m` contains the command `ImplicitPlot` which can be used to graph some equations; `ImplicitPlot.m` is not included in earlier versions of *Mathematica*.

`ImplicitPlot` is discussed in more detail in Chapter 9.

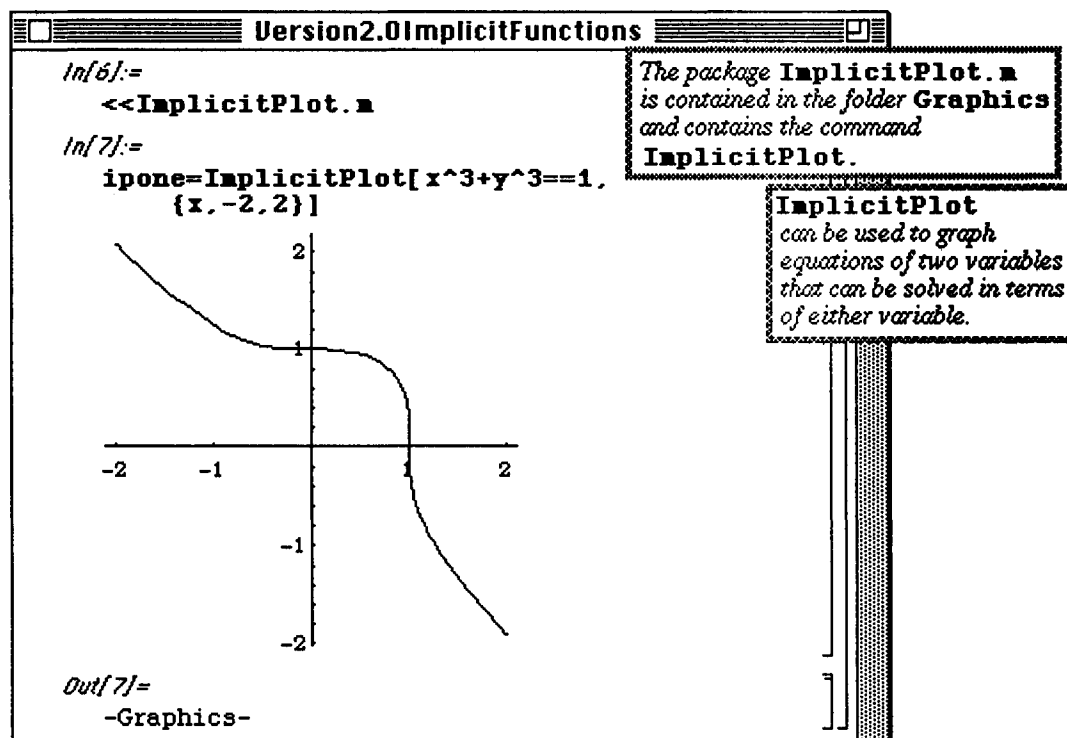
The most basic form of the syntax for the command `ImplicitPlot` is

`ImplicitPlot[equation, {x, xmin, xmax}]`. The set of y-values displayed can also be specified by entering the command in the form

`ImplicitPlot[equation, {x, xmin, xmax}, {y, ymin, ymax}]`. Be sure to always include the double-equals sign between the right- and left-hand side of equations.

○ Example:

Use `ImplicitPlot` to graph the equation $x^3 + y^3 = 1$.



The command `ImplicitPlot` works best with equations that are (easily) solvable. Notice that

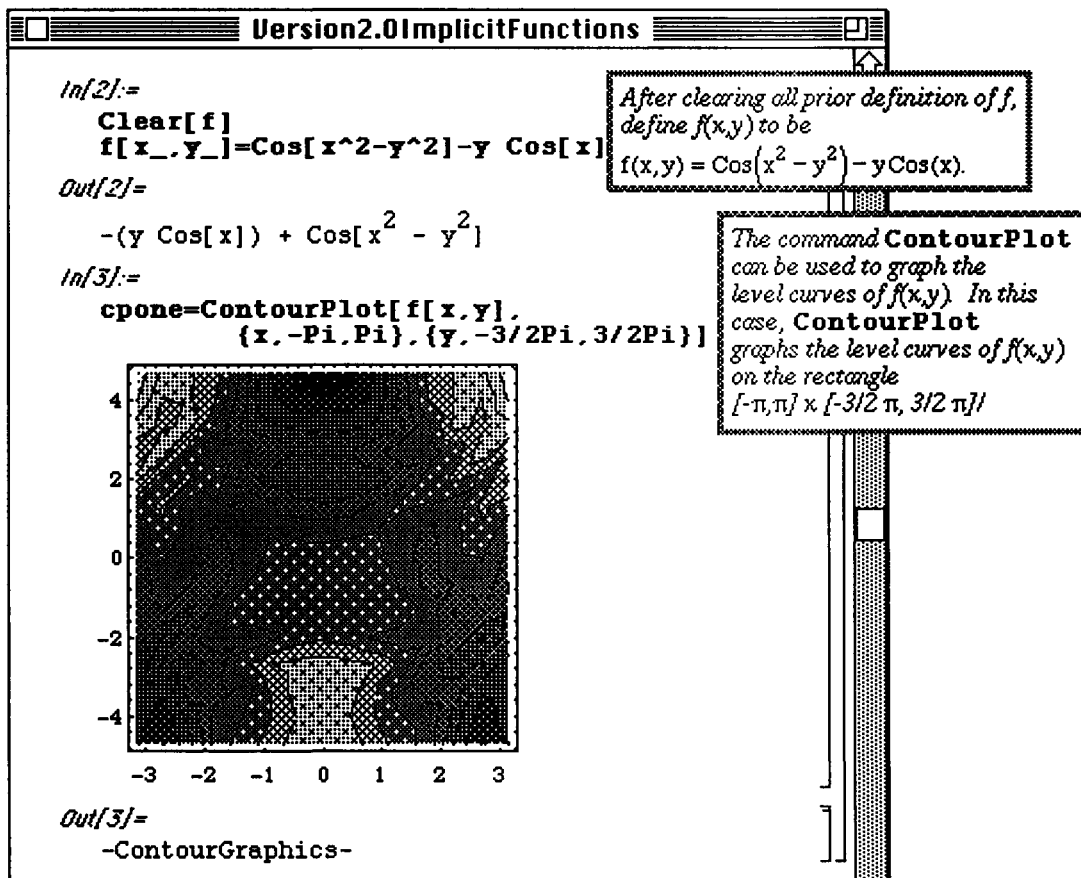
`ImplicitPlot` cannot be used to graph the equation $\text{Cos}(x^2 - y^2) = y \text{Cos}(x)$:

<pre> In[8]:= iptwo=ImplicitPlot[Cos[x^2-y^2]==y Cos[x], {x,-Pi,Pi}] Solve::ifun: Warning: Inverse functions are being used by Solve, so some solutions may not be found. Solve::ifun: Warning: Inverse functions are being used by Solve, so some solutions may not be found. Solve::tdep: The equations appear to involve transcendental functions of the variables in an essentially non-algebraic way. ReplaceAll::reps: {Solve[1. y + <<1>> == 0, y]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing. Solve::ifun: Warning: Inverse functions are being used by Solve, so some solutions may not be found. General::stop: Further output of Solve::ifun will be suppressed during this calculation. </pre>	<p><i>Since Mathematica is unable to solve the equation</i></p> $\text{Cos}(x^2 - y^2) = y \text{Cos}(x)$ <p><i>for either x or y, the command</i> ImplicitPlot <i>cannot be used to generate a graph of this equation.</i></p>
---	--

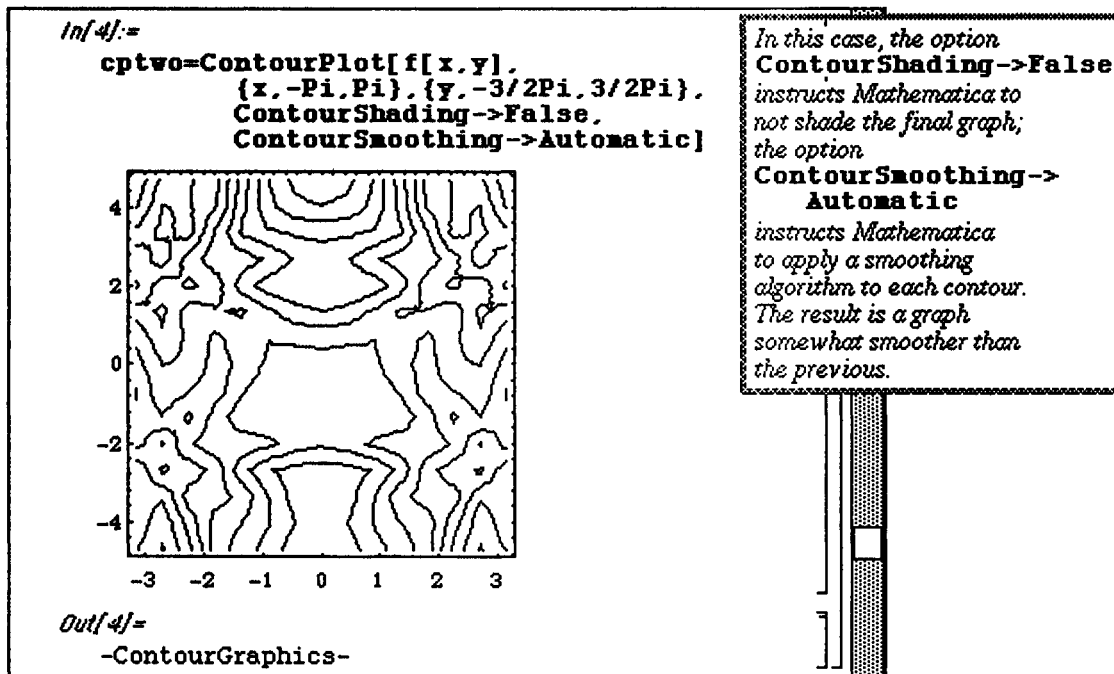
Instead a different approach is used taking advantage of the built-in function `ContourPlot`. The contour graphs shown here were created with the `ContourPlot` command as it is in Version 2.0. The Version 2.0 `ContourPlot` command is substantially different from earlier versions of `ContourPlot` which are discussed later.

To use `ContourPlot` to graph the equation $\cos(x^2 - y^2) = y \cos(x)$, we begin by noticing that graphing the equation $\cos(x^2 - y^2) = y \cos(x)$ is equivalent to defining $f(x,y) = \cos(x^2 - y^2) - y \cos(x)$ and graphing $f(x,y) = 0$.

`ContourPlot[f[x,y], {x,xmin,xmax}, {y,ymin,ymax}]` graphs a set of level curves of the function $f[x,y]$ on the rectangle $[xmin,max] \times [ymin,ymax]$:



ContourPlot has many available options. For example, *Mathematica* can apply a smoothing algorithm to each contour which results in a smoother graph with the option **ContourSmoothing**→**Automatic**; the option **ContourShading**→**False** specifies that *Mathematica* not shade the resulting graph:

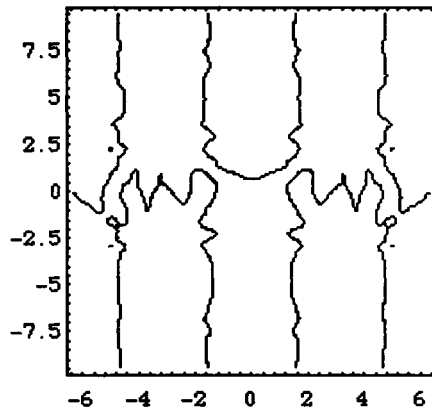


In addition, the actual contour values or the number of contours to be graphed can be specified. The option **Contours**→**n** specifies that *Mathematica* draw **n** evenly-spaced level curves (the default value is 10). The option **Contours**→{**val1**, **val2**, ..., **valn**} specifies that *Mathematica* graph level curves corresponding to **val1**, **val2**, ..., **valn**.

Since the graph of $f(x,y)=0$ corresponds to a level curve of $f(x,y)$ for the value 0, the desired graph is obtained as follows:

In[5]:=

```
cpthree=ContourPlot[
  f[x,y],
  {x,-2Pi,2Pi},{y,-3Pi,3Pi},
  Contours->{0},
  PlotPoints->30,
  ContourShading->False,
  ContourSmoothing->Automatic]
```



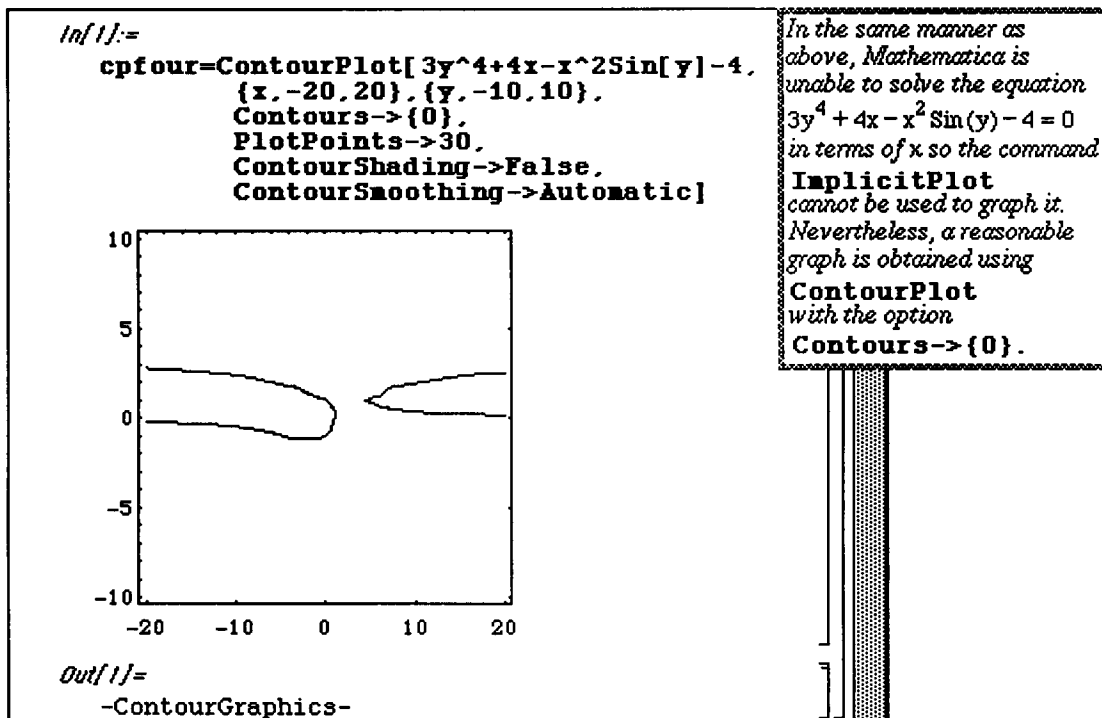
Out[5]=

```
-ContourGraphics-
```

The option **Contours->{0}** specifies that Mathematica attempt to graph the contour corresponding to the equation $f(x,y)=0$.

The option **PlotPoints->30** specifies that Mathematica use 30 sample points (the default is 15 sample points).

In the same manner as above, the graph of the equation $3y^4 + 4x - x^2 \sin(y) - 4 = 0$ corresponds to a contour graph of $3y^4 + 4x - x^2 \sin(y) - 4$ for the contour with value 0:



■ 3.4 Integral Calculus

■ Computing Definite and Indefinite Integrals

- In order to compute definite integrals, Version 1.2 (or earlier) of *Mathematica* must load the package **IntegralTables.m**. The package **IntegralTables.m** is contained in the folder **StartUp** which is contained in the folder **Packages** in the *Mathematica* folder. The easiest way to load the package **IntegralTables.m** is to **Enter** the command `<<IntegralTables.m`. If you are using Version 2.0, this procedure is not applicable since Version 2.0 automatically loads the package **IntegralTables.m**.
- Version 2.0 automatically loads the package **IntegralTables.m**; hence, the above procedure is **not** pertinent to Version 2.0 users.
- **Note Regarding Frequent Computations of Definite Integrals**

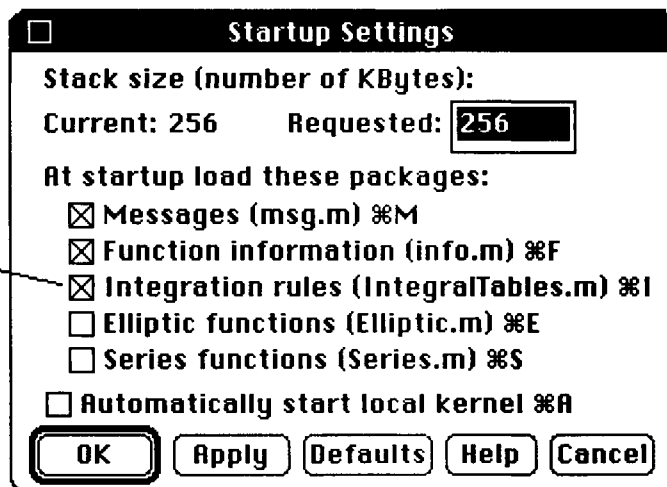
This note is only applicable if you are NOT using Version 2.0.

If you are going to be computing definite integrals frequently you will want to have *Mathematica* automatically load up **IntegralTables.m** when the *Mathematica* kernel is started. To do this, proceed as follows.

- 1) Go to **Edit** and select **Settings**;
- 2) Select **StartUp**;
- 3)

Click inside the box beside Integration Rules. Boxes with x's in them indicate which packages are automatically loaded when the kernel is started.

Click in this box



Each of the following examples illustrate typical commands used to compute indefinite integrals.

The *Mathematica* command to compute $\int f(x)dx$ is `Integrate [f[x],x]`.

The command `Integrate [expression, variable]` instructs *Mathematica* to integrate **expression** with respect to **variable**.

□ Example:

Use *Mathematica* to compute $\int x^2(1-x^3)^4 dx$, $\int e^x \cos(x) dx$, $\int \frac{\ln(x)}{x^3} dx$,

and $\int \sin(2x)\cos^2(x) dx$.

Integration	
<pre>In[1]:= <<IntegralTables.m Out[1]= Integrator` In[6]:= Integrate[x^2 (1-x^3)^4, x] Out[6]= 3 5 -(1 - x) ----- 15 In[7]:= Integrate[Exp[x] Cos[x], x] Out[7]= x x E Cos[x] E Sin[x] ----- + ----- 2 2</pre>	<p><i>Be sure to load IntegralTables.m before attempting to compute definite or indefinite integrals.</i></p> <p>Integrate[x^2 (1-x^3)^4, x] <i>computes the indefinite integral</i> $\int x^2(1-x^3)^4 dx$.</p> <p>Integrate[Exp[x] Cos[x], x] <i>computes the indefinite integral</i> $\int e^x \cos(x) dx$.</p>

To compute $\int \frac{\text{Ln}(x)}{x^3} dx$, remember that the *Mathematica* function `Log [x]` denotes the natural logarithm function $\text{Ln}(x)$:

<pre>In[8]:= Integrate[Log[x]/x^3,x]</pre>	<pre>Integrate[Log[x]/x^3,x] computes the indefinite integral ∫ Ln(x)/x^3 dx.</pre>
<pre>Out[8]= -1 Log[x] ---- 2 2 4 x 2 x</pre>	
<pre>In[9]:= Integrate[Sin[2 x](Cos[x])^2,x]</pre>	<pre>Integrate[Sin[2 x](Cos[x])^2,x] computes the indefinite integral ∫ Sin(2x)Cos^2(x)dx.</pre>
<pre>Out[9]= -Cos[2 x] Cos[4 x] ---- 4 16</pre>	

Mathematica computes the definite integral $\int_a^b f(x)dx$ with the command `Integrate [f[x],{x,a,b}]`.

In general, the command

`Integrate[expression,{variable,lower limit,upper limit}]`
 integrates **expression** with respect to **variable** and evaluates from **lower limit** to **upper limit**.

□ Example:

Use *Mathematica* to compute $\int_0^\pi \sin(x) dx$, $\int_0^2 \sqrt{4-x^2} dx$, $\int_1^2 x^3 e^{4x} dx$, and $\int_{-\pi}^{2\pi} e^{2x} \sin^2(2x) dx$.

Integration	
<pre>In[2]:= Integrate[Sin[x], {x, 0, Pi}] Out[2]= 2</pre>	<pre>Integrate[Sin[x], {x, 0, Pi}] computes the definite integral ∫₀^π Sin(x) dx.</pre>
<pre>In[4]:= Integrate[Sqrt[4-x^2], {x, 0, 2}] Out[4]= Pi</pre>	<pre>Integrate[Sqrt[4-x^2], {x, 0, 2}] computes the definite integral ∫₀² √(4-x²) dx.</pre>
<pre>In[5]:= Integrate[Exp[4x] x^3, {x, 1, 2}] Out[5]= 4 8 -17 E + 181 E ----- 128 128</pre>	<pre>Integrate[Exp[4x] x^3, {x, 1, 2}] computes the definite integral ∫₁² x³ e⁴ˣ dx.</pre>
<pre>In[6]:= Integrate[Exp[2x] (Sin[2x])^2, {x, -Pi, 2 Pi}] Out[6]= 4 Pi -1 E ----- + ----- 2 Pi 5 5 E</pre>	<pre>Integrate[Exp[2x] (Sin[2x])^2, {x, -Pi, 2 Pi}] computes the definite integral ∫₋π²π e²ˣ Sin²(2x) dx.</pre>

When the command `Integrate[f[x], {x, xmin, xmax}]` is entered, *Mathematica* computes an anti-derivative F of f and computes $F[x_{\max}] - F[x_{\min}]$. Nevertheless, *Mathematica* does not apply the Fundamental Theorem of Calculus since *Mathematica* does not verify that f is continuous on the interval $[x_{\min}, x_{\max}]$. In cases when f is not continuous on $[x_{\min}, x_{\max}]$, errors often occur:

<pre>In[3]:= Integrate[1/x, {x, -1, 1}] Out[3]= -Log[-1]</pre>	<pre>Mathematica computes Integrate[1/x, {x, -1, 1}] even though the Fundamental Theorem of Calculus does not apply to the integral ∫₋₁¹ 1/x dx.</pre>
--	--

■ Numerically Computing Definite Integrals

Mathematica can also numerically integrate definite integrals of the form

$$\int_{\text{lowerlimit}}^{\text{upperlimit}} \text{expression} d\text{variable}$$

with the command

NIntegrate[**expression**, {**variable**, **lowerlimit**, **upperlimit**}] . **NIntegrate** is a built-in command and is NOT contained in the package **IntegralTables.m**. Consequently, Version 1.2 users should be aware that it is **not** necessary to load the package **IntegralTables.m** to use the command **NIntegrate**.

The command **NIntegrate** is useful when an anti-derivative of **expression** cannot be (easily) found and **expression** is fairly smooth on the interval [**lower limit**, **upper limit**]. Also, in those cases in which an anti-derivative can be determined, the value of the definite integral can usually be computed more quickly by an approximation with **NIntegrate** rather than **Integrate**.

- In Version 2.0, the package **GaussianQuadrature.m** contained in the **Numerical Math** folder can also be used to numerically compute integrals. The package **GaussianQuadrature.m** is discussed in Chapter 9.

Integrate[**f[x]**, {**x**, **a**, **b**}] applies the Fundamental Theorem of Calculus, if applicable: it finds an anti-derivative of **f[x]**, evaluates the anti-derivative at the upper limit of integration, and subtracts the value of the anti-derivative evaluated at the lower limit of integration. As noted above, if **f** is not continuous on [**a**,**b**], error often occurs.

□ Examples:

(A) Approximate $\int_0^{\pi^{1/3}} e^{-x^2} \cos(x^3) dx$; and

(B) Compute both exact and approximate values of $\int_0^{\pi/8} e^{3x} \cos(4x) dx$.

The screenshot shows a Mathematica window titled "Integration". It contains the following input and output:

```

In[7]:=
Integrate[Exp[-x^2]Cos[x^3],
{x, 0, Pi^(1/3)}]
Out[7]=
      3
      Cos[x ]
Integrate[-----, {x, 0, Pi  }]
            2
            x
            E

In[8]:=
NIntegrate[Exp[-x^2]Cos[x^3],
{x, 0, Pi^(1/3)}]
Out[8]=
0.701566

In[9]:=
Integrate[Exp[3x]Cos[4x],
{x, 0, Pi/8}]
Out[9]=
      3      4 E
      -(---) + -----
      25      25
      (3 Pi)/8
  
```

At the bottom of the window, it displays "Time: 2.40 seconds".

Mathematica cannot evaluate the definite integral

$$\int_0^{\pi^{1/3}} e^{-x^2} \cos(x^3) dx.$$

However, Mathematica can approximate the integral using the command **NIntegrate**.

It takes Mathematica 2.4 seconds to compute the exact value of the definite integral

$$\int_0^{\pi/8} e^{3x} \cos(4x) dx.$$

The screenshot shows a Mathematica window titled "Integration". It contains the following input and output:

```

In[10]:=
NIntegrate[Exp[3x]Cos[4x],
{x, 0, Pi/8}]
Out[10]=
0.39971

Time: 1.03 seconds
  
```

However, the definite integral can be approximated in only 1.03 seconds.

• Definite Integration with Version 2.0

The point to be made concerning integration in Version 2.0 is that `IntegralTables.m` does not have to be loaded before evaluating definite integrals. Several examples are given below. In each case, the same results are obtained with Version 1.2:

The screenshot shows a Mathematica notebook window titled "Version2Integrals". The notebook contains the following input and output cells:

```

In[1]:=
Integrate[Sqrt[1-1/3 (Sin[x])^2], x]
Out[1]=
      2
      Sin[x]
Integrate[Sqrt[1 - ----], x]
                3

Integrate[Cos[2x]/(x^2+1), {x, -Infinity, Infinity}]
Pi
-----
  2
E

Integrate[Exp[I x]/x, {x, -Infinity, Infinity}]
I Pi

Integrate[1/(3+2 Cos[x]), {x, 0, Pi}]
  Pi
-----
Sqrt[5]

Integrate[1/(3+2 Cos[x])^2, {x, 0, Pi}]
  3 Pi
-----
  3/2
  5

Integrate[1/x, {x, 0, 1}]
Infinity

Integrate[1/x, {x, -1, 1}]
-I Pi
  
```

Callout boxes provide additional information:

- For the first integral: *Mathematica cannot compute*

$$\int \sqrt{1 - \frac{1}{3} \sin^2(x)} dx.$$
- For the second integral: *computes the principal value of the integral*

$$\int_{-\infty}^{\infty} \frac{\cos(2x)}{x^2+1} dx.$$
- For the third integral: *computes the principal value of the integral*

$$\int_{-\infty}^{\infty} \frac{e^{ix}}{x} dx.$$
- For the fourth integral: *computes the value of*

$$\int_0^{\pi} \frac{1}{3+2\cos(x)} dx$$
- For the fifth integral: *computes the value of*

$$\int_0^{\pi} \frac{1}{(3+2\cos(x))^2} dx.$$
- For the sixth integral: $\int_0^1 \frac{1}{x} dx$ is infinity
- For the seventh integral: *The principal value of* $\int_{-1}^1 \frac{1}{x} dx$ is $-i\pi$

■ Application: Area Between Curves

A type of problem which incorporates the commands **Integrate** and **NIntegrate** is that of finding the area between curves. These problems also use several other *Mathematica* commands (**Plot**, **NRoots**, **FindRoot**, ...) which were introduced earlier in the text.

□ Example:

Let $p(x) = \frac{3}{10}x^5 - 3x^4 + 11x^3 - 18x^2 + 12x + 1$ and $q(x) = -4x^3 + 28x^2 - 56x + 32$

Approximate the area of the region bounded by the graphs of p and q .

Mathematica is quite helpful in problems of this type. We can observe the region whose area we are seeking using the **Plot** command, and we can locate the points of intersection with one of the commands used in solving equations (**NRoots**, **FindRoot**, **Solve**, or **NSolve** (Version 2.0 only)) These steps are carried out below:

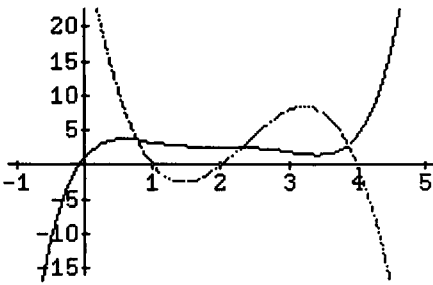
Area
□

```

In[1]:=
<<IntegralTables.m
Clear[p,q]
p[x_]=3x^5 /10-3x^4+11x^3-18x^2+12x+1
q[x_]=-4x^3+28x^2-56x+32
Plot[{p[x],q[x]},{x,-1,5},
PlotStyle->{GrayLevel[0],GrayLevel[.3]}]

```

Be sure the package IntegralTables.m has been loaded before computing definite integrals.



The x-coordinates of the intersection points are the solutions to the polynomial equation $p(x)=q(x)$.

```

Out[1]=
-Graphics-
In[2]:=
NRoots[p[x]==q[x],x]

```

*Mathematica is unable to exactly solve the polynomial equation $p(x)=q(x)$. Therefore, use the command **NRoots**[p[x]==q[x],x] to approximate the solutions to the equation.*

```

Out[2]=
x == 0.772058 ||
x == 1.5355 - 3.57094 I ||
x == 1.5355 + 3.57094 I ||
x == 2.29182 || x == 3.86513

```

Using the roots to the equation $p(x) = q(x)$ found above, the graph clearly shows that $p(x) > q(x)$ between $x=0.772058$ and $x=2.29182$; and $q(x) > p(x)$ between $x=2.29182$ and $x=3.86513$.

Hence, an approximation of the area bounded by $p(x)$ and $q(x)$ given by the integral

$\int_{0.772058}^{2.29182} (p(x) - q(x)) dx + \int_{2.29182}^{3.86513} (q(x) - p(x)) dx$ is computed with either of the following commands

```

In[3]:=
  Integrate[(p[x]-q[x]),{x,.772058,2.29182}]+
  Integrate[(q[x]-p[x]),{x,2.29182,3.86513}]

Out[3]=
  12.1951

In[4]:=
  NIntegrate[(p[x]-q[x]),{x,.772058,2.29182}]+
  NIntegrate[(q[x]-p[x]),{x,2.29182,3.86513}]

Out[4]=
  12.1951

```

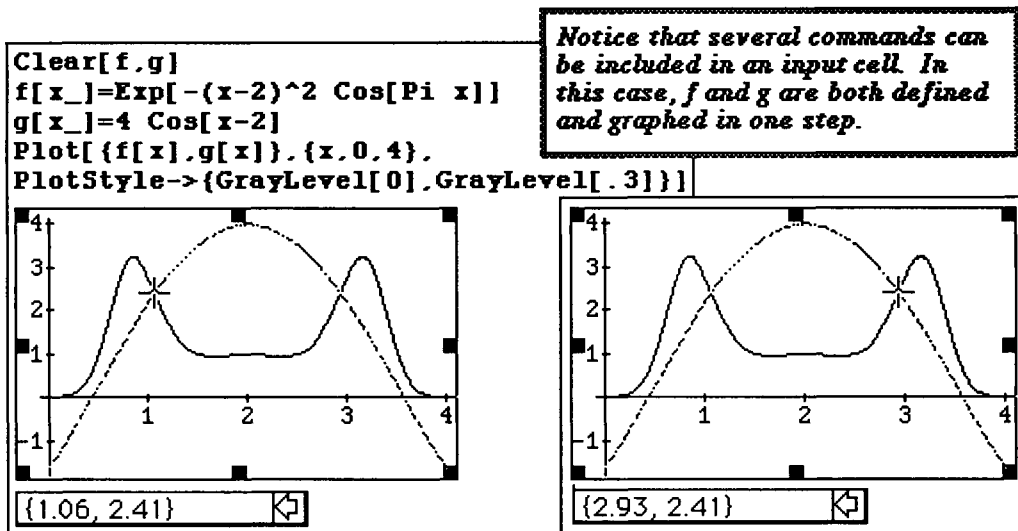
In this case, both
Integrate
and
NIntegrate
yield the same
approximations
of the area bounded
by the graphs.

Next, consider a problem which involves functions which are not polynomials.

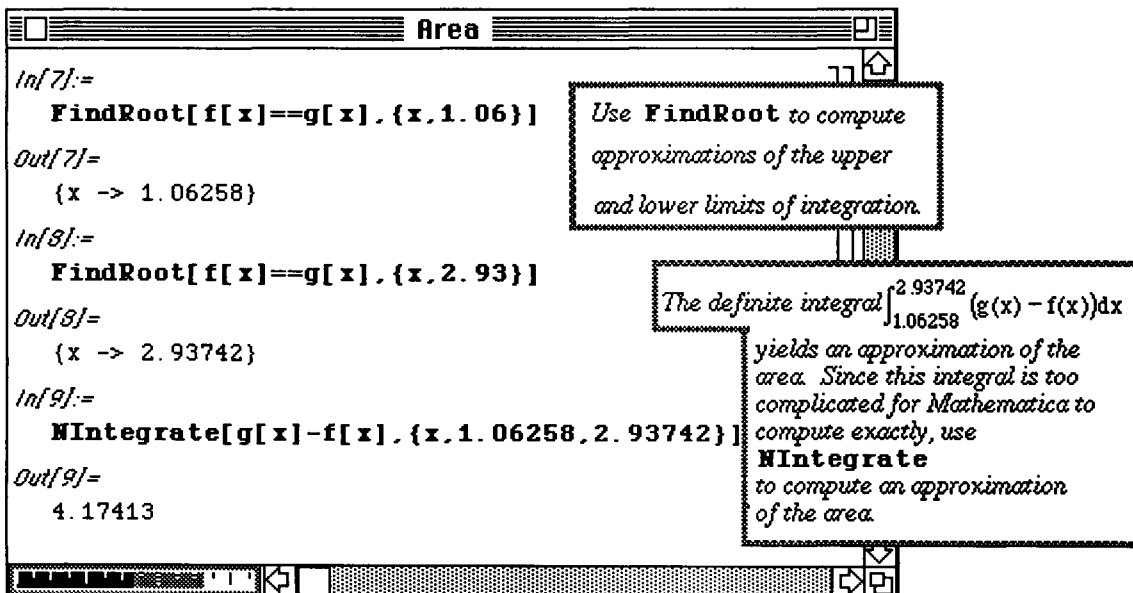
□ Example:

Let $f(x) = e^{-(x-2)^2 \cos[\pi x]}$ and $g(x) = 4 \cos(x-2)$ on the interval $[0, 4]$.

Approximate the area of the region bounded by the graphs of f and g . Since these functions are not polynomials, **FindRoot** must be used to determine the points of intersection. Recall that **FindRoot** depends on an initial guess of the root. Therefore, the first step towards solving this problem is to graph the functions f and g . Then the cursor is used to locate the initial guesses. (This topic was discussed earlier in the text.)



Once the initial guesses have been determined with the cursor, **FindRoot** is used to approximate the solutions to the equation $f(x) = g(x)$, and the area is approximated with **NIntegrate**.



■ Application: Arc Length

□ Example:

Approximate the arc length of the graph of $f(x) = \sin(\pi \sin(x-2)^2)$ on the interval $[4, 5]$.

Recall the formula for the length of the smooth curve $g(x)$ from the point $(a, g(a))$ to $(b, g(b))$

$$\text{is given by: Length} = \int_a^b \sqrt{1 + (g'(x))^2} \, dx.$$

The resulting definite integrals used for determining arc length are usually difficult to compute since they involve a radical. Since the built-in command `NIntegrate[f[x], {x, a, b}]` numerically approximates integrals, *Mathematica* is very helpful with approximating solutions to these types of problems!

First define and graph f.

$f'[x]$ computes $f'(x)$.

The length of the graph of a smooth function f on an interval $[a, b]$ is given by

$$\int_a^b \sqrt{1 + (f'(x))^2}$$

The arc length is given by $\int_4^5 \sqrt{1 + (f'(x))^2}$

approximates the definite integral

$$\int_4^5 \sqrt{1 + (f'(x))^2}.$$

■ **Application:** Volumes of Solids of Revolution

□ **Example:**

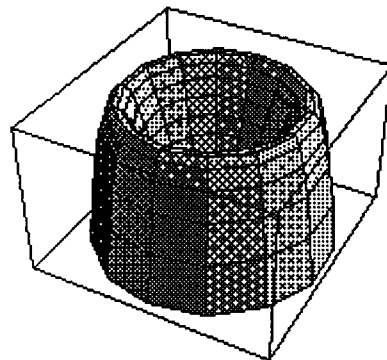
Find the volume of the solid generated by revolving the region bounded by the graphs of

$$g(x) = x^2 \sin(x), \quad x = 0, \quad x = \pi, \quad \text{and} \quad y = 0 \quad \text{about the } y\text{-axis.}$$

Before solving the problem, we remark that solids generated by revolving the graph of a function about the x - or y -axis can be visualized with *Mathematica*. The commands used to generate the following graphics are discussed in the **Appendix**.

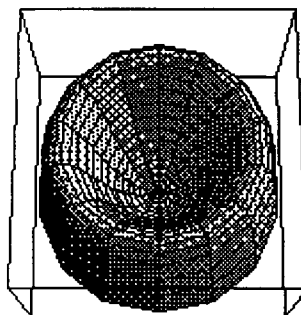
Side view of the solid generated by revolving the region bounded by the graphs of

$$g(x) = x^2 \sin(x), \quad x = 0, \quad x = \pi, \quad \text{and} \quad y = 0 \quad \text{about the } y\text{-axis.}$$



Top view of the solid generated by revolving the region bounded by the graphs of

$$g(x) = x^2 \sin(x), \quad x = 0, \quad x = \pi, \quad \text{and} \quad y = 0 \quad \text{about the } y\text{-axis.}$$



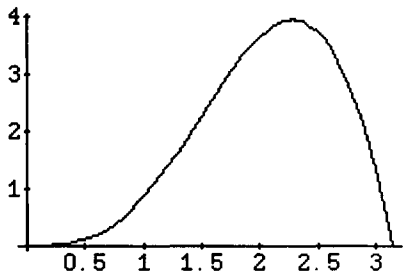
The method of cylindrical shells is used to compute the volume of this solid.

SolidsofRevolution

```

In[1]:=
<<IntegralTables.m
Clear[g]
g[x_]=x^2 Sin[x]
Plot[g[x],{x,0,Pi}]

```



```

Out[1]=
-Graphics-

```

```

In[2]:=
Integrate[2 Pi x g[x],{x,0,Pi}]

```

```

Out[2]=
      2      4
-12 Pi  + 2 Pi

```

Be sure the package `IntegralTables.m` has been loaded before attempting to compute definite integrals.

First define and graph g .

The volume of the solid generated by revolving the region bounded by the graph of g , $x=0$, $x=\pi$, and $y=0$ about the y -axis is given by the definite integral

$$\int_0^\pi 2\pi x g(x) dx = 2\pi \int_0^\pi x^3 \sin(x) dx.$$

calculates $\int_0^\pi 2\pi x g(x) dx = 2\pi \int_0^\pi x^3 \sin(x) dx$

Consequently, the volume of the solid is $2\pi^4 - 12\pi^2$.

□ Example:

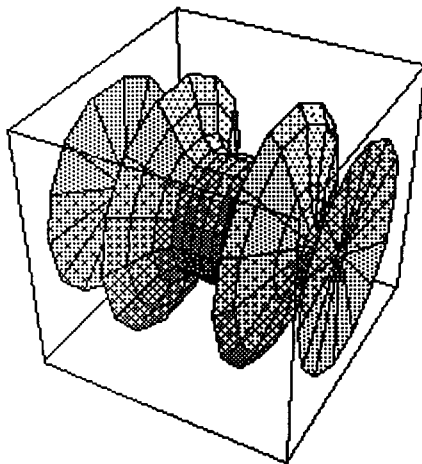
Use *Mathematica* to approximate the volume of the solid generated by revolving the region bounded by the graphs of

$$m(x) = e^{-(x-3)^2 \cos[4(x-3)]}, \quad x=1, \quad x=5, \quad \text{and} \quad y=0$$

The solid generated by revolving the region bounded by the graphs of

$$m(x) = e^{-(x-3)^2 \cos[4(x-3)]}, \quad x=1, \quad x=5, \quad \text{and} \quad y=0$$

about the x -axis can be visualized using *Mathematica*.



The disk method is used to determine the volume of this spool-shaped solid.

☐
SolidsofRevolution
☐ ☰

In[3]:=

```
Clear[m]
m[x_]=Exp[-(x-3)^2 Cos[4(x-3)]]
Plot[m[x], {x,1,5}]
```

Out[3]=

-Graphics-

In[4]:=

```
NIntegrate[Pi (m[x])^2, {x,1,5}]
```

Out[4]=

16.0762

First define and graph m.

The volume of the solid generated by revolving the graph of m, x=1, x=5, and y=0 about the x-axis is given by

$$\int_1^5 \pi (m(x))^2 dx = \pi \int_1^5 \left(e^{-(x-3)^2 \cos[4(x-3)]} \right)^2 dx$$

$$= \pi \int_1^5 e^{-2(x-3)^2 \cos[4(x-3)]} dx.$$

*Since Mathematica cannot compute an exact value of this integral, the command **NIntegrate** is used to approximate its value.*

computes an approximation of the definite integral $\int_1^5 \pi (m(x))^2 dx$.

Therefore the volume of the solid is approximately 16.0762.

☐ ☰
☐ ☰

■ 3.5 Series

■ Computing Power Series

Recall that a power series expansion of a function $f(x)$ about the point $x=a$ is given by the following expression:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!} (x-a)^k. \quad \textit{Mathematica} \text{ computes the power series expansion of a function } f(x) \text{ about}$$

the point $x = a$ up to order n with the command: `Series[f(x), {x, a, n}]`.

Several familiar power series are computed below using this command.

□ Example:

Compute the first few terms of the power series expansion of $f(x)$ about the point $x=a$ for

- (A) $f(x) = e^x$, $a = 0$; (B) $f(x) = \sin(x)$, $a = \pi$; (C) $f(x) = \cos(x)$, $a = 0$; and
 (D) $f(x) = \log(x)$, $a = 1$.

Series

In[1]:= **Series[Exp[x], {x, 0, 5}]**

Out[1]=

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O[x]^6$$

In[2]:= **Series[Sin[x], {x, Pi, 3}]**

Out[2]=

$$-(-\pi + x) + \frac{(-\pi + x)^3}{6} + O[-\pi + x]^4$$

In[3]:= **Series[Cos[x], {x, 0, 4}]**

Out[3]=

$$1 - \frac{x^2}{2} + \frac{x^4}{24} + O[x]^5$$

In[4]:= **Series[Log[x], {x, 1, 3}]**

Out[4]=

$$(-1 + x) - \frac{(-1 + x)^2}{2} + \frac{(-1 + x)^3}{3} + O[-1 + x]^4$$

Series[Exp[x], {x, 0, 5}]
computes the power series expansion of e^x about the point $x=0$ up to order x^5 .

Series[Sin[x], {x, Pi, 3}]
computes the power series expansion of $\sin(x)$ about the point $x=\pi$ up to order $(x-\pi)^3$.

Series[Cos[x], {x, 0, 4}]
computes the power series expansion of $\cos(x)$ about the point $x=0$ up to order x^4 .

Remember that Log[x] denotes the natural logarithm function Ln(x).

Series[Log[x], {x, 1, 3}]
compute the power series expansion of $\ln(x)$ about the point $x=1$ up to order $(x-1)^3$.

Mathematica can also compute the general formula for the power series expansion of a function $y(x)$:

<pre>In[5]:= Series[y[x], {x, 0, 3}] Out[5]=</pre> $y[0] + y'[0] x + \frac{y''[0] x^2}{2} + \frac{y^{(3)}[0] x^3}{6} + O[x]^4$		<p>Series[y[x], {x, 0, 3}] calculates the power series expansion for $y(x)$ about the point $x=0$ to order x^3.</p>
<pre>In[6]:= Series[y[x], {x, a, 3}] Out[6]=</pre> $y[a] + y'[a] (-a + x) + \frac{y''[a] (-a + x)^2}{2} + \frac{y^{(3)}[a] (-a + x)^3}{6} + O[-a + x]^4$		<p>Series[y[x], {x, a, 3}] calculates the power series expansion for $y(x)$ about the point $x=a$ to order $(x-a)^3$.</p>

Mathematica can truncate (remove the remainder term) of the power series **Series[f[x], {x, a, n}]** with the command **Normal[Series[f[x], {x, a, n}]]**. Hence, with the **Normal** command, a polynomial is obtained. This polynomial serves as an approximation to the function $f(x)$. These ideas are illustrated below :

□ Example:

Let $f(x)=\sin(x)\cos(x)$. Compute the first 8 terms of the power series for $f(x)$ about $x=0$. Use the command **Normal** to remove the remainder term from the series. Call the resulting polynomial function $g(x)$. Compare the graphs of $f(x)$ and $g(x)$ on the interval $[-\pi/2,\pi/2]$. Graph the function $|f(x)-g(x)|$ on the interval $[-\pi/2,\pi/2]$.

Series

```

In[4]:=
  f[x_]:=Sin[x] Cos[x]
Out[4]=
  Cos[x] Sin[x]
In[5]:=
  Series[f[x],{x,0,8}]
Out[5]=
  x -  $\frac{2x^3}{3}$  +  $\frac{2x^5}{15}$  -  $\frac{4x^7}{315}$  + O[x]
In[6]:=
  g[x_]:=Normal[Series[f[x],{x,0,8}]]
Out[6]=
  x -  $\frac{2x^3}{3}$  +  $\frac{2x^5}{15}$  -  $\frac{4x^7}{315}$ 
In[8]:=
  Plot[{f[x],g[x]},{x,-Pi/2,Pi/2},
  PlotStyle->{GrayLevel[0],GrayLevel[.3]}]
Out[8]=
  -Graphics-

```

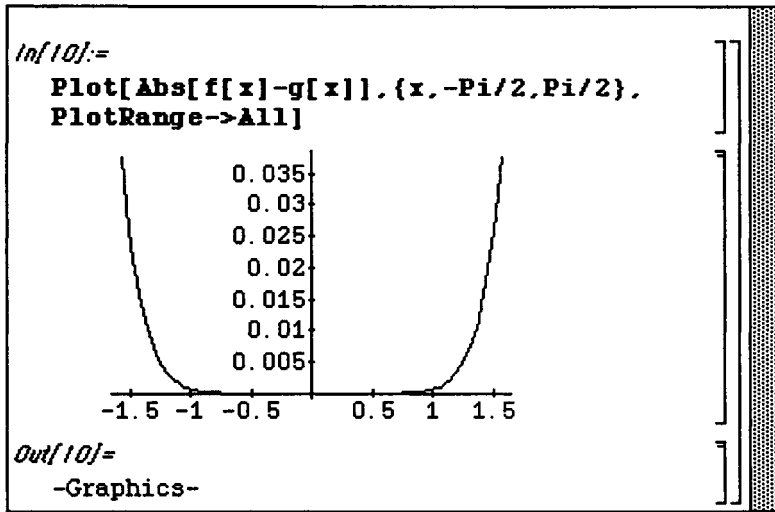
Define $f(x)=\sin(x)\cos(x)$.

Series[f[x],{x,0,8}]
computes the power series for $f(x)=\sin(x)\cos(x)$ about $x=0$ up to order x^8 .

The command

Normal[Series[f[x],{x,0,8}]]
removes the remainder term from the power series. The resulting polynomial is a function (of x).

Notice that the graphs of f and g on the interval $[-\pi/2,\pi/2]$ are almost the same.



To see that f and g are almost the same on $[-\pi/2, \pi/2]$, graph $|f(x)-g(x)|$ on the same interval

The effect of the option **PlotRange->All** is to guarantee that the entire graph is shown.

■ **Application:** Approximating the Remainder

Let f have (at least) $n+1$ derivatives in an interval containing a . If x is any number in the interval, then

$$f(x) = f(a) + \frac{f^{(1)}(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \dots + \frac{f^{(n-1)}(a)}{(n-1)!}(x-a)^{n-1} + \frac{f^{(n)}(a)}{n!}(x-a)^n +$$

Taylor Polynomial of degree n for $f(x)$ at a

$$\frac{f^{(n+1)}(z)}{(n+1)!}(x-a)^{n+1}$$

Taylor Remainder of Taylor Polynomial of degree n for $f(x)$ at a

where z is between a and x .

□ **Example:**

As in the above, let $f(x)=\sin(x) \cos(x)$. Compute the Taylor Remainder of the Taylor Polynomial of degree 8, 9 and 10 for $x=0$.

We proceed by defining a function that symbolically computes the Taylor remainder of degree n for $f[x]$ at 0:

```
SeriesRemainder
In[29]:=
f[x_]:=Sin[x] Cos[x]
Out[29]=
Cos[x] Sin[x]
In[30]:=
remainder[n_]:=
(D[f[x],{x,n+1}]/.x->z) x^(n+1)/
(n+1)!
```

remainder[n] is a function that gives the Taylor Remainder of $f(x)$ at 0 for each n (z is between 0 and x).

Then, `remainder[8]`, `remainder[9]`, and `remainder[10]` are the desired remainders:

<pre>In[34]:= remainder[8] Out[34]= 9 2 2 x (256 Cos[z] - 256 Sin[z]) ----- 362880</pre>	<p>remainder[8] calculates the remainder for $n=8$.</p>
<pre>In[35]:= remainder[9] Out[35]= 10 -4 x Cos[z] Sin[z] ----- 14175</pre>	<p>remainder[9] calculates the remainder for $n=9$.</p>
<pre>In[36]:= remainder[10] Out[36]= 11 2 2 x (-1024 Cos[z] + 1024 Sin[z]) ----- 39916800</pre>	<p>remainder[10] calculates the remainder for $n=10$.</p>

Compute the Taylor Remainder of the Taylor Polynomial of degree 11 for $x=0$. What is an upper bound for the remainder on the interval $[-\pi/2, \pi/2]$?

First compute the Taylor remainder of the Taylor polynomial of degree 11 for $x=0$ using the function `remainder` defined above:

<pre>In[37]:= remainder[11] Out[37]= 12 4 x Cos[z] Sin[z] ----- 467775</pre>	<p>remainder[11] calculates the remainder for $n=11$.</p>
--	---

To obtain an upper bound of the remainder on the interval $[-\pi/2, \pi/2]$, notice that for

$x \in [-\pi/2, \pi/2]$, $|x| \leq \frac{\pi}{2}$ and for all values of z , $|\cos(z)\sin(z)| \leq 1$. Thus, an upper bound on

remainder[11] is obtained by replacing x by $\pi/2$ and $\cos[z]\sin[z]$ by 1:

<pre>In[39]:= remainder[11] /. x->Pi/2 /. Cos[z]Sin[z] ->1 Out[39]= 12 Pi ----- 479001600 In[40]:= N[remainder[11] /. x->Pi/2 /. Cos[z]Sin[z] ->1] Out[40]= 0.00192957</pre>	<p><i>evaluates remainder[11] by replacing x by pi/2 and Cos(z)Sin(z) by 1. The result is an exact number. A numerical approximation is obtained with the subsequent calculation.</i></p>
---	---

■ **Application:** Series Solutions to Differential Equations

□ **Example:**

Find a function $y(x)$ that satisfies the ordinary differential equation $y'' - 4y' - 5y = 0$ and the initial conditions $y(0) = 1$ and $y'(0) = 5$.

Remark: In Chapter 5, the command **DSolve** will be used to solve the differential equation $y'' - 4y' - 5y = 0$ subject to the initial conditions $y(0) = 1$ and $y'(0) = 5$.

Since the point $x = 0$ is an ordinary point of the differential equation, the solution $y(x)$ is assumed to be the power

$$\text{series } y(x) = \sum_{k=0}^{\infty} \frac{y^{(k)}(0)}{k!} x^k$$

The power series is then substituted into the differential equation in order to determine the coefficients.

Applications of Series

```
In[7]:=
ser=Series[y[x],{x,0,5}] /.
y[0]->1 /. y'[0]->5
Out[7]=
1 + 5 x +  $\frac{y''[0] x^2}{2}$  +  $\frac{y^{(3)}[0] x^3}{6}$  +
 $\frac{y^{(4)}[0] x^4}{24}$  +  $\frac{y^{(5)}[0] x^5}{120}$  + O[x]
In[8]:=
equation=D[ser,{x,2}]-4D[ser,x]-
5 ser==0
Out[8]=
(-25 + y''[0]) +
 $(-25 - 4 y''[0] + y^{(3)}[0]) x +$ 
 $(-\frac{5 y''[0]}{2} - 2 y^{(3)}[0] +$ 
 $\frac{y^{(4)}[0]}{2}) x^2 +$ 
 $(\frac{-5 y^{(3)}[0]}{6} - \frac{2 y^{(4)}[0]}{3} +$ 
 $\frac{y^{(5)}[0]}{6}) x^3 + O[x] == 0$ 
```

`ser=Series[y[x],{x,0,5}] /.
y[0]->1 /. y'[0]->5`
is the power series for $y(x)$ at 0 with
 $y(0)$ replaced by 1 and $y'(0)$ replaced by
5.

`D[ser,{x,2}]` computes the second
derivative of `ser` with respect to x .

`D[ser,x]` computes the derivative
of `ser` with respect to x .

Consequently, `equation` is the
original differential equation with
 $y(x)$ replaced by `ser`.

If two series $\sum_{n=0}^{\infty} a_n x^n$ and
 $\sum_{n=0}^{\infty} b_n x^n$ satisfy the property
that $\sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty} b_n x^n$, then

$a_n = b_n$ for all n .

This may be rephrased as: "If
two power series are equal,
then coefficients of corresponding
terms are equal."

Mathematica equates the coefficients of like powers of x on each side of the equation with the command **LogicalExpand[equation]**.

```
In[9]:=
  linearequations=
    LogicalExpand[equation]
Out[9]=
  -25 + y''[0] == 0 &&
  -25 - 4 y''[0] + y(3)[0] == 0 &&
  -5 y''[0]
  ----- - 2 y(3)[0] +
  2
  (4)
  y(4)[0]
  ----- == 0 &&
  2
  (3)      (4)
  -5 y(3)[0] - 2 y(4)[0]
  ----- +
  6          3
  (5)
  y(5)[0]
  ----- == 0
  6
```

LogicalExpand[equation]
equates the corresponding coefficients
in equation ; the resulting system
of equations is named
linearequations.

The above system of equations is then solved using `Solve` in order to determine the values of the higher order derivatives of y evaluated at $x = 0$. Once these values are determined, they are substituted back into the power series to obtain an approximate solution to the differential equation.

```

In[10]:=
  values=Solve[ linearequations ]
Out[10]=
  (3)
  {(y'[0] -> 25, y [0] -> 125,
  (4)          (5)
  y [0] -> 625, y [0] -> 3125}
  }
In[11]:=
  ser /. values[[1]]
Out[11]=
  1 + 5 x +  $\frac{25 x^2}{2} + \frac{125 x^3}{6} +$ 
   $\frac{625 x^4}{24} + \frac{625 x^5}{24} + O[x]^6$ 

```

Solve[linearequations]

solves the system of equations

linearequations.

The solution set is named values.

ser /. values[[1]]

replaces the unknowns of ser

by the obtained solutions values.

In this case, be sure to type

values[[1]]

exactly. values

is a list and list operations

will be discussed in detail

in Chapters 4 and 5.

■ 3.6 Multi-Variable Calculus

■ Elementary Three-Dimensional Graphics

As was mentioned in **Chapter 1**, functions of more than one variable can be defined with *Mathematica*. Of particular interest are functions of two variables. The command which plots the graph of the function $f(x,y)$

over the rectangular domain R where $R: x_{\min} \leq x \leq x_{\max}, y_{\min} \leq y \leq y_{\max}$

is: `Plot3D[f[x,y], {x,xmin,xmax}, {y,ymin,ymax}]`. Since the graphs of functions of two variables are surfaces in three dimensions, the *Mathematica* command, `Plot3D`, must be introduced.

Mathematica can also plot the level curves of the function $f(x,y)$. (Recall : **Level curves** are curves in the xy -plane which satisfy the equation $f(x,y)=\text{constant}$.) Level curves are plotted with the command

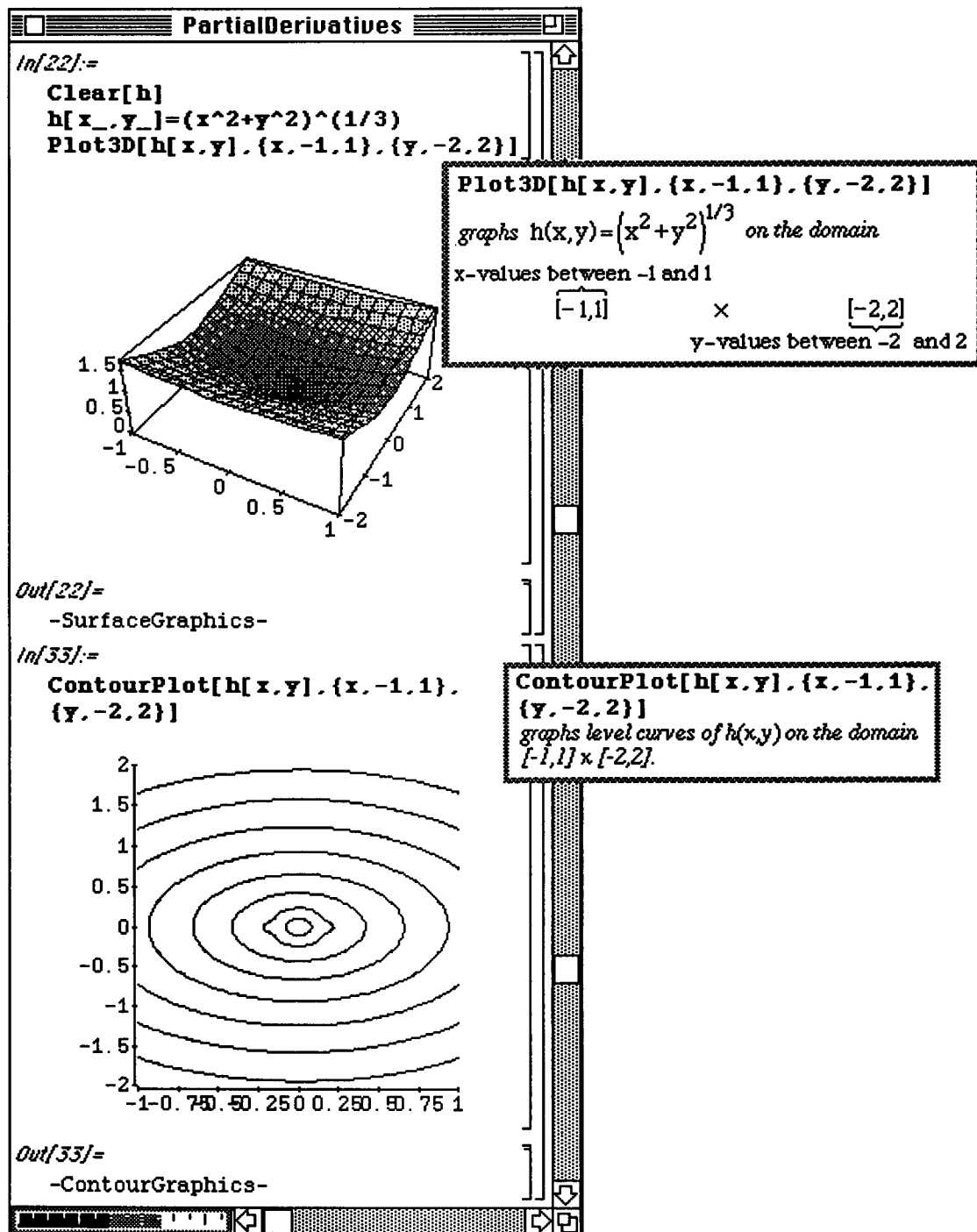
`ContourPlot[f[x,y], {x,xmin,xmax}, {y,ymin,ymax}]`.

These commands are demonstrated below using Version 1.2:

- Many options are available with the Version 2.0 command `ContourPlot` that are not available in earlier editions of *Mathematica* and are discussed last.

□ Example:

Let $h(x,y) = (x^2 + y^2)^{1/3}$. Graph h on the rectangle $[-1,1] \times [-2,2]$.



There are many options which can be included in the **Plot3D** command. The following examples illustrate the **Shading** and **PlotPoints** options.

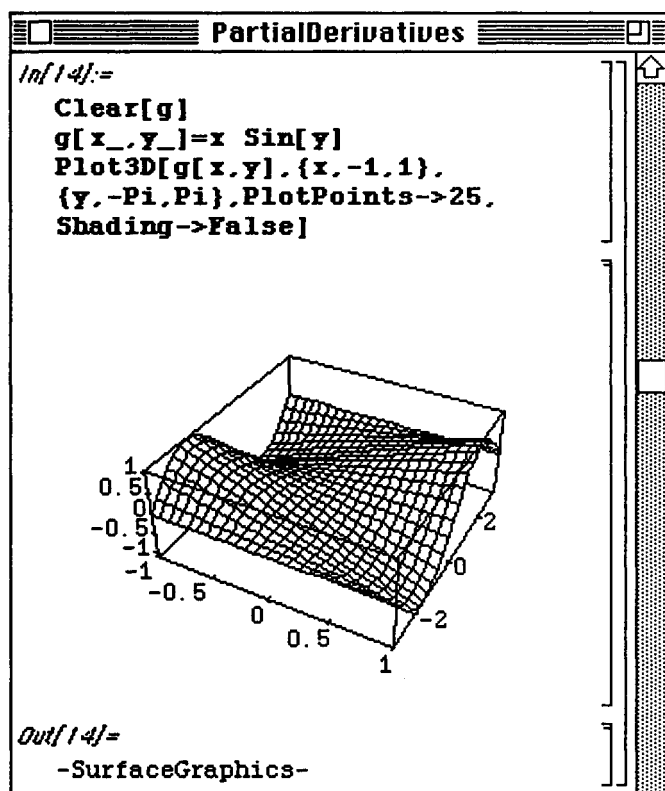
Shading→**False** causes *Mathematica* to NOT shade squares in the graph.

PlotPoints→**n** causes *Mathematica* to evaluate the function at n^2 points when plotting the graph. These n^2 points are called sample points. In the command,

Plot3D[**f**[**x**, **y**], {**x**, **xmin**, **xmax**}, {**y**, **ymin**, **ymax**}, **PlotPoints**→**n**], the sample points are obtained by dividing each interval [**xmin**, **xmax**] and [**ymin**, **ymax**] into **n** subintervals. Hence, a larger value of **n** yields a smoother graph.

Notice the difference in the shading in the following graph of $g(x,y)$ as opposed to the graph of $h(x,y)$ above. The graph of $g(x,y)$ is smoother than that of $h(x,y)$ since the default (the value assumed when not otherwise stated) on **PlotPoints** is 15. Thus, the **Plot3D** command which plotted $f(x,y)$ earlier automatically used 225 sample points.

□ Example:



After clearing all prior definitions of g ,
define $g(x,y)=x \sin(y)$

Plot3D[**g**[**x**, **y**], {**x**, -1, 1},
{**y**, -Pi, Pi}, **PlotPoints**→25,
Shading→**False**]

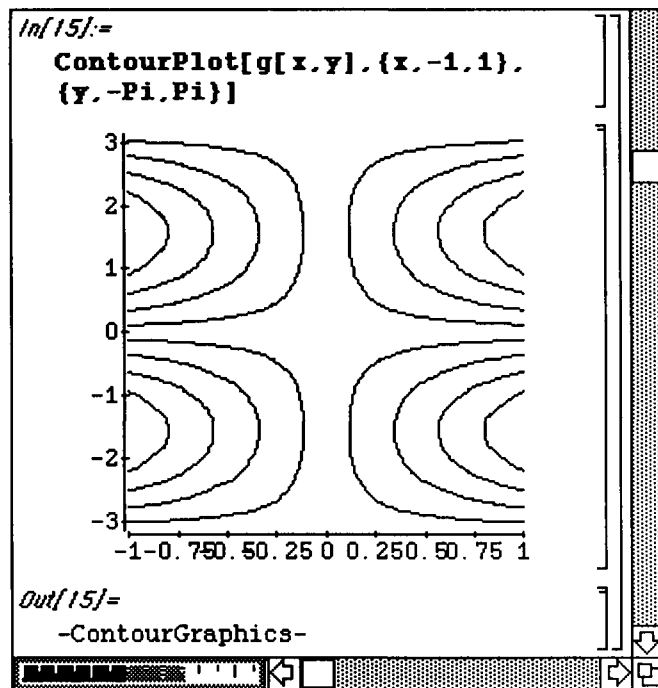
graphs $g(x,y)$ on the domain $[-1, 1] \times [-\pi, \pi]$.
The option

PlotPoints→25

makes the graph appear
smoother than before; the option

Shading→**False**

causes *Mathematica* to NOT shade each
square.



```

ContourPlot[g[x,y],{x,-1,1},
{y,-Pi,Pi}]

```

graphs several level curves for $g[x,y]$ on the domain $[-1,1] \times [-\pi,\pi]$.

• ContourPlot and Version 2.0

The **ContourPlot** command is much improved in Version 2.0. Evidence of this is observed in the contour plot of the function $g[x,y]$ defined below. The three-dimensional plot of this function is given in **plot3d**. Recall that contour levels represent intersections of planes of the form $g[x,y] = \text{constant}$ with the surface shown in **plot3d**. **cp1** contains the contour plot of g which is obtained without any of the **ContourPlot** options. This plot differs greatly from that obtained in Version 1.2 in that shading is included in all contour plots unless otherwise specified.

ContourOptions

```

Clear[g]
g[x_,y_]=Exp[-(x^2+y^2)/8]((Cos[x])^2+(Sin[y])^2)

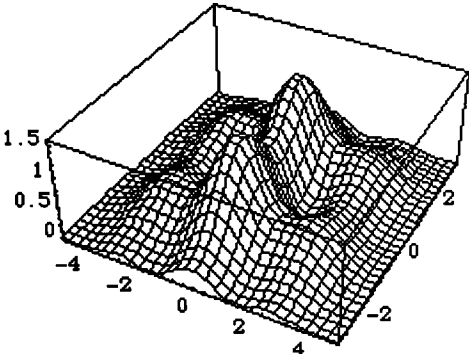
```

$$\frac{\cos^2(x) + \sin^2(y)}{e^{(x^2 + y^2)/8}}$$

```

plot3d=Plot3D[g[x,y],
{x,-5,5},{y,-Pi,Pi},
Shading->False,PlotPoints->30]

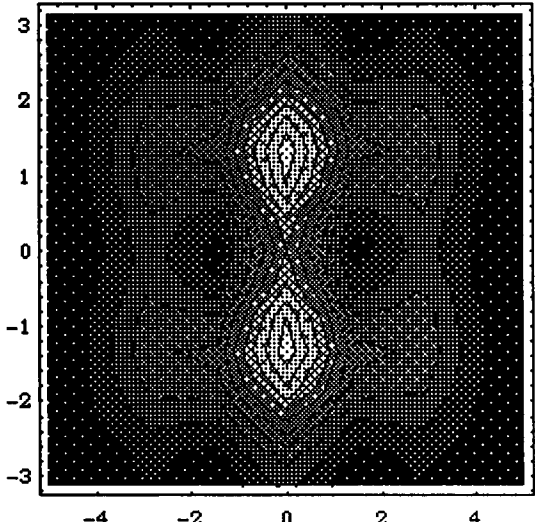
```



```

-SurfaceGraphics-
cp1=ContourPlot[g[x,y],{x,-5,5},{y,-Pi,Pi}]

```



```

-ContourGraphics-

```

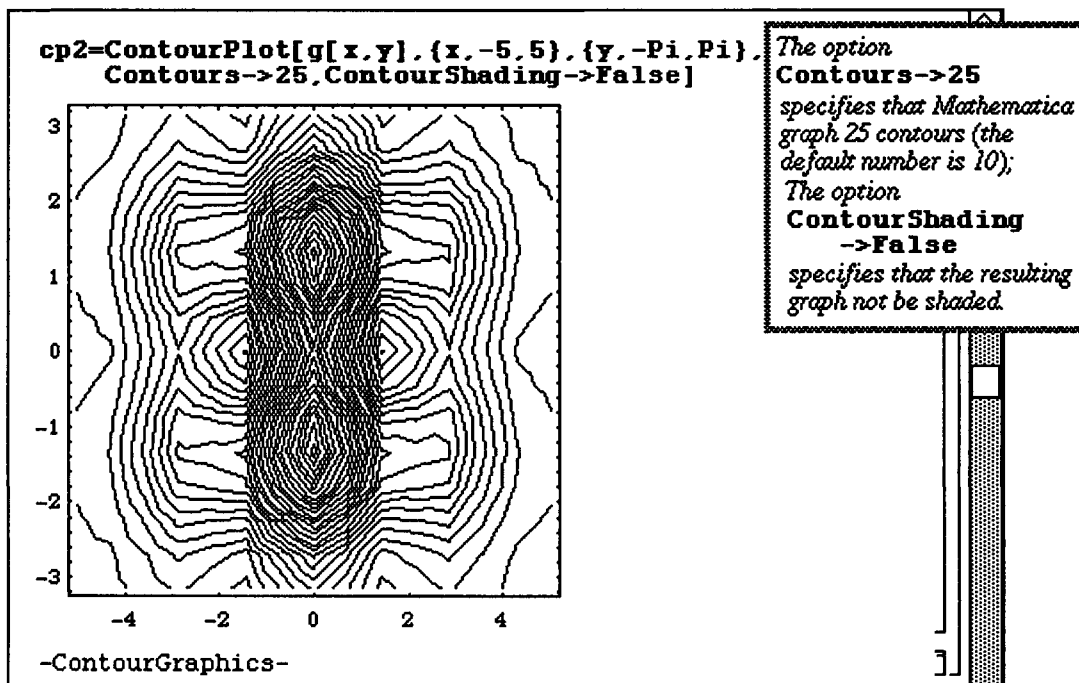
*To illustrate some of the features of the command **ContourPlot** we first define*

$$g(x,y) = \frac{\cos^2(x) + \sin^2(y)}{e^{(x^2+y^2)/8}}$$

and graph $g(x,y)$ on the rectangle $[-5,5] \times [-\pi,\pi]$.

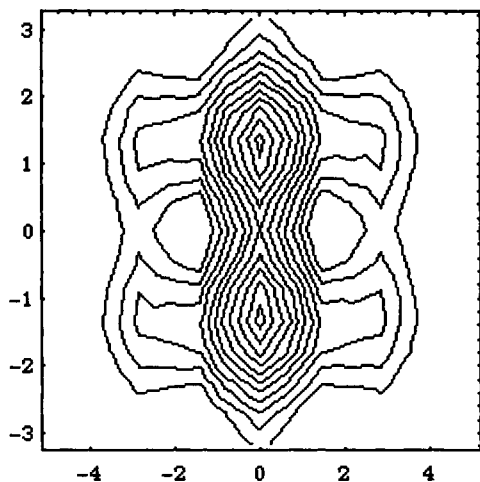
Creates a contour graph of $g(x,y)$ on the rectangle $[-5, 5] \times [-\pi, \pi]$.

As mentioned above, all contour plots are shaded unless the `ContourShading->False` option is employed. This option is illustrated below along with `Contours->k` which instructs *Mathematica* to use *k* contour levels. (This replaces the `ContourLevels` option in Version 1.2.) In `cp2`, 25 contour levels are used and shading is suppressed. In the previous contour plot, the default value of 10 contour levels was used.



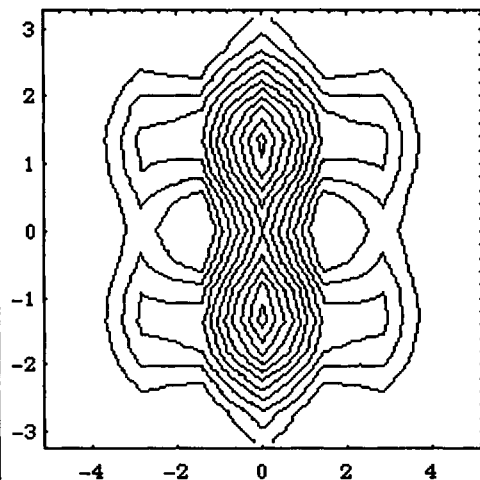
In the previous examples, *Mathematica* has selected the contour levels. However, these values can be chosen by the user with the `Contour->valuelist` option. This is demonstrated below with the table `contvals`. After this table is created, `Contours->contvals` forces *Mathematica* to use the contour levels given in `contvals`. Another option is shown below as well. The contour plot in `cp3` is redone in `cp4` with the addition of the `ContourSmoothing->Automatic` option. This causes the contour levels in `cp3` to appear smoother in `cp4`.

```
contvals=Table[i, {i, .25, 1.5, 1.25/10}]
{0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1., 1.125,
 1.25, 1.375, 1.5}
cp3=ContourPlot[g[x,y], {x, -5, 5}, {y, -Pi, Pi},
  Contours->contvals, ContourShading->False]
```



-ContourGraphics-

```
cp4=ContourPlot[g[x,y], {x, -5, 5}, {y, -Pi, Pi},
  Contours->contvals, ContourShading->False,
  ContourSmoothing->Automatic]
```



-ContourGraphics-

The option
**Contours->
contvals**
instructs *Mathematica*
to draw contours for
the numbers in the
list **contvals**.

The option
**ContourSmoothing
->Automatic**
instructs *Mathematica*
to attempt to make
each contour smooth
(compare with the
previous example).

■ Partial Differentiation

Partial derivatives can be calculated with *Mathematica* using the command

D[f[x,y],variable]

where **f[x,y]** is differentiated with respect to **variable**.

Second order derivatives can be found using **D[f[x,y],variable1,variable2]**

where **f[x,y]** is differentiated first with respect to **variable2** and then with respect to **variable1**.

□ Example:

Let $h(x,y) = (x^2 + y^2)^{1/3}$ (as above). Calculate $\frac{\partial h}{\partial x}$, $\frac{\partial h}{\partial y}$, and $\frac{\partial^2 h}{\partial y \partial x}$.

The screenshot shows a Mathematica notebook window titled "PartialDerivatives" with three input-output pairs:

- Input 23:** `D[h[x,y],x]`
Output 23:
$$\frac{2x}{3(x^2 + y^2)^{2/3}}$$
- Input 24:** `D[h[x,y],y]`
Output 24:
$$\frac{2y}{3(x^2 + y^2)^{2/3}}$$
- Input 25:** `D[h[x,y],y,x]`
Output 25:
$$\frac{-8xy}{9(x^2 + y^2)^{5/3}}$$

To the right of the notebook window, there are three text annotations:

- `D[h[x,y],x]` calculates $\frac{\partial h}{\partial x}$.
- `D[h[x,y],y]` calculates $\frac{\partial h}{\partial y}$.
- `D[h[x,y],y,x]` calculates $\frac{\partial^2 h}{\partial y \partial x}$.

Higher order derivatives with respect to the same variable can be determined with the command

D[f[x,y],{variable,n}]. This command computes the **n**th partial derivative of **f** with respect to **variable**.

For example, **D[f[x,y],{x,n}]** computes $\frac{\partial^n f}{\partial x^n}$.

□ **Example:**

Let $h(x, y) = (x^2 + y^2)^{1/3}$ (as above). Calculate $\frac{\partial^2 h}{\partial x^2}$ and $\frac{\partial^2 h}{\partial y^2}$.

The screenshot shows a Mathematica notebook window titled "PartialDerivatives" with the following content:

```
In[29]:=
D[h[x,y],{x,2}]
Out[29]=
      2
     -8 x
----- + -----
      2      2 5/3      2      2 2/3
     9 (x + y )      3 (x + y )

In[30]:=
Together[D[h[x,y],{x,2}]]
Out[30]=
      2      2
     -2 x + 6 y
-----
      2      2 5/3
     9 (x + y )

In[31]:=
Together[D[h[x,y],{y,2}]]
Out[31]=
      2      2
     6 x - 2 y
-----
      2      2 5/3
     9 (x + y )
```

Annotations on the right side of the notebook:

- Next to `D[h[x,y],{x,2}]`: `D[h[x,y],{x,2}]` computes $\frac{\partial^2 h}{\partial x^2}$.
- Next to `Together[D[h[x,y],{x,2}]]`: `Together[D[h[x,y],{x,2}]]` computes $\frac{\partial^2 h}{\partial x^2}$ and combines into a single fraction.
- Next to `Together[D[h[x,y],{y,2}]]`: `Together[D[h[x,y],{y,2}]]` computes $\frac{\partial^2 h}{\partial y^2}$ and combines into a single fraction.

A callout box with a dotted border contains the text: *Be sure to use square brackets correctly when several Mathematica commands are combined.*

■ **Other Methods of Computing Derivatives**

The command `Derivative` can also be used to compute derivatives of functions. For example, if $f[x]$ is a function of a single variable, the command `Derivative[1][f][a]` computes the derivative of f with respect to x and evaluates the result by replacing x by a ; the command `Derivative[n][f][a]` computes the n th derivative of f with respect to x and evaluates the result by replacing x by a .

Similarly, if $f[x, y]$ is a function of two variables, the command `Derivative[1, 0][f][a, b]` computes the partial derivative of f with respect to x and evaluates the result by replacing x by a and y by b ; the command `Derivative[0, 1][f][a, b]` computes the partial derivative of f with respect to y and evaluates the result by replacing x by a and y by b ; and the command `Derivative[n, m][f][a, b]` computes the n th partial derivative of f with respect to x and then the m th partial derivative of f with respect to y and evaluates the result by replacing x by a and y by b .

□ Example:

Use the command **Derivative** to compute $\frac{\partial g}{\partial x}$, $\frac{\partial g}{\partial y}$, $\frac{\partial^2 g}{\partial x \partial y}$, $\frac{\partial^2 g}{\partial x^2}$, $\frac{\partial^2 g}{\partial y^2}$, and $\frac{\partial^3 g}{\partial x \partial y^2} \left(\frac{\pi}{3}, \frac{\pi}{6} \right)$ if

$$g(x, y) = e^{-\frac{(x^2 + y^2)}{8}} (\cos^2(x) + \sin^2(y)).$$

After defining **g**, we illustrate that **Derivative[1, 0][g][x, y]** and **D[g[x, y], x]** both produce the same result.

PartialDerivatives

```

In[59]:=
Clear[g]
g[x_, y_]=Exp[-(x^2+y^2)/8]*
          (Cos[x]^2+Sin[y]^2)

Out[59]=
          Cos[x]^2 + Sin[y]^2
          -----
          E^(x^2 + y^2)/8

In[60]:=
gx=Derivative[1, 0][g][x, y]

Out[60]=
          -2 Cos[x] Sin[x]
          -----
          E^(x^2 + y^2)/8
          x (Cos[x]^2 + Sin[y]^2)
          -----
          4 E^(x^2 + y^2)/8

In[61]:=
D[g[x, y], x]

Out[61]=
          -2 Cos[x] Sin[x]
          -----
          E^(x^2 + y^2)/8
          x (Cos[x]^2 + Sin[y]^2)
          -----
          4 E^(x^2 + y^2)/8

```

After clearing all prior definitions of **g**, define

$$g(x, y) = e^{-\frac{(x^2 + y^2)}{8}} (\cos^2(x) + \sin^2(y)).$$

Since **g** is defined on two lines, be sure to include the ***** to denote multiplication.

Both **Derivative[1, 0][g][x, y]** and **D[g[x, y], x]** produce $\frac{\partial g}{\partial x}$.

Similarly `Derivative[1,1][g][x,y]` and `Derivative[g,x,y]` produce the same result:

`In[62]:=`

`gxy=Derivative[1,1][g][x,y]`

`Out[62]=`

$$\frac{y \cos[x] \sin[x]}{2 E^{(x^2 + y^2)/8}} - \frac{x \cos[y] \sin[y]}{2 E^{(x^2 + y^2)/8}} + \frac{x y (\cos[x]^2 + \sin[y]^2)}{16 E^{(x^2 + y^2)/8}}$$

`In[63]:=`

`D[g[x,y],x,y]`

`Out[63]=`

$$\frac{y \cos[x] \sin[x]}{2 E^{(x^2 + y^2)/8}} - \frac{x \cos[y] \sin[y]}{2 E^{(x^2 + y^2)/8}} + \frac{x y (\cos[x]^2 + \sin[y]^2)}{16 E^{(x^2 + y^2)/8}}$$

Similarly, both
`Derivative[1,1][g][x,y]`
 and `D[g[x,y],x,y]` compute
 $\frac{\partial^2 g}{\partial x \partial y}$.

In[64]:=

Derivative[2,0][g][x,y]

Out[64]=

$$\frac{-2 \cos[x]^2}{E^{(x^2 + y^2)/8}} + \frac{x \cos[x] \sin[x]}{E^{(x^2 + y^2)/8}} +$$

$$\frac{2 \sin[x]^2}{E^{(x^2 + y^2)/8}} - \frac{\cos[x]^2 + \sin[y]^2}{4 E^{(x^2 + y^2)/8}} +$$

$$\frac{x^2 (\cos[x]^2 + \sin[y]^2)}{16 E^{(x^2 + y^2)/8}}$$

In[65]:=

Derivative[0,2][g][x,y]

Out[65]=

$$\frac{2 \cos[y]^2}{E^{(x^2 + y^2)/8}} - \frac{y \cos[y] \sin[y]}{E^{(x^2 + y^2)/8}} -$$

$$\frac{2 \sin[y]^2}{E^{(x^2 + y^2)/8}} - \frac{\cos[x]^2 + \sin[y]^2}{4 E^{(x^2 + y^2)/8}} +$$

$$\frac{y^2 (\cos[x]^2 + \sin[y]^2)}{16 E^{(x^2 + y^2)/8}}$$

Derivative[2,0][g][x,y]computes $\frac{\partial^2 g}{\partial x^2}$; the same result would

be obtained with the command

D[g[x,y],{x,2}]**Derivative[0,2][g][x,y]**computes $\frac{\partial^2 g}{\partial y^2}$; the same result would

be obtained with the command

D[g[x,y],{y,2}].

In[66]:= **value=Derivative[1,2][g][Pi/3,Pi/6]**

Out[66]=

$$\frac{\text{Sqrt}[3]}{8 E^{(5 \text{ Pi}^2)/288}} - \frac{7 \text{ Pi}}{96 E^{(5 \text{ Pi}^2)/288}} + \frac{\text{Pi}^2}{128 \text{ Sqrt}[3] E^{(5 \text{ Pi}^2)/288}} - \frac{\text{Pi}^3}{13824 E^{(5 \text{ Pi}^2)/288}}$$

In[67]:=

Together[value]

Out[67]=

$$\frac{64 3^{7/2} - 1008 \text{ Pi} + 4 3^{5/2} \text{ Pi}^2 - \text{Pi}^3}{13824 E^{(5 \text{ Pi}^2)/288}}$$

In[68]:=

N[value]

Out[68]=

0.0250284

computes $\frac{\partial^3 g}{\partial x \partial y^2}$

and then evaluates by replacing x by $\pi/3$ and y by $\pi/6$. The resulting number is named **value**.

Writes **value** as a single fraction.

gives a numerical approximation of **value**.

■ **Application: Classifying Critical Points**

Recall the definition of a critical point :

- Let $f(x,y)$ be a real-valued function with continuous second-order partial derivatives. A **critical point** of f is a point (x_0, y_0) in the interior of the domain of f for which $f_x(x_0, y_0) = 0$ and $f_y(x_0, y_0) = 0$.

□ **Remark:** The following notation is used:

$$f_x(x, y) = \frac{\partial f}{\partial x}(x, y), \quad f_y(x, y) = \frac{\partial f}{\partial y}(x, y), \quad f_{yx}(x, y) = \frac{\partial f}{\partial x \partial y}(x, y), \quad f_{xx}(x, y) = \frac{\partial^2 f}{\partial x^2}(x, y), \quad \text{and} \quad f_{yy}(x, y) = \frac{\partial^2 f}{\partial y^2}(x, y)$$

Critical points are classified using the following test:

(Second-Derivative Test for Extrema) Let

$$D(f; (x_0, y_0)) = \begin{vmatrix} f_{xx}(x_0, y_0) & f_{xy}(x_0, y_0) \\ f_{xy}(x_0, y_0) & f_{yy}(x_0, y_0) \end{vmatrix} = (f_{xx}(x_0, y_0))(f_{yy}(x_0, y_0)) - (f_{xy}(x_0, y_0))^2.$$

- (a) If $D > 0$ and $f_{xx}(x_0, y_0) > 0$, then f has a relative minimum at (x_0, y_0) ;
- (b) If $D > 0$ and $f_{xx}(x_0, y_0) < 0$, then f has a relative maximum at (x_0, y_0) ;
- (c) If $D < 0$, then f has a saddle point at (x_0, y_0) ; and
- (d) If $D = 0$, no conclusion can be drawn and (x_0, y_0) is called a **degenerate critical point**.

□ Example:

Locate and classify all critical points of the function $f(x,y) = -120x^3 - 30x^4 + 18x^5 + 5x^6 + 30xy^2$.

In order to find the critical points of $f(x,y)$, the derivatives $f_x(x,y)$ and $f_y(x,y)$

are calculated and set equal to zero. These steps are shown below. Notice how the derivatives are given names to make using them later in the problem easier.

CriticalPoints

Begin by clearing all prior definitions of f and defining

```
In[1]:=
Clear[f]
f[x_,y_]=
-120x^3-30x^4+18x^5+5x^6+30x y^2
```

$f(x,y) = -120x^3 - 30x^4 + 18x^5 + 5x^6 + 30xy^2$

Remember to include the space between x and y^2 to denote multiplication.

```
Out[1]=
      3      4      5      6
-120 x  - 30 x  + 18 x  + 5 x  +
      2
30 x y
```

dfx=D[f[x,y],x]

computes $\frac{\partial f}{\partial x}$ and names it dfx.

```
In[2]:=
dfx=D[f[x,y],x]
Out[2]=
      2      3      4      5
-360 x  - 120 x  + 90 x  + 30 x  +
      2
30 y
```

dfy=D[f[x,y],y]

computes $\frac{\partial f}{\partial y}$ and names it dfy.

```
In[3]:=
dfy=D[f[x,y],y]
Out[3]=
60 x y
```

Solve[{dfx==0,dfy==0},{x,y}]

solves the system of equations

$$\begin{cases} \frac{\partial f(x,y)}{\partial x} = 0 \\ \frac{\partial f(x,y)}{\partial y} = 0 \end{cases}$$

for x and y; the solutions are the critical points.

```
Out[4]=
{{x -> -2, y -> 0},
 {x -> -3, y -> 0},
 {x -> 2, y -> 0},
 {x -> 0, y -> 0},
 {x -> 0, y -> 0}}
```

The second derivatives $\frac{\partial^2 f}{\partial x^2}$, $\frac{\partial^2 f}{\partial y^2}$, and $\frac{\partial^2 f}{\partial x \partial y}$ are computed below. Recall that if

$\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial^2 f}{\partial x \partial y}$, and $\frac{\partial^2 f}{\partial y \partial x}$ are continuous on an open set, then $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$

for each point in the set. Therefore, since $f(x,y)$ is a polynomial and, thus, continuous for all values of x and y , only three second order derivatives need to be calculated.

CriticalPoints

```

In[5]:=
  dfxx=D[f[x,y].{x,2}]
Out[5]=
  -720 x - 360 x2 + 360 x3 + 150 x4

In[6]:=
  dfyy=D[f[x,y].{y,2}]
Out[6]=
  60 x

In[7]:=
  dfxy=D[f[x,y].x,y]
Out[7]=
  60 y

```

dfxx=D[f[x,y].{x,2}]
 computes $\frac{\partial^2 f}{\partial x^2}$
 and names it **dfxx**.

dfyy=D[f[x,y].{y,2}]
 computes $\frac{\partial^2 f}{\partial y^2}$
 and names it **dfyy**.

dfxy=D[f[x,y].x,y]
 computes $\frac{\partial^2 f}{\partial x \partial y}$
 and names it **dfxy**.

The second-order derivative $\frac{\partial^2 f}{\partial^2 x}$ is evaluated at the critical points for later use in the second derivative test:

The screenshot shows a window titled "CriticalPoints" with a list of input and output commands. The input commands are:

- `In[3]:= dfxx /. x -> -2`
- `In[4]:= dfxx /. x -> -3`
- `In[6]:= dfxx /. x -> 2`
- `In[5]:= dfxx /. x -> 0`

 The corresponding output values are:

- `Out[3]= -480`
- `Out[4]= 1350`
- `Out[6]= 2400`
- `Out[5]= 0`

 To the right of the window, there are four explanatory text blocks:

- `dfxx /. x -> -2`
evaluates dfxx when x = -2.
- `dfxx /. x -> -3`
evaluates dfxx when x = -3.
- `dfxx /. x -> 2`
evaluates dfxx when x = 2.
- `dfxx /. x -> 0`
evaluates dfxx when x = 0.

The discriminant $\frac{\partial^2 f}{\partial^2 x}(x_0, y_0) \frac{\partial^2 f}{\partial^2 y}(x_0, y_0) - \left(\frac{\partial^2 f}{\partial x \partial y}(x_0, y_0) \right)^2$.

is defined as a function so that its value at the critical points can be computed easily.

The screenshot shows a Mathematica notebook window titled "CriticalPoints". The notebook contains the following input and output cells:

```

In[8]:=
discriminant[a_, b_] :=
dfxx dfyy - (dfxy)^2 /. x -> a /. y -> b

In[9]:=
discriminant[2, 0]
Out[9]=
288000

In[10]:=
discriminant[-2, 0]
Out[10]=
57600

In[11]:=
discriminant[-3, 0]
Out[11]=
-243000

In[12]:=
discriminant[0, 0]
Out[12]=
0

```

Four callout boxes on the right side of the notebook provide explanations for the output cells:

- For In[9]: computes $dfxx dfyy - (dfxy)^2$ and then replaces x by 2 and y by 0 .
- For In[10]: computes $dfxx dfyy - (dfxy)^2$ and then replaces x by -2 and y by 0 .
- For In[11]: computes $dfxx dfyy - (dfxy)^2$ and then replaces x by -3 and y by 0 .
- For In[12]: computes $dfxx dfyy - (dfxy)^2$ and then replaces x by 0 and y by 0 .

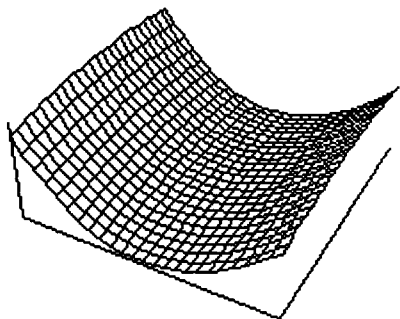
A larger callout box at the top right explains the function definition in In[8]: computes $dfxx dfyy - (dfxy)^2$ and then replaces x by a and y by b .

The **Plot3D** command is used to verify that the critical points have been classified correctly by plotting $f(x,y)$ around each critical point.

These graphs also illustrate some of the **Plot3D** options.

```
Plot3D[f[x,y], {x, -3.1, -2.9}, {y, -.1, .1},
Shading->False, Ticks->None,
PlotPoints->25,
Boxed->False]
```

$(-3,0)$ is an inflection point.

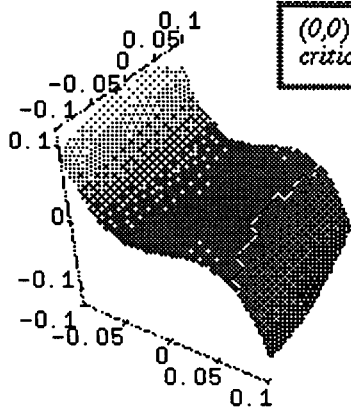


The option **Shading->False** specifies that the polygons *NOT* be shaded; the option **Ticks->None** specifies that no tick marks be placed on any axis; the option **Boxed->False** specifies that a box is *NOT* placed around the graph.

```
Plot3D[f[x,y], {x, -.1, .1}, {y, -.1, .1},
Boxed->False, PlotPoints->25,
PlotLabel->"Degenerate",
Mesh->False, BoxRatios->{1,1,1}]
```

Degenerate

$(0,0)$ is a degenerate critical point.

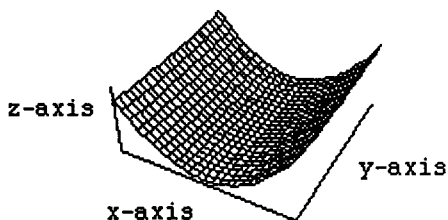


The option

PlotLabel->"Degenerate" labels the graph "Degenerate"; the option **Mesh->False** hides the edges of each polygon (but each polygon is shaded); the option **BoxRatios->{1,1,1}** specifies that the ratio of the lengths of the bounding 3-dimensional box ratios are 1:1:1.

```
Plot3D[f[x,y], {x, 1.9, 2.1},
{y, -.1, .1}, Shading->False,
Ticks->None, AxesLabel->
{"x-axis", "y-axis", "z-axis"},
PlotPoints->25, Boxed->False]
```

(2,0) is a minimum



The option

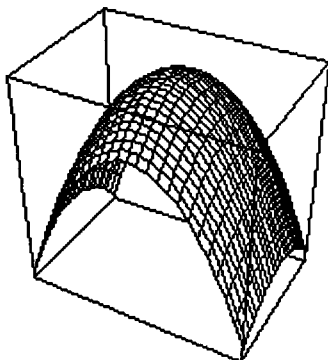
AxesLabel->

{"x-axis", "y-axis", "z-axis"}

specifies that the x-axis be marked "x-axis", the y-axis marked "y-axis", and the z-axis marked "z-axis".

```
Plot3D[f[x,y], {x, -2.1, -1.9}, {y, -.1, .1},
PlotPoints->25, PlotLabel->"Maximum",
Axes->None, BoxRatios->{1.5, 1, 1.5},
Shading->False]
```

Maximum



The option

Axes->None

specifies that axes are NOT included in the graph.

■ Application: Tangent Planes

Let $z=f(x,y)$. Then a function that describes the plane tangent to the graph of $z=f(x,y)$

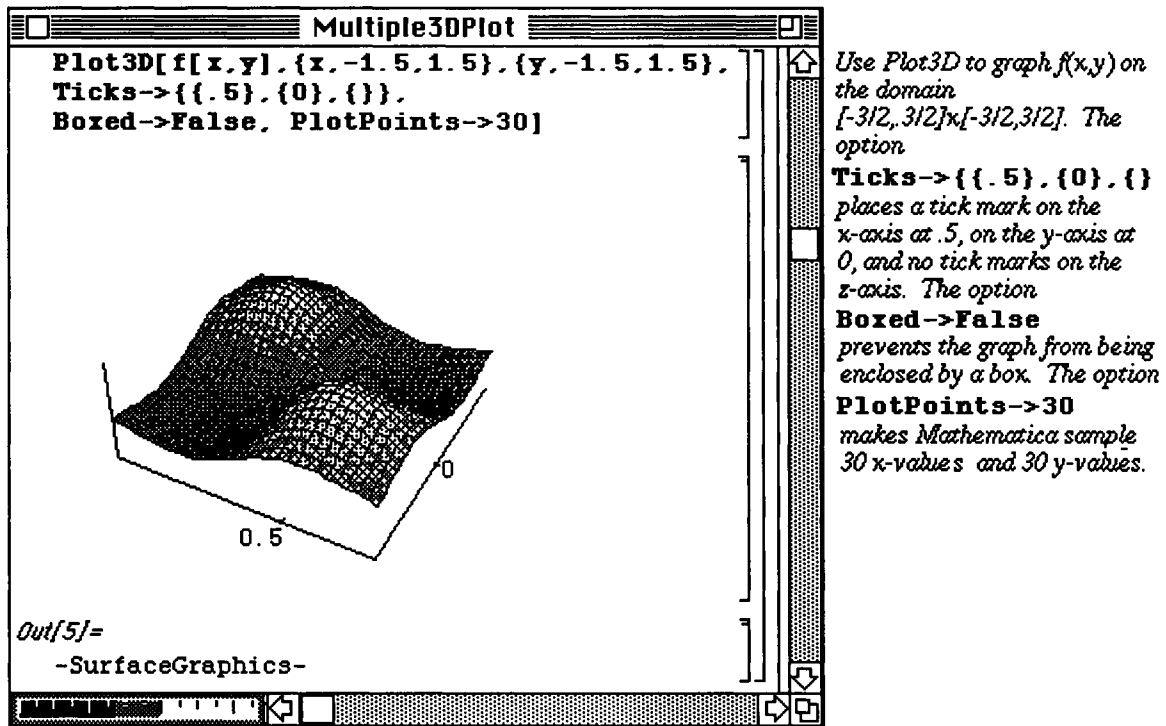
at the point $(x_0, y_0, z_0 = f(x_0, y_0))$ is given by

$$\text{tanplane}_{(f, x_0, y_0)}(x, y) = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0) + z_0.$$

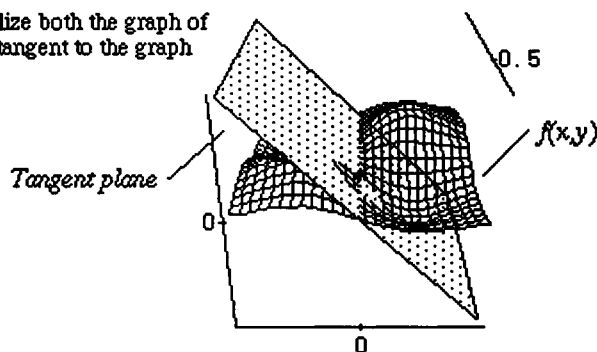
□ Example:

Find a function that describes the plane that is tangent to the graph of $f(x, y) = -6xy e^{-x^2 - y^2}$ at the point $(.5, 0)$.

Mathematica can be used to plot the graph of $f(x, y)$. Using some of the options available to **Plot3D**, this graph can be plotted in a manner which will aid in visualizing the tangent plane to the graph at $(.5, 0)$.



Mathematica can be used to visualize both the graph of $f(x,y)$ and the graph of the plane tangent to the graph of $f(x,y)$ when $x=.5$ and $y=0$.



```

Multiple3DPlot
In[2]:=
Clear[f]
f[x_,y_]=-6 x y Exp[-x^2-y^2]
Out[2]=
      2      2
     -x  -y
    -6 E    x y
In[3]:=
D[f[x,y],x] /. x->.5 /. y->0
Out[3]=
0
In[4]:=
D[f[x,y],y] /. x->.5 /. y->0
Out[4]=
-2.3364

```

Begin by clearing prior definitions of f and define $f(x,y)=-6xye^{-x^2-y^2}$.

Don't forget to include the space between x and y to denote multiplication.

$D[f[x,y],x] /. x->.5 /. y->0$ computes $\frac{\partial f}{\partial x}$ and then evaluates $\frac{\partial f}{\partial x}$ when $x=.5$ and $y=0$.

$D[f[x,y],y] /. x->.5 /. y->0$ computes $\frac{\partial f}{\partial y}$ and then evaluates $\frac{\partial f}{\partial y}$ when $x=.5$ and $y=0$.

Hence, the tangent plane is defined by the function $z = 0(x - .5) - 2.3364(y - 0)$ or, $z = -2.3364y$

o f and z can each be graphed with the command `Plot3D`. However, prior to the release of Version 2.0, two 3-dimension graphics objects created with `Plot3D` could not be shown simultaneously. However, Version 2.0 permits two (or more) objects created with the command `Plot3D` to be shown simultaneously with the command `Show`. In the following example, f is graphed on the rectangle $[-3,3] \times [-2,2]$, the resulting graph is not displayed and named `plotf`. Similarly, z is graphed on the rectangle $[-2,2] \times [-1,1]$, the resulting graph is not displayed and named `plotz`. The graphs are shown together (but not actually displayed) with the command `both=Show[plotf,plotz,BoxRatios->{1,1,1}]`.

```

In[23]:=
f[x_,y_]=-6x y Exp[-x^2-y^2];
z[x_,y_]=-2.3364y;

plotf=Plot3D[f[x,y],{x,-3,3},{y,-2,2},
  PlotPoints->25,
  DisplayFunction->Identity];

plotz=Plot3D[z[x,y],{x,-2,2},{y,-1,1},
  Shading->False,Ticks->None,
  DisplayFunction->Identity];

both=Show[plotf,plotz,BoxRatios->{1,1,1}];

```

defines $f(x,y) = -6xye^{-x^2-y^2}$ and $z(x,y) = -2.3364y$.

`plotf` corresponds to the graph of f .

`plotz` corresponds to the graph of z .

`both` consists of both `plotf` and `plotz`.

Since `plotf`, `plotz`, and `both` are graphics objects, they may be shown in a single graphics cell with the command `GraphicsArray`:

```

In[24]:=
Show[GraphicsArray[{plotf,plotz,both}]]

```

Out[24]=
-GraphicsArray-

■ **Application: Lagrange Multipliers**

Certain types of optimization problems can be solved using the method of Lagrange multipliers. This method is based on the following theorem :

□ **Lagrange's Theorem:**

Let f and g have continuous partial derivatives and f have an extremum at a point

(x_0, y_0) on the smooth constraint curve $g(x, y) = c$. If $g_x(x_0, y_0) \neq 0$ and $g_y(x_0, y_0) \neq 0$,

then there is a real number λ such that $f_x(x_0, y_0) = \lambda g_x(x_0, y_0)$, $f_y(x_0, y_0) = \lambda g_y(x_0, y_0)$; and

$g(x_0, y_0) = c$.

□ **Example:**

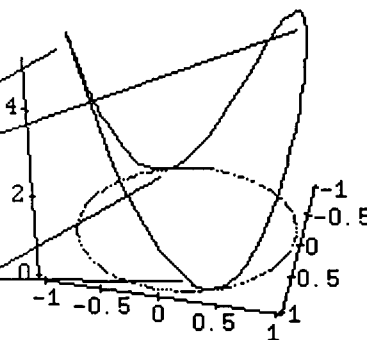
Find the maximum and minimum values of $f(x, y) = 4y^2 - 4xy + x^2$ subject to the constraint $x^2 + y^2 = 1$.

The commands used to create the following graphics are discussed in the **Appendix**.

Mathematica can be used to visualize the graph of $z=f(x, y)$ for (x, y) that satisfy the equation $g(x, y)=0$.

These points correspond to maxima.

These points correspond to minima.



The first order derivatives (with respect to x and y) of f and g are computed in order that Lagrange's Theorem can be applied. (The λ in Lagrange's Theorem is represented in the calculations below as ll .)

LagrangeMults

```

In[121]:=
Clear[f,g,dfx,dfy,dgx,dgy]
f[x_,y_]=4 y^2-4 x y+x^2
g[x_,y_]=x^2+y^2-1;

```

Since objects will be named $f, g, dfx, dfy, dgx,$ and $dgy,$ begin by clearing all prior definitions then define $f(x,y)=4y^2-4xy+x^2$ and $g(x,y)=x^2+y^2-1.$

```

In[122]:=
dfx=D[f[x,y],x]
dfy=D[f[x,y],y]
dgx=D[g[x,y],x]
dgy=D[g[x,y],y];

```

A semi-colon placed at the end of a command prevents the output from being shown.

```

In[123]:=
eq1=dfx==ll dgx

```

We use ll to represent the lambda from Lagrange's Theorem.

```

Out[123]=
2 x - 4 y == 2 ll x

```

```

In[124]:=
eq2=dfy==ll dgy

```

```

Out[124]=
-4 x + 8 y == 2 ll y

```

```

In[125]:=
eq3=g[x,y]==0

```

```

Out[125]=
      2      2
-1 + x  + y  == 0

```

eq1 represents the equation $\frac{\partial f(x,y)}{\partial x} = \lambda \frac{\partial g(x,y)}{\partial x}$.

eq2 represents the equation $\frac{\partial f(x,y)}{\partial y} = \lambda \frac{\partial g(x,y)}{\partial y}$.

eq3 represents the equation $g(x,y)=0$.

The values of x , y , and λ which satisfy the system of three equations in Lagrange's Theorem are determined using **Solve**. The solutions of this system are ordered triples (x, y, λ) . The values of x and y in each ordered triple represent the point at which f may have a maximum or minimum value.

```

LagrangeMults
In[126]:=
  Solve[{eq1, eq2, eq3}, {x, y, ll}] // N
Out[126]:=
  {{ll -> 0., y -> 0.447214,
    x -> 0.894427},
  {ll -> 0., y -> -0.447214,
    x -> -0.894427},
  {ll -> 5., y -> -0.894427,
    x -> 0.447214},
  {ll -> 5., y -> 0.894427,
    x -> -0.447214}}

```

solves the system of equations for x , y and ll . The values of x and y that result in the maximum and minimum values of f subject to the constraint will occur at these points.

Note: In this case, Mathematica produces exact values for x , y , and ll . However, since they are cumbersome to manipulate, we work with numerical approximations instead.

Thus, the maximum and minimum values of f are found by substituting these points back into the function $f(x,y)$ and comparing the resulting values of f .

```

In[127]:=
  f[x,y] /. x->.447214 /. y->-.894427
Out[127]=
  5.
In[128]:=
  f[x,y] /. x->-.447214 /. y->.894427
Out[128]=
  5.
In[129]:=
  f[x,y] /. y->.447214 /. x->.894427
Out[129]=
  -12
  1. 10
In[130]:=
  f[x,y] /. y->-.447214 /. x->-.894427
Out[130]=
  -12
  1. 10
  
```

calculates $f(x,y)$ when $x=.447214$ and $y=-.894427$

calculates $f(x,y)$ when $x=-.447214$ and $y=.894427$.

calculates $f(x,y)$ when $x=.894427$ and $y=.447214$.

calculates $f(x,y)$ when $x=-.894427$ and $y=-.447214$.

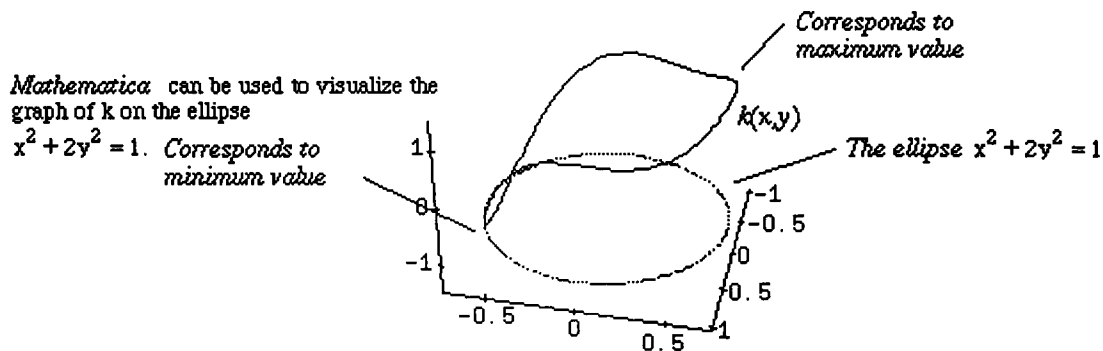
Consequently, the maximum value is 5 and the minimum value is 0. Notice that

$1 \cdot 10^{-12}$ is assumed to be zero.

In fact, the command `Chop[f[x,y] /. y->-.447214 /. x->-.894427]` yields zero.

□ Example:

Find the maximum and minimum values of $k(x, y) = x^2 + 4y^3$ subject to the constraint $x^2 + 2y^2 = 1$.



This problem is solved much like the previous example. Notice how the first derivatives are calculated and the system of equations stated in Lagrange's Theorem are established in a single command. The numerical approximation of the solutions of this system are then found.

```

LagrangeMults

In[2]:=
Clear[k,h]
k[x_,y_]=x^2+4y^3
h[x_,y_]=x^2+2y^2-1;

In[3]:=
dkx=D[k[x,y],x]
dky=D[k[x,y],y]
dhx=D[h[x,y],x]
dhy=D[h[x,y],y]
eq1=dkx==ll dhx
eq2=dky==ll dhy
eq3=h[x,y]==0;

In[4]:=
Solve[{eq1,eq2,eq3},{x,y,ll}] // N

Out[4]:=
{{ll -> 1., y -> 0.333333,
  x -> 0.881917},
 {ll -> 1., y -> 0.333333,
  x -> -0.881917},
 {ll -> 1., y -> 0., x -> 1.},
 {ll -> 1., y -> 0., x -> -1.},
 {ll -> 2.12132, y -> 0.707107,
  x -> 0.}, {ll -> -2.12132,
  y -> -0.707107, x -> 0.}}

```

Notice that several commands can be evaluated if they are combined into a single input cell. The semi-colon prevents the output from being shown.

Since the goal is to find the maximum and minimum values of $k(x,y)$ subject to the constraint $x^2+2y^2=1$

begin by defining $k(x,y)=x^2+4y^3$

and $h(x,y)=x^2+2y^2-1$.

eq1 represents $\frac{\partial k(x,y)}{\partial x} = \lambda \frac{\partial h(x,y)}{\partial x}$;

eq2 represents $\frac{\partial k(x,y)}{\partial y} = \lambda \frac{\partial h(x,y)}{\partial y}$;

and eq3 represents $h(x,y)=0$.

Solve solves eq1, eq2, and eq3 for x , y , and λ . Even though Mathematica finds the exact roots, we use numerical approximations for convenience.

The values of x and y that maximize and minimize k must occur at these points. To determine which values maximize k and which values minimize k , evaluate $k(x,y)$ for each set of values.

The points which satisfy the system are substituted into $k(x,y)$ to obtain the maximum and minimum values of the function by comparing the values obtained.

```

In[5]:=
k[.88197,.333333] computes
Out[5]=
0.926019
k(.88197,.333333).

In[6]:=
k[-.88197,.333333] computes
Out[6]=
0.926019
k(-.88197,.333333)

In[7]:=
k[1,0] computes k(1,0)
Out[7]=
1

In[8]:=
k[-1,0] computes k(-1,0)
Out[8]=
1

In[9]:=
k[0,.707101] computes k(0,.707101)
Out[9]=
1.41418

In[10]:=
k[0,-.707101] computes k(0,-.707101)
Out[10]=
-1.41418

```

Thus, the maximum value is approximately 1.41418.

The minimum value is approximately -1.41418.

■ Multiple Integrals

Mathematica can compute multiple integrals. The command which computes the double integral

$$\int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} f(x,y) dy dx \text{ is:}$$

Integrate[$f[x,y]$, { x , x_{\min} , x_{\max} }, { y , y_{\min} , y_{\max} }]

Multiple integrals are numerically computed with the command

NIntegrate[$f[x,y]$, { x , x_{\min} , x_{\max} }, { y , y_{\min} , y_{\max} }].

The first variable given (in this case, x), corresponds to the outermost integral and integration with respect to this variable is done last. Also, the inner limits of integration (in this case, y_{\min} and y_{\max}) can be functions of the outermost variable. Limits on the outermost integral must always be constants.

- o When using Version 1.2, be sure to load the package `IntegralTables.m`; if using Version 2.0, loading `IntegralTables.m` is not necessary since *Mathematica* automatically loads the package.

Several examples are shown below :

□ Example:

The screenshot shows a Mathematica notebook window titled "MultipleIntegrals". It contains three input-output pairs, each with a callout box explaining the code and the integral being calculated.

Example 1:

```
In[2]:=
<<IntegralTables.m
Out[2]:=
Integrator`

In[15]:=
Integrate[x y^2, {y, 1, 2},
{x, 1-y, Sqrt[y]}]
Out[15]:=
163
---
120
```

Callout 1: *Be sure the package IntegralTables.m has been loaded before attempting to compute definite integrals.*

Callout 2: `Integrate[x y^2, {y, 1, 2}, {x, 1-y, Sqrt[y]}]`
calculates $\int_1^2 \int_{1-y}^{\sqrt{y}} xy^2 dx dy$.

Example 2:

```
In[16]:=
Integrate[y Sin[x]-x Sin[y],
{x, 0, Pi/6}, {y, 0, Pi/2}]
Out[16]:=
      2          2
Pi   Sqrt[3] Pi
-----
      9          16
```

Callout 3: `Integrate[y Sin[x]-x Sin[y], {x, 0, Pi/6}, {y, 0, Pi/2}]`
calculates $\int_0^{\pi/6} \int_0^{\pi/2} (y \sin(x) - x \sin(y)) dy dx$.

Example 3:

```
In[18]:=
Integrate[Exp[x] Sin[y],
{y, Pi/6, Pi/4}, {x, 0, Cos[y]}]
Out[18]:=
      Sqrt[2]   Sqrt[3]   Sqrt[2]/2
----- - E      +
      2          2
      Sqrt[3]/2
      E
```

Callout 4: `Integrate[Exp[x] Sin[y], {y, Pi/6, Pi/4}, {x, 0, Cos[y]}]`
calculates $\int_{\pi/6}^{\pi/4} \int_0^{\cos(y)} e^x \sin(y) dx dy$.

In cases in which the double integral cannot be computed exactly, the command `NIntegrate[f[x, y], {x, xmin, xmax}, {y, ymin, ymax}]` can be used to calculate a numerical approximation of the integral

$$\int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} f[x, y] dy dx.$$

□ Example:

Approximate the value of the double integral $\int_0^1 \int_0^1 \sin(e^{xy}) dy dx$.

MultipleIntegrals

In[23]:=

```
Integrate[Sin[Exp[x y]], {x, 0, 1}, {y, 0, 1}]
```

Out[23]=

$$\frac{-\pi}{2} (\infty) + \frac{-1}{2} \text{ExpIntegralE}[1, -1] + \frac{1}{2} \text{ExpIntegralE}[1, 1] + \int_0^1 \frac{\text{SinIntegral}[E^x]}{x} dx$$

In[24]:=

```
NIntegrate[Sin[Exp[x y]], {x, 0, 1}, {y, 0, 1}]
```

Out[24]=

0.917402

Mathematica cannot exactly compute the double integral $\int_0^1 \int_0^1 \sin(e^{xy}) dy dx$.

NIntegrate[Sin[Exp[x y]], {x, 0, 1}, {y, 0, 1}] approximates the double integral $\int_0^1 \int_0^1 \sin(e^{xy}) dy dx$.

However, **NIntegrate** does not always produce the quickest result. The following example illustrates what happens when an integral which can be computed exactly is attempted using **NIntegrate**.

Recall that the Error function $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-w^2} dw$ is given by **Erf[x]**.

MultipleIntegrals

In[26]:=
value=Integrate[Cos[x^2-y^2], {x, 0, Sqrt[Pi]}, {y, 0, Sqrt[Pi]}]

Out[26]=
 (Pi Erf[Sqrt[-I] Sqrt[Pi]]
 Erf[Sqrt[I] Sqrt[Pi]]) /
 (4 Sqrt[-I] Sqrt[I])

*Mathematica computes
 $\int_0^{\sqrt{\pi}} \int_0^{\sqrt{\pi}} \text{Cos}(x^2 - y^2) dy dx$
 in 25.9 seconds.*

Time: 25.90 second

In[27]:=
N[value] **N[value]** computes a numerical approximation
 of value.

Out[27]=
 1.24012

The command **NIntegrate[Cos[x^2-y^2], {x, 0, Sqrt[Pi]}, {y, 0, Sqrt[Pi]}**] does not produce an output for an approximation of the integral after ten minutes of computing on a Macintosh IIcx.

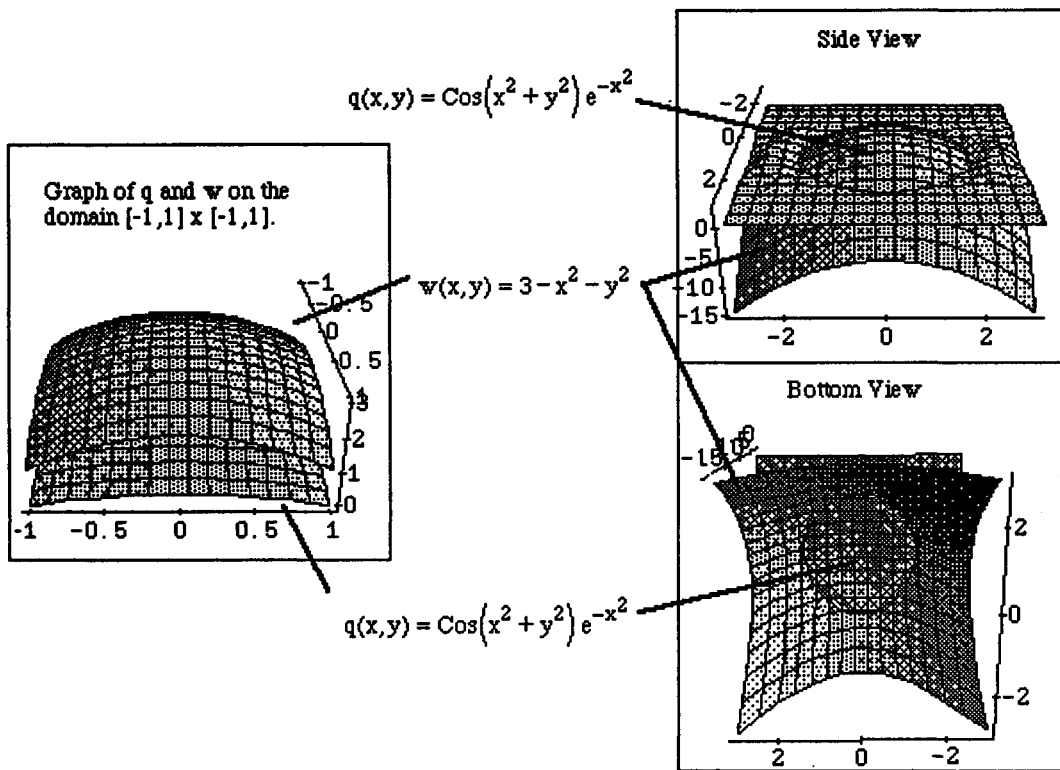
■ **Application: Volume**

□ **Example:**

Find the volume of the region between the graphs of

$$q(x, y) = \cos(x^2 + y^2) e^{-x^2} \quad \text{and} \quad w(x, y) = 3 - x^2 - y^2 \quad \text{on the domain } [-1, 1] \times [-1, 1].$$

The region can be viewed using *Mathematica's Plot3D* command.



The above graphs show that the region is bounded above by $w(x,y)$ and below by $q(x,y)$. Hence, the volume is determined as follows :

```

In[4]:=
<<IntegralTables.m
Clear[q,w]
q[x_,y_]=Exp[-x^2] Cos[y^2+x^2]
w[x_,y_]=3-x^2-y^2;

In[5]:=
volume=Integrate[w[x,y]-q[x,y],
{x,-1,1},{y,-1,1}];

In[6]:=
N[volume]

Out[6]:=
7.02707
    
```

Be sure to open the package IntegralTables.m before attempting to compute definite integrals.

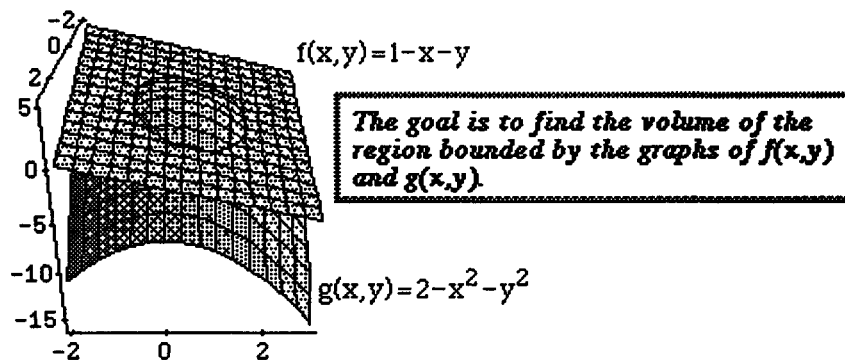
volume is the desired volume. Mathematica produces an exact value which is difficult to interpret hence a numerical approximation is obtained via N[volume].

A semi-colon placed at the end of a command prevents the output from being displayed.

□ Example:

Find the volume of the solid bounded by the graphs of $f(x,y)=1-x-y$ and $g(x,y)=2-x^2-y^2$.

The graph is used to determine that the region is bounded above by the paraboloid and below by the plane.



This problem is more difficult than the first example since the limits of integration must be determined. However, using **Solve** to find the values of x and y such that $f(x,y) = g(x,y)$, these limits are found easily. First, the equation $f(x,y) = g(x,y)$ is solved for y (in terms of x). To facilitate the use of these y -values, they are given the names $y1$ and $y2$.

Volumes
⌵

```

In[17]:=
  <<IntegralTables.m
  Clear[f,g]
  f[x_,y_]=1-x-y
  g[x_,y_]=2-x^2-y^2;

In[18]:=
  Solve[f[x,y]==g[x,y],y]

Out[18]=
  {{y ->
    
$$\frac{1 + \text{Sqrt}[1 - 4(-1 - x + x^2)]}{2}$$

  }, {y ->
    
$$\frac{1 - \text{Sqrt}[1 - 4(-1 - x + x^2)]}{2}}$$


In[20]:=
  y1=(1+Sqrt[1-4(-1-x+x^2)])/2

Out[20]=
    
$$\frac{1 + \text{Sqrt}[1 - 4(-1 - x + x^2)]}{2}$$


In[21]:=
  y2=(1-Sqrt[1-4(-1-x+x^2)])/2

Out[21]=
    
$$\frac{1 - \text{Sqrt}[1 - 4(-1 - x + x^2)]}{2}$$


```

*Be sure the package **IntegralTables.m** has been loaded before attempting to compute definite integrals.*

To locate the points (x,y) where $f(x,y)=g(x,y)$ we solve the equation $f(x,y)=g(x,y)$ for y (in terms of x)

To save typing, name
y1=(1+Sqrt[1-4(-1-x+x^2)])/2
and
y2=(1-Sqrt[1-4(-1-x+x^2)])/2

Next, the limits on the x-coordinate are determined :

```

In[19]:=
  Solve[1-4(-1-x+x^2)==0,x]
Out[19]=
  {{x -> (4 + 4 Sqrt[6])/8},
   {x -> (4 - 4 Sqrt[6])/8}}

```

Solve[1-4(-1-x+x²)==0,x]

solves the equation

$$1-4(-1-x+x^2)=0.$$

The values $\frac{4+4\sqrt{6}}{8}$

and $\frac{4-4\sqrt{6}}{8}$

are the upper and lower limits of integration.

Finally after the limits of integration have been determined, the volume is found :

```

Volumes
In[22]:=
volume=Integrate[g[x,y]-f[x,y],
{x,(4-4Sqrt[6])/8,(4+4Sqrt[6])/8},
{y,y2,y1}]
Out[22]=
      4 - 4 Sqrt[6]
      -1 + -----
              4
-9 ArcSin[-----]
      Sqrt[6]
----- +
      8
      4 + 4 Sqrt[6]
      -1 + -----
              4
      9 ArcSin[-----]
      Sqrt[6]
-----
      8

In[23]:=
N[volume]
Out[23]=
      3.53429

```

computes

$$\int_{(4-4\sqrt{6})/8}^{(4+4\sqrt{6})/8} \int_{y2}^{y1} (g(x,y)-f(x,y)) dy dx$$

*and names the result **volume**.*

N[volume] *yields a numerical approximation of **volume**.*

■ Series in More than One Variable

In the same manner as **Integrate**, **NIntegrate**, and **D** generalize to functions of more than one variable, the command **Series** also computes power series of functions of more than one variable.

The command **Series[f[x, y], {x, x0, n}, {y, y0, m}]** computes a power series expansion of **f[x, y]** about **y0** up to order **m** and then computes a power series expansion about **x0** up to order **n**.

The result of using the **Series** command always includes a remainder term and hence cannot be treated as a function. In order to remove the remainder term, use the command **Normal**.

□ Example:

Define $f(x, y) = \cos(x - \pi \sin(y))$. (i) Generate the first two terms of the power series for f about $y=1$; (ii) generate the first two terms of the power series of f about $y=1$ and then generate the first three terms of the power series about $x=0$; and (iii) generate the first three terms of the power series of f about $x=0$ and then generate the first two terms of the power series about $y=1$.

After removing the remainder term, graph and compare the results.

We begin by defining f and then computing the first two terms of the power series for f about $y=1$:

```

SeriesinTwoVariables
In[5]:=
  f[x_, y_]=Cos[x-Pi Sin[y]]
Out[5]=
  Cos[x - Pi Sin[y]]
In[7]:=
  sery=Series[f[x, y], {y, 1, 2}]
Out[7]=
  Cos[x - Pi Sin[1]] + Pi Cos[1] Sin[x - Pi Sin[1]] (-1 + y) +
    -(Pi Cos[1] Cos[x - Pi Sin[1]])
    (-----)
          2
    Pi Sin[1] Sin[x - Pi Sin[1]]          2          3
    (-----) (-1 + y) + O[-1 + y]
          2

```

SeriesinTwoVariables

In[1]:=
f[x_,y_]=Cos[x-Pi Sin[y-1]] *Defines f(x,y) = Cos(x - π Sin(y)).*

Out[1]=
 Cos[x + Pi Sin[1 - y]]

In[2]:=
serone=Series[f[x,y],{x,0,3},{y,1,2}] *generates the power series of f with respect to y then x.*

Out[2]=

$$1 - \frac{\text{Pi}^2 (-1 + y)^2}{2} + \text{O}[-1 + y]^3 +$$

$$(\text{Pi} (-1 + y) + \text{O}[-1 + y]^3) x +$$

$$(-\frac{1}{2}) + \frac{\text{Pi}^2 (-1 + y)^2}{4} + \text{O}[-1 + y]^3) x^2 + \text{O}[x]^3$$

In[3]:=
serxy=Normal[serone] *removes the remainder terms from serone the result is an expression in x and y which can be graphed.*

Out[3]=

$$1 - \frac{x^2}{2} + \text{Pi} x (-1 + y) + \left(\frac{-\text{Pi}^2}{2} + \frac{\text{Pi}^2 x^2}{4}\right) (-1 + y)^2$$

The command `seryx=Series[f[x,y],{y,1,2},{x,0,3}]/Normal` computes the series with respect to x then y and removes the remainder terms.

In[4]:=

`seryx=Series[f[x,y],{y,1,2},{x,0,3}]/Normal`

Out[4]=

$$\begin{aligned}
 & 1 + x^4 \left(\frac{1}{24} - \frac{\text{Pi}^2 (-1 + y)^2}{48} \right) + x^8 \left(\frac{1}{40320} - \frac{\text{Pi}^2 (-1 + y)^2}{80640} \right) + \\
 & x^6 \left(-\left(\frac{1}{720}\right) + \frac{\text{Pi}^2 (-1 + y)^2}{1440} \right) + x^2 \left(-\left(\frac{1}{2}\right) + \frac{\text{Pi}^2 (-1 + y)^2}{4} \right) + \\
 & \text{Pi} x (-1 + y) - \frac{\text{Pi} x^3 (-1 + y)}{6} + \frac{\text{Pi} x^5 (-1 + y)}{120} - \\
 & \frac{\text{Pi} x^7 (-1 + y)}{5040} - \frac{\text{Pi}^2 (-1 + y)^2}{2}
 \end{aligned}$$

We create the graphs with Version 2.0 to take advantage of Version 2.0's improved graphics features. In particular, we graph $f[x, y]$, $serxy$, and $seryx$ with `Plot3D` and then show all three simultaneously with the `Show` command. Finally, all four graphics objects are displayed in a single cell with the command `GraphicsArray`.

```
inf 9:=
  plotf=Plot3D[f[x,y],{x,0,2Pi},{y,0,2Pi},
    PlotPoints->25,Boxed->False,
    Ticks->{{0,Pi,2Pi},None,{-1,0,1}},
    DisplayFunction->Identity,
    Shading->False];

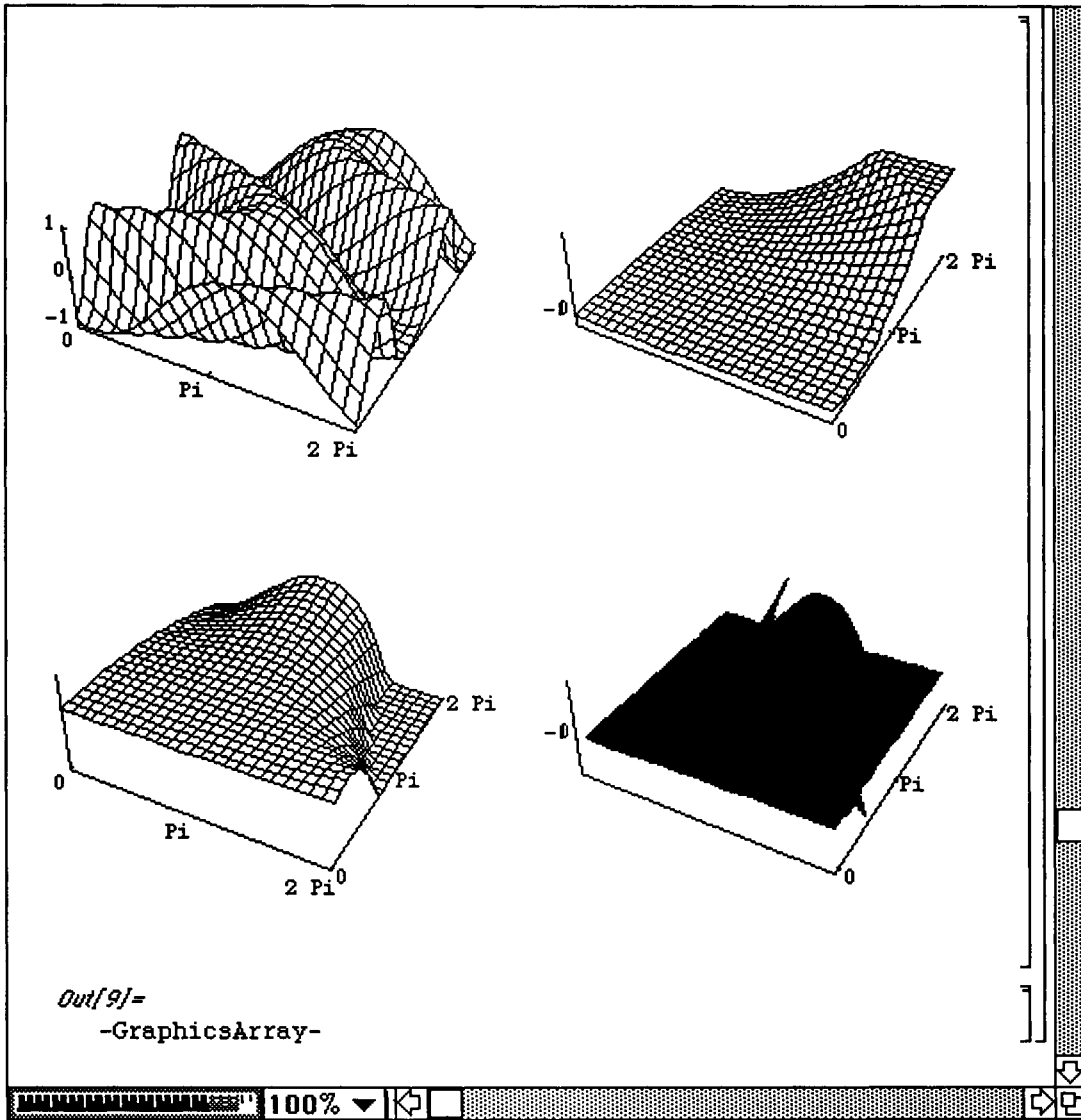
  plotxy=Plot3D[serxy,{x,0,2Pi},{y,0,2Pi},
    PlotPoints->25,Boxed->False,
    DisplayFunction->Identity,
    Ticks->{None,{0,Pi,2Pi},{-1,0,1}},
    Shading->False];

  plotyx=Plot3D[seryx,{x,0,2Pi},{y,0,2Pi},
    PlotPoints->25,Boxed->False,
    Ticks->{{0,Pi,2Pi},{0,Pi,2Pi},None},
    DisplayFunction->Identity,
    Shading->False];

  allthree=Show[plotxy,plotyx,plotf];

  Show[GraphicsArray[{{plotf,plotxy},
    {plotyx,allthree}}]]
```


The result of the final **Show** command is displayed below:



Chapter 4

Introduction to Lists and Tables

- Chapter 4 introduces elementary operations on lists and tables. Chapter 4 is a prerequisite for Chapter 5 which discusses nested lists and tables in detail. The examples used to illustrate the various commands in this chapter are taken from calculus, business, and engineering applications.
- Commands introduced and discussed in this chapter from Version 1.2 are:

Operations on Tables and Lists:

```

Table[function[index],{index,start,finish,step}]
TableForm[table]
table // TableForm
MatrixForm[table]
table // MatrixForm
Prime[positiveinteger]
table[[positiveinteger]]
Short[expression]
Length[list]
Last[list]
First[list]
Apply[function,list]
Map[function,list]
Release[table]
Fit[data,{linearlyindependentfunctionsofvariable},variable]
Sum[f[j],{j,jstart,jstop,jstep}]
//
/0

```

Other Operations:

```

Do[statement[i],{i,istart,istop,istep}]
Print[InputForm[expression]]
HermiteH[n,x]
LaguerreL[n,x]
Plus
Times
Random[type,{min,max}]

```

Graphics Operations on Lists:

```

ListPlot[list,options]
ParametricPlot[{x[t],y[t]},{t,tmin,tmax},options]

```

Options:

```

PlotJoined->True
Ticks->None
AspectRatio->positivenumber
PlotRange->All

```

- Commands introduced and discussed in this chapter from Version 2.0 include:

```

SymbolicSum[rationalfunction,{k,kmin,kmax}]
AccountingForm[number]

```

■ Applications in this chapter include:

□ Business:

Creating Business Tables:

Interest, Annuities, and Amortization

□ Calculus:

Calculating and Graphing Lists of Functions

Graphing Equations

Tangent Lines and Animations

□ Engineering:

Curve Fitting

Introduction to Fourier Series:

Calculating Fourier Series

The Heat Equation

4.1 Defining Lists

A **list** is a *Mathematica* object of the form

$\{\text{element}[1], \text{element}[2], \dots, \text{element}[n-1], \text{element}[n]\}$ where $\text{element}[i]$ is called the i th element of the list. Elements of a list are separated by commas. Notice that lists are **always** enclosed in "curly" brackets $\{\}$ and each element of a list may be (almost any) *Mathematica* object; even other lists. Since lists are *Mathematica* objects, they can be named. For easy reference, we will usually name lists.

Lists may be defined in a variety of ways. Lists may be completely typed in or they may be created by the **Table** command. For a function f with domain non-negative integers and a positive integer n , the command **Table** $[f[i], \{i, 0, n\}]$ creates the list $\{f[0], f[1], \dots, f[n-1], f[n]\}$.

□ Example:

The screenshot shows a Mathematica notebook window titled "Lists". It contains three input-output pairs:

```

In[27]:=
  list1={1, 3/2, 2, 5/2, 3, 7/2, 4}
Out[27]=
  3      5      7
  {1, -, 2, -, 3, -, 4}
  2      2      2

In[28]:=
  list2=Table[i^2+1, {i, 1, 5}]
Out[28]=
  {2, 5, 10, 17, 26}

In[29]:=
  list3=Table[{i, i^3-i}, {i, 1, 5}]
Out[29]=
  {{1, 0}, {2, 6}, {3, 24}, {4, 60}, {5, 120}}
  
```

On the right side of the notebook, there are three text annotations:

- list1** is the list consisting of the numbers 1, 3/2, 2, 5/2, 3, 7/2 and 4.
- Table** $[i^2+1, \{i, 1, 5\}]$ creates a list consisting of the values of i^2+1 for $i=1, 2, 3, 4,$ and 5 .
- list2** is the list of numbers 2, 5, 10, 17, 26.
- list3** is the list of ordered pairs consisting of $\{1, 0\}, \{2, 6\}, \{3, 24\}, \{4, 60\},$ and $\{5, 120\}$.

Mathematica will display a list, like other output, on successive lines which may sometimes be difficult to read or interpret. The commands **TableForm** and **MatrixForm** are used to display lists in traditional row/column form. In the following example, **list4** is the list of ordered triples $\{i, \text{Sqrt}[i] // \mathbf{N}, \text{Sin}[i] // \mathbf{N}\}$ for $i=1, 2, 3, 4, 5$. Notice that each ordered triple is a list. This will be discussed in more detail in Chapter 5.

□ Example:

The screenshot shows a Mathematica notebook with the following content:

```

In[30]:=
list4=Table[{i,Sqrt[i] // N,Sin[i] // N},
{i,1.5}]
Out[30]=
{{1. 1., 0.841471}, {2. 1.41421, 0.909297},
{3. 1.73205, 0.14112},
{4. 2., -0.756802},
{5. 2.23607, -0.958924}}
In[31]:=
TableForm[list4]
Out[31]//TableForm=
1      1.      0.841471
2      1.41421  0.909297
3      1.73205  0.14112
4      2.      -0.756802
5      2.23607 -0.958924

```

list4
is the list of ordered triples
consisting of {1,1,841},
{2,1.414,909},
{3,1.732,141},
{4,2,-.756}, and
{5,2.236,-.958}.

TableForm[list4]
represents list4
in tabular (or matrix) form.
The same result could have
been obtained with
MatrixForm[list4].

As indicated above, elements of lists can be numbers, ordered pairs, functions, and even other lists. For example, *Mathematica* has built-in definitions of many commonly used special functions. Consequently, lists of special functions can be quickly created.

□ Example:

The Hermite polynomials, $H_n(x)$, satisfy the differential equation $y'' - 2x y' + 2ny = 0$.

The built-in command `HermiteH[n,x]` yields the Hermite polynomial $H_n(x)$.

Create a table of the first five Hermite polynomials and name the resulting list `hermitetable`.

The screenshot shows a Mathematica notebook window titled "ListsofFunction". It contains the following input and output cells:

```

In[6]:=
  hermitetable=Table[HermiteH[n,x],{n,1,5}]
Out[6]=
  {2 x, -2 + 4 x2, -12 x + 8 x3,
  12 - 48 x2 + 16 x4, 120 x3 - 160 x5 + 32 x5}
In[7]:=
  TableForm[hermitetable]
Out[7]//TableForm=
  2 x
  -2 + 4 x2
  -12 x + 8 x3
  12 - 48 x2 + 16 x4
  120 x3 - 160 x5 + 32 x5
  
```

Two callout boxes provide additional information:

- The first callout box, located to the right of the first input cell, states: "creates a table of polynomials HermiteH[1,x], ..., HermiteH[5,x]. The resulting list is named hermitetable."
- The second callout box, located to the right of the second input cell, states: "displays hermitetable as a column."

In fact, lists can be evaluated at certain numbers so that lists of numbers are created:

```
In[18]:=
```

```
hermitetable /. x->1
```

*calculates the value of
by replacing x by 1.*

```
Out[18]=
```

```
{2., 2., -4., -20., -8.}
```

```
In[19]:=
```

```
values=Table[
  N[hermitetable /. x->j],
  {j,0,2,2/10}];
TableForm[values]
```

*computes a table of approximations of the value
of hermitetable
by replacing x by j for j=0, 2/10, 4/10, ... , 18/10, 2.
The resulting list of numbers is named*

```
Out[19]//TableForm=
```

0.	-2.	0.	12.	0.
0.4	-1.84	-2.336	10.1056	22.7302
0.8	-1.36	-4.288	4.7296	38.0877
1.2	-0.56	-5.472	-3.2064	39.9283
1.6	0.56	-5.504	-12.1664	24.5658
2.	2.	-4.	-20.	-8.
2.4	3.76	-0.576	-23.9424	-52.8538
2.8	5.84	5.152	-20.6144	-98.9363
3.2	8.24	13.568	-6.0224	-127.816
3.6	10.96	25.056	24.4416	-112.458
4.	14.	40.	76.	-16.

*The semi-colon at the end
of the command suppresses
the resulting output. Instead
values is displayed
in a tabular form.*

Moreover, operations, such as differentiation, can be performed lists:

```
In[25]:=
  D[hermitetable,x] // TableForm
Out[25]//TableForm=
```

$$\begin{array}{r}
 2 \\
 8 x \\
 -12 + 24 x^2 \\
 -96 x + 64 x^3 \\
 120 - 480 x^2 + 160 x^4
 \end{array}$$

computes the derivative (with respect to x) of each element of hermitetable and expresses the result in a tabular form.

■ 4.2 Operations on Lists

■ Extracting Elements of Lists

Individual elements of lists are obtained using **double-square brackets**. For example if **table** is a list, then **table[[2]]** is the second element of the list **table**. The **j**th element of **table** is **table[[j]]**.

□ Example:

The *Mathematica* function **Prime** can be used to calculate prime numbers. **Prime[1]** yields 2; **Prime[2]** yields 3; and, in general, **Prime[k]** yields the **k**th prime number.

Make a table of the first fifteen prime numbers. What is the third prime number? the thirteenth prime number?

The screenshot shows a Mathematica notebook window titled "ExtractingLists". The notebook contains the following code and output:

```

In[2]:=
  primelist=Table[Prime[j],{j,1,15}]
Out[2]=
  {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
  41, 43, 47}
In[3]:=
  primelist[[3]]
Out[3]=
  5
In[4]:=
  primelist[[13]]
Out[4]=
  41

```

A callout box with a dotted border contains the text: "Double-Square Brackets '[[]]' are ALWAYS used to extract elements of lists."

Annotations on the right side of the notebook explain the output:

- primelist** is a list of the first 15 prime numbers.
- primelist[[3]]** gives the third element of **primelist**.
- primelist[[13]]** gives the thirteenth element of **primelist**.

■ Graphing Lists and Lists of Functions

If **list={a[1], a[2], ... , a[n]}** is a list of numbers, **ListPlot[list]** plots the points **{1,a[1]}, {2,a[2]}, {3,a[3]}, ... , {n,a[n]}**. In general, the command **ListPlot** has the same options as the command **Plot**.

Sometimes it is desirable to suppress the output of lists; particularly when long lists are used. In general, a semi-colon ";" placed at the end of a command suppresses the resulting output.

□ Example:

The following example demonstrates how `ListPlot[list]` is used to plot the points $\{1, f[1]\}$, $\{2, f[2]\}$, $\{3, f[3]\}$, ..., $\{1000, f[1000]\}$ where $f[x] = \text{Sin}[x]/N$.

Let $f(x) = \text{Sin}(x)$. First, make a table of the values $\text{Sin}(1), \text{Sin}(2), \dots, \text{Sin}(10)$; then make a table of the values $\text{Sin}(i)$ for $i=1, 2, \dots, 1000$. Graph the points $(i, \text{Sin}(i))$ for $i=1, \dots, 1000$.

OperationsOnLists

```

In[1]:=
Clear[f]
f[x_]:=Sin[x] // N

In[2]:=
table1=Table[f[x].{x,1,10}]

```

table1=Table[f[x].{x,1,10}] is a list consisting of the values of f(x) for x=1,2,3,...,10.

```

Out[2]=
{0.841471, 0.909297, 0.141112,
 -0.756802, -0.958924, -0.279415,
 0.656987, 0.989358, 0.412118,
 -0.544021}

In[3]:=
table2=Table[Sin[x].{x,1,1000}];

```

table2 is a list consisting of the values of f(x) for x=1,2,...,1000. Since this list consists of 1000 elements, we choose to not display the list.

```

In[4]:=
ListPlot[table2]

```

ListPlot[table2] plots the points {x,Sin[x]} for x=1,2,3,...,1000.

The plot shows a dense scatter of points forming a sine wave. The x-axis is labeled from 0 to 1000 in increments of 200. The y-axis is labeled from -1 to 1 in increments of 0.5.

Both tables of numbers and tables of functions can be graphed. In the following example, we graph the elements of `hermitetable` (created above) on the interval $[-3,3]$.

□ Example:

Use *Mathematica* to graph the first five Hermite polynomials on the interval $[-3,3]$.

Notice that within the **Plot** command, **hermitetable** is enclosed by the command **Release**.

Release[hermitetable] allows the elements of **hermitetable** to be evaluated for the values of x on $[-3,3]$ instead of recreating the table for each value of x .

- Version 1.2 (or earlier) users must always enclose the table to be graphed by **Release**. In Version 2.0, the command **Release** has been replaced by the command **Evaluate**. Hence, when graphing tables of functions with Version 2.0, be sure to enclose the table by **Evaluate**.

```
In[8]:=
grays=Table[GrayLevel[j/10],{j,1,5}]
Out[8]=
      1          1          3
{GrayLevel[--], GrayLevel[-], GrayLevel[--],
 10          5          10

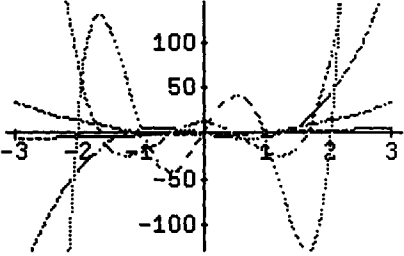
      2          1
GrayLevel[-], GrayLevel[-]}
      5          2

In[11]:=
Plot[Release[hermitetable],{x,-3,3},
PlotStyle->grays]
Out[11]=
-Graphics-
```

grays is a table of various graylevels that will be used to graph the table of functions hermitetable.

Each element of hermitetable is graphed on the interval[-3,3] according to the graylevel specified by the list grays.

When using versions prior to Version 2.0, be sure to enclose the name of the list by Release. When using Version 2.0, be sure to enclose the name of the list by Evaluate.



In the previous examples, the domain of the functions has been the set of natural numbers. This does not have to be the case, however. The command

Table[f[x], {x,xmin, xmax, xstep}]

creates a list by evaluating f at values of x from $x = x_{\min}$ to $x = x_{\max}$ using a stepsize of x_{step} . If n is a positive integer, the command **Table[f[x], {x, xmin, xmax, (xmax-xmin)/n}]** creates the list, containing the $n+1$ elements:

$$\left\{ f[x_{\min}], f\left[x_{\min} + \frac{x_{\max} - x_{\min}}{n}\right], f\left[x_{\min} + 2\frac{x_{\max} - x_{\min}}{n}\right], \dots, f[x_{\max}] \right\}.$$

When dealing with a long *Mathematica* object expression, another useful *Mathematica* command is **Short[expression]**. This command produces an abbreviated, one-line output of **expression**. If **list** is a table, the command **Short[list]** produces a one-line output of **list**. If **n** is a positive integer greater than one, **Short[list, n]** produces an abbreviated **n**-line output of **list**. This abbreviated list includes an element of the form `<< n >>` which indicates the number of elements of **list** that are omitted in the abbreviated output.

■ Evaluation of Lists by Functions

Another helpful command is **Map[f, list]**.

This command creates a list consisting of elements obtained by evaluating **f** for each element of **list**, provided that each member of **list** is an element of the domain of **f**.

■ Notice:

To avoid errors, be sure to check that each element of **list** is in the domain of **f** prior to executing the command **Map[f, list]**.

If **list** is a table of **n** numbers and each element of **list** is in the domain of **f**, recall that **list[[i]]** denotes the **i**th element of **list**. The command **Map[f, list]** produces the same list as the command **Table[f[list[[i]]], {i, 1, n}]**. Using **Map** in conjunction with **ListPlot** yields an alternative approach to graphing functions.

□ Example:

Let $g(x) = \cos(x) - 2\sin(x)$. First create a table of the values from π to 5π in steps of $\frac{4\pi}{40}$. Name the list `table3`. This is the same as creating the table of values

$$\left\{ \pi, \frac{44\pi}{40}, \dots, \pi + n\frac{4\pi}{40}, \dots, 5\pi \right\} = \left\{ \pi + j\frac{4\pi}{40} : j = 0, 1, 2, \dots, 40 \right\}.$$

Evaluate g for each element in `table3`.

The screenshot shows a Mathematica notebook window titled "OperationsOnLists". The notebook contains the following code and output:

```

In[5]:=
Clear[g]
g[x_]:=Cos[x]-2 Sin[x]

In[6]:=
table3=Table[j,{j,Pi,5Pi,4Pi/40}];

In[7]:=
Short[table3]

Out[7]::Short=
  11 Pi
{Pi, -----, <<38>>, 5 Pi}
  10

In[8]:=
table4=Map[g,table3];

In[9]:=
Short[table4]

Out[9]::Short=
{-1, <<39>>, -1}

In[10]:=
ListPlot[table4]

Out[10]=
-Graphics-

```

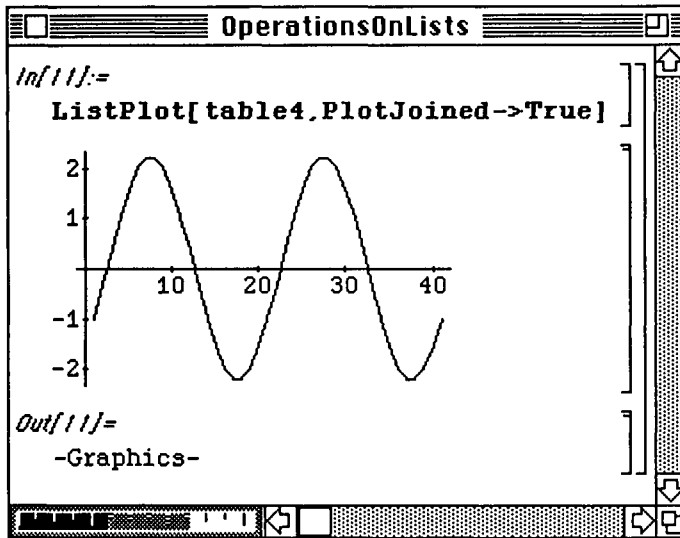
Annotations on the right side of the notebook explain the code:

- Begin by clearing prior definitions of g and defining $g(x) = \cos(x) - 2\sin(x)$.** (Refers to In[5])
- table3 consists of the values $\pi + n\frac{4\pi}{40}$ for $n = 1, 2, 3, \dots, 40$.** (Refers to In[6])
- Short[table3]** prints a portion of table3 that consists of one line. In this case it has yielded the first few elements and the last few elements. (Refers to In[7])
- Map[g, table3]** creates a table of the values $g(\text{table3}[[i]])$ for $i = 1, 2, 3, \dots, 40$. The table is named table4. (Refers to In[8])
- Short[table4]** prints a portion of table4 that consists of one line. (Refers to In[9])
- ListPlot[table4]** graphs the points $(i, \text{table4}[[i]])$ for $i = 1, 2, 3, \dots, 40$. (Refers to In[10])

A callout box with a dotted border contains the text: "Remember that a semi-colon placed at the end of a command prevents the output from being shown." This refers to the semi-colons at the end of In[5], In[6], In[8], and In[9].

The plot shows a discrete plot of the function $g(x) = \cos(x) - 2\sin(x)$ over the interval $[\pi, 5\pi]$. The x-axis is labeled from 0 to 40, and the y-axis is labeled from -2 to 2. The plot shows a periodic wave with 40 data points.

In general, **ListPlot** and **Plot** share many of the same options. However, since the **ListPlot** command graphs a set of points, and is **NOT** connected, a connected graph is obtained by using the **PlotJoined** option:



plots the points (i,table4[[i]]) for i=1, 2, 3, ..., 40 and then connects the points with line segments.

Naturally, lists of functions can be evaluated in the same manner as above:

□ **Example:**

The **Laguerre polynomials** are defined recursively by the relationship $L_0(x)=1$ and

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x}).$$

For each n , $L_n(x)$ satisfies the differential equation $xy'' + (1-x)y' + ny = 0$.

$L_n(x)$ is computed with the built-in function **LaguerreL [n,x]**.

For $n=1,2,3,4$, use *Mathematica* to verify that $L_n(x)$ satisfies $xy'' + (1-x)y' + ny = 0$.

First a table of the first four Laguerre polynomials is created and then a function **f** is defined as follows:

For a given ordered pair $\{n, \text{poly}\}$, **f[{n, poly}]** returns an ordered quadruple given by

```
{xD[poly, {x, 2}],
 (1-x)D[poly, x],
 n*poly, D[poly,
 {x, 2}]+(1-x)D[poly, x]n poly
 }.
```

Remember that `D[poly, {x, n}]` computes the n th derivative of `poly` with respect to `x`.

lagtable
is a list of ordered pairs.
The first element of each ordered pair is a positive integer i , the second element of each ordered pair is the i th Laguerre polynomial.

`lagtable` corresponds to `lagtable[[3]]`

`lagtable` corresponds to `lagtable[[4]]`

For a given ordered pair $\{n, \text{poly}\}$, `f[{n, poly}]` returns an ordered quadruple.

```

In[32]:=
lagtable=Table[{n,LaguerreL[n,x]},{n,1,4}]
Out[32]=
{{1, 1 - x}, {2,  $\frac{2 - 4x + x^2}{2}$ },
{3,  $\frac{6 - 18x + 9x^2 - x^3}{6}$ },
{4,  $\frac{24 - 96x + 72x^2 - 16x^3 + x^4}{24}$ }}
In[33]:=
f[{n_,poly_}] :=
{x D[poly,{x,2}],(1-x)D[poly,x],n poly,
Simplify[
x D[poly,{x,2}]+(1-x)D[poly,x]+n poly]}

```

`Map[f, lagtable]` evaluates `f` for each element of `lagtable`. The same result would have been obtained with the command `Table[f[lagtable[[i]], {i, 1, 4}]`.

Note that the fourth component in each ordered quadruple below is zero. Hence, each member of `lagtable` is a solution.

```

In[34]:=
Map[f, lagtable]
Out[34]:=
{{(0, -(1 - x), 1 - x, 0), {x,  $\frac{(1 - x)(-4 + 2x)}{2}$ ,  $2 - 4x + x^2$ , 0}},
{ $\frac{(18 - 6x)x}{6}$ ,  $\frac{(1 - x)(-18 + 18x - 3x^2)}{6}$ ,  $\frac{6 - 18x + 9x^2 - x^3}{2}$ , 0}},
{ $\frac{x(144 - 96x + 12x^2)}{24}$ ,  $\frac{(1 - x)(-96 + 144x - 48x^2 + 4x^3)}{24}$ ,
 $\frac{24 - 96x + 72x^2 - 16x^3 + x^4}{6}$ , 0}}

```

■ Other List Operations

A specific operation can be applied to the elements of a list through the command `Apply[operation, list]`. Of course, in order to use this command, the given `operation` must be defined for the elements of `list`.

For example, if `numbers` is a list of real numbers, then the command `Apply[Plus, numbers]` adds together all the elements of `numbers`.

Some other *Mathematica* commands used with lists are:

`Length[list]`, which gives the number of elements in `list`;

`First[list]`, which gives the first element of `list`; and

`Last[list]`, which gives the last member of `list`.

Several examples of these commands are shown below.

Also notice that the definition of a vector-valued function $f: \mathcal{R} \rightarrow \mathcal{R}^n$

which maps the real numbers to n -space can be made using a list. This is done below in the following manner :

$f[\mathbf{x}_-]:= \{f_1(x), f_2(x), f_3(x), \dots, f_n(x)\}$ where $f_k(x): \mathcal{R} \rightarrow \mathcal{R}$ for $1 \leq k \leq n$.

□ Example:

Define $f(x) = \left\{ \left(\frac{x}{100} \right)^2, 36 - \left(\frac{x}{100} \right)^2 \right\}$. Create a table of the numbers $10^2, 20^2, \dots, 50^2,$

and 60^2 . Evaluate f for each number in the table. Name the resulting table **list2**.

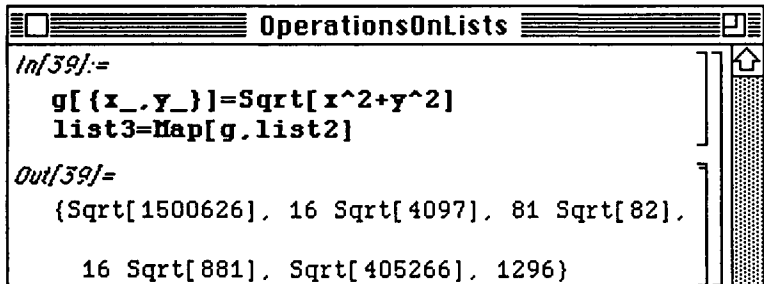
OperationsOnLists	
<code>In[32]:=</code>	
<code>Clear[list, list2, list3, g, f]</code>	
<code>list=Table[j^2, {j, 10, 60, 10}]</code>	
<code>Out[32]=</code>	<code>list</code> is the list of numbers $10^2=100, 20^2=400, 30^2=900$ $40^2=1600, 50^2=2500,$ and $60^2=3600.$
<code>{100, 400, 900, 1600, 2500, 3600}</code>	
<code>In[33]:=</code>	
<code>Apply[Times, list]</code>	<code>Apply[Times, list]</code> multiplies the elements of <code>list</code> together.
<code>Out[33]=</code>	
<code>5184000000000000000</code>	
<code>In[34]:=</code>	
<code>f[x_]={(x/100)^2, (36-(x/100))^2}</code>	<code>Map[f, list]</code> computes <code>f[list[[i]]]</code> for $i=1, 2, 3, 4, 5,$ and $6.$ The resulting list is named <code>list2</code> .
<code>list2=Map[f, list]</code>	
<code>Out[34]=</code>	
<code>{{1, 1225}, {16, 1024}, {81, 729},</code> <code>{256, 400}, {625, 121}, {1296, 0}}</code>	
<code>In[35]:=</code>	
<code>Length[list2]</code>	<code>Length[list2]</code> gives the number of elements in <code>list2</code> .
<code>Out[35]=</code>	
<code>6</code>	
<code>In[36]:=</code>	
<code>First[list2]</code>	<code>First[list2]</code> gives the first element of <code>list2</code> .
<code>Out[36]=</code>	
<code>{1, 1225}</code>	
<code>In[37]:=</code>	
<code>Last[list2]</code>	<code>Last[list2]</code> gives the last element of <code>list2</code> .
<code>Out[37]=</code>	
<code>{1296, 0}</code>	
<code>In[38]:=</code>	
<code>Apply[Plus, list2]</code>	<code>Apply[Plus, list2]</code> computes the sum of the first coordinates and second coordinates of <code>list2</code> .
<code>Out[38]=</code>	
<code>{2275, 3499}</code>	

In the following example, `Map` is used with the function $g: \mathcal{R}^2 \rightarrow \mathcal{R}$ defined via $g(x,y) = \sqrt{x^2 + y^2}$.

□ **Example:**

Evaluate g for each element of `list2`; call the resulting table `list3`. Add up the elements of `list3`; multiply together the elements of `list3`.

As has been the case with other examples, a numerical approximation of each member of the list is obtained using `//N`. Otherwise, exact values are given.



```

In[39]:=
  g[{x_, y_}] = Sqrt[x^2 + y^2]
  list3 = Map[g, list2]

Out[39]=
  {Sqrt[1500626], 16 Sqrt[4097], 81 Sqrt[82],
   16 Sqrt[881], Sqrt[405266], 1296}

```

Map[g, list2]
computes
g[list2[[i]]
for i = 1, 2, 3, 4, 5, 6.

We use the **Apply** command to compute the desired sum and product. *Mathematica* gives exact results unless otherwise requested:

<code>In[40]:=</code>	<code>Apply[Plus,list3]</code>	<code>Apply[Plus,list3]</code> <i>computes the exact sum of the elements of list3.</i>
<code>Out[40]=</code>	1296 + 81 Sqrt[82] + 16 Sqrt[881] + 16 Sqrt[4097] + Sqrt[405266] + Sqrt[1500626]	
<code>In[41]:=</code>	<code>Apply[Plus,list3] // N</code>	<code>Apply[Plus,list3] // N</code> <i>computes an approximation of the sum of the elements of list3.</i>
<code>Out[41]=</code>	5390.12	
<code>In[42]:=</code>	<code>Apply[Times,list3]</code>	<code>Apply[Times,list3]</code> <i>computes the exact product of the elements of list3.</i>
<code>Out[42]=</code>	26873856 Sqrt[82] Sqrt[881] Sqrt[4097] Sqrt[405266] Sqrt[1500626]	
<code>In[43]:=</code>	<code>Apply[Times,list3] // N</code>	<code>Apply[Times,list3] // N</code> <i>computes an approximation of the product of the elements of list3.</i>
<code>Out[43]=</code>	3.60549 10 ¹⁷	

■ Alternative Way to Evaluate Lists by Functions

□ Example:

A table consisting of ten random real numbers on the interval $\{0, 5\}$ is found with

`Table[Random[Real, {0, 5}], {10}]` and is called `t1`. A function `g[x]=Mod[x, 1]` is then defined; hence, `g` is merely x Modulo 1. In the same manner as above, the command `Map[g,t1]` evaluates `g` at each element in `t1`. However, the same result is obtained with `t1/g` and `g/@t1` as illustrated below:

- Note that since the command `Random` is used, if you enter the following sequence of calculations, `t1` will differ each time.

The screenshot shows a Mathematica notebook window titled "UsingOutput". The notebook contains several input-output pairs and explanatory text. The input cells are marked with "In[...]:=" and the output cells with "Out[...]=".

Input 115: `t1=Table[Random[Real,{0,5}],{10}]`

Output 115: `{0.769002, 2.85436, 4.88951, 1.6913, 4.0376, 0.23177, 3.79822, 2.88585, 1.25087, 3.53513}`

t1 is a table of ten "random" real numbers between 0 and 5.

Input 116: `g[x_]:=Mod[x,1]`

Defines g(x) to be x Modulo 1.

Input 117: `Map[g,t1]`

Output 117: `{0.769002, 0.854362, 0.889511, 0.691299, 0.0375963, 0.23177, 0.798223, 0.885855, 0.250874, 0.535125}`

Map[g,t1] computes g[t1[[i]]] for i=1,2,...,10.

Input 118: `t1 // g`

Output 118: `{0.769002, 0.854362, 0.889511, 0.691299, 0.0375963, 0.23177, 0.798223, 0.885855, 0.250874, 0.535125}`

t1 // g and g /@ t1 produce the same results as Map[g,t1].

Input 119: `g /@ t1`

Output 119: `{0.769002, 0.854362, 0.889511, 0.691299, 0.0375963, 0.23177, 0.798223, 0.885855, 0.250874, 0.535125}`

□ Example:

The sum of the squares of the first 100 positive integers is computed by several different methods below. First, table `t2` of the squares of the first 100 positive integers is created. The commands `Sum[i^2, {i, 1, 100}]`, `Apply[Plus, t2]`, and `Plus @@ t2` all achieve the correct sum of 338350.

```

UsingOutput

In[120]:=
  t2=Table[i^2, {i, 1, 100}];
  Sum[i^2, {i, 1, 100}]

Out[120]=
  338350

In[121]:=
  Apply[Plus, t2]

Out[121]=
  338350

In[122]:=
  Plus @@ t2

Out[122]=
  338350

```

t2 is the table {1, 4, 9, ..., 10000}.
`Sum[i^2, {i, 1, 100}]` computes

$$\sum_{i=1}^{100} i^2 = 1 + 4 + 9 + \dots + 10000.$$

`Apply[Plus, t2]` adds together
the elements of `t2`.

`Plus @@ t2` produces the exact
same result.

■ 4.3 Applications

■ **Application:** Interest, Annuities, and Amortization

The use of lists and tables are quite useful in economic applications which deal with interest rates, annuities, and amortization. *Mathematica* is, therefore, of great use in these types of problems through its ability to show the results of problems in tabular form. Also, if a change is made in the problem, *Mathematica* can easily recompute the results.

A common problem in economics is the determination of the amount of interest earned from an investment. Consider the following: If P dollars are invested for t years at an annual interest rate of $r\%$ compounded m times per

year, the compound amount $A(t)$ at time t is given by:
$$A(t) = P \left(1 + \frac{r}{m} \right)^{mt} .$$

A specific example is shown below where the amount of money accrued at time t represents the sum of the original investment and the amount of interest earned on that investment at time t .

□ Example:

Suppose \$12,500 is invested at an annual rate of 7% compounded daily. How much money has accumulated at the end of each five year period for $t = 5, 10, 15, 20, 25, 30$?

Interest

```
In[17]:=
Clear[ac, interest]
ac[t_]=12500 (1+.07/365)^(365 t)
interest[t_]=ac[t]-12500;
```

ac[t] gives the total value of the investment at the end of t years.

interest[t] yields the total amount of interest earned at the end of t years.

```
In[18]:=
Table[{t, ac[t]},
{t, 0, 30, 5}] // TableForm
```

Table[{t, ac[t]}, {t, 0, 30, 5}] // TableForm produces the table of ordered pairs {t, ac[t]} for t=0, 5, ..., 30 and then presents the final output in TableForm.

```
Out[18]//TableForm=
```

0	12500
5	17737.7
10	25170.2
15	35717.
20	50683.2
25	71920.5
30	102057.

The total value of the investment is given for t=0, 5, 10, 15, 20, 25, and 30 (years).

```
In[19]:=
Table[{t, ac[t], interest[t]},
{t, 0, 30, 5}] // TableForm
```

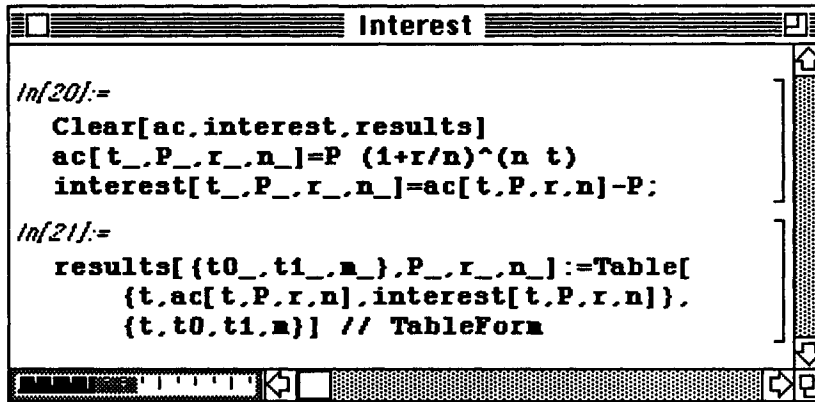
The total value of the investment is given in the first column and the total amount of the interest earned is given in the second column for t=0, 5, 10, 15, 20, 25, and 30.

```
Out[19]//TableForm=
```

0	12500	0
5	17737.7	5237.75
10	25170.2	12670.2
15	35717.	23217.
20	50683.2	38183.2
25	71920.5	59420.5
30	102057.	89556.6

Table[{t, ac[t], interest[t]}, {t, 0, 30, 5}] // TableForm produces the table of ordered triples {t, ac[t], interest[t]} for t=0, 5, ..., 30 and then presents the final output in TableForm.

The problem can be redefined for arbitrary values of t , P , r , and n as follows :



```

In[20]:=
Clear[ac, interest, results]
ac[t_, P_, r_, n_] = P (1 + r/n)^(n t)
interest[t_, P_, r_, n_] = ac[t, P, r, n] - P;

In[21]:=
results[{t0_, t1_, m_}, P_, r_, n_] := Table[
  {t, ac[t, P, r, n], interest[t, P, r, n]},
  {t, t0, t1, m}] // TableForm

```

Notice that the previous functions can be generalized for arbitrary t , P , r , and n .

*Notice that the user-defined function **results** combines several user-defined functions.*

Hence, any problem of this type can be worked using the functions defined above.

□ **Example:**

For example, suppose \$10,000 is invested at an interest rate of 12% compounded daily. Create a table consisting of the total value of the investment and the interest earned at the end of 0, 5, 10, 15, 20, and 25 years.

In this case, we use the function **results** defined above. Here, $t_0=0$, $t_1=25$, $m=5$, $P=10000$, $r=.12$, and $n=365$:

Notice that if the conditions are changed to $t_0=0$, $t_1=30$, $m=10$, $P=15000$, $r=.15$, and $n=365$, the desired table can be quickly calculated:

Interest

In[22]:= `results[{0, 25, 5}, 10000, .12, 365]`

Out[22]//TableForm=

0	10000	0
5	18219.4	8219.39
10	33194.6	23194.6
15	60478.6	50478.6
20	110188.	100188.
25	200756.	190756.

gives the total value and total interest of an investment of \$10,000 invested at 12% annually compounded daily for 0, 5, 10, 15, 20, and 25 years.

In[23]:= `results[{0, 30, 10}, 15000, .15, 365]`

Out[23]//TableForm=

0	15000	0
10	67204.6	52204.6
20	301097.	286097.
30	1.34901 10 ⁶	1.33401 10 ⁶

gives the total value and total interest of an investment of \$15,000 invested at 15% annually compounded daily for 0, 10, 20, and 30 years.

The problem of calculating the interest earned on an investment is altered if the interest is compounded continuously. The formula used in this case is as follows :

If P dollars are invested for t years at an annual interest rate of $r\%$ compounded continuously, the compound amount

$A(t)$ at time t is given by: $A(t) = Pe^{rt}$.

□ Example: (Future Value)

Consider the following :

If R dollars are deposited at the end of each period for n periods in an annuity hat earns interest at a rate

of j per period, the future value of the annuity is given by: $S_{\text{future}} = R \left[\frac{(1+j)^n - 1}{j} \right]$

A function which calculates the future value of the annuity and several examples using this function are given below :

Interest

```
In[7]:=
future[r_,j_,n_]=r (((1+j)^n-1)/j)
Out[7]=
      n
(-1 + (1 + j) ) r
-----
      j

In[8]:=
future[250.,.07/12,5 12]
Out[8]=
17898.2

In[9]:=
Table[{t, future[150.,.08/12,12 t]},
{t,1,25,4}] // TableForm
Out[9]//TableForm=
1      1867.49
5      11021.5
9      23614.4
13     40938.1
17     64769.6
21     97553.8
25     142654.
```

future[r, j, n]
computes $r \left[\frac{(1+j)^n - 1}{j} \right]$.

future[250., .07/12, 5 12]
computes the future value of the annuity where \$250 is deposited at the end of each month for 60 months at a rate of 7/12 % per month.

Table[{t, future[150., .08/12, 12 t]}, {t, 1, 25, 4}] // TableForm
makes a table of the future values of the annuity where \$150 is deposited at the end of each month for 12 t months at a rate of 8/12 % per month for t=1, 5, 9, 13, ..., 21, 25. The result is expressed in TableForm.

In this table, the first column corresponds to time (in years) and the second column corresponds to the future value of the annuity.

□ Example: (Annuity Due)

Another type of annuity is as follows :

If R dollars are deposited at the beginning of each period for n periods in an annuity with interest that earns

interest at a rate of j per period, the annuity due is given by: $S_{\text{due}} = R \left[\frac{(1+j)^{n+1} - 1}{j} \right] - R$.

Again, the function to determine the amount due is defined below with accompanying examples.

Interest

```
In[62]:=
due[r_, j_, n_] = r(((1+j)^(n+1)-1)/j) - r
Out[62]=
-r +  $\frac{(-1 + (1 + j)^{1 + n}) r}{j}$ 

In[63]:=
due[500, .12/12, 3 12]
Out[63]=
21753.8

In[64]:=
Table[{100 k, due[100 k, .09/12, 10 12]}, {k, 1, 10}] // TableForm
Out[64]:=TableForm=
100      19496.6
200      38993.1
300      58489.7
400      77986.3
500      97482.8
600      116979.
700      136476.
800      155973.
900      175469.
1000     194966.
```

due[r, j, n] computes
 $r \left[\frac{(1+j)^{n+1} - 1}{j} \right] - r$.

due[500, .12/12, 3 12] calculates the annuity due of \$500 deposited at the beginning of each month at an annual rate of 12% compounded monthly for three years.

calculates the annuity due of \$100k deposited at the beginning of each month at an annual rate of 9% compounded monthly for 10 years for k=1, 2, 3, ..., 10. The result is expressed in TableForm.

Notice that the first column corresponds to the amount deposited each month at an annual rate of 9% compounded monthly and the second column corresponds to the value of the annuity.

□ Example:

The following table compares the annuity due on a \$100 k monthly investment at an annual rate of 8% compounded monthly for t= 5, 10,15, 20; and k = 1, 2, 3, 4, 5. This type of table can prove to quite useful in the analysis of investments. (Note that the values of k and t were later added to the table.)

This yields a two-dimensional table that gives the annuity due of \$100 k deposited monthly at an annual rate of 8% compounded monthly for t years where k= 1, 2, 3, 4, and 5; t= 5, 10, 15, and 20.

For example, the annuity due of \$300 deposited monthly at an annual rate of 8% compounded monthly for 15 years is \$104,504.

	t=5	t=10	t=15	t=20
k=1	7396.67	18416.6	34834.5	59294.7
k=2	14793.3	36833.1	69669.	118589.
k=3	22190.	55249.7	104504.	177884.
k=4	29586.7	73666.3	139338.	237179.
k=5	36983.4	92082.8	174173.	296474.

□ Example: (Present Value)

Yet another type of problem deals with determining the amount of money which must be invested in order to insure a particular return on the investment over a certain period of time. This is given with the following :

The **present value P** of an annuity of n payments of R dollars each at the end of consecutive interest periods with

interest compounded at a rate of interest j per period is given by:
$$P = R \left[\frac{1 - (1 + j)^{-n}}{j} \right]$$

This problem is illustrated below :

Interest

In[66]:= `present[r_,j_,n_]=r ((1-(1+j)^(-n))/j)`

Out[66]=
$$\frac{(1 - (1 + j)^{-n}) r}{j}$$

In[67]:= `present[45000,.075,40]`

Out[67]= 566748.

In[68]:= `Table[{20000+5000k,present[20000+5000k,.08,35]},{k,0,5}] // TableForm`

Out[68]//TableForm=

20000	233091.
25000	291364.
30000	349637.
35000	407910.
40000	466183.
45000	524456.

present[r, j, n] computes

$$r \left[\frac{1 - (1 + j)^{-n}}{j} \right]$$

present[45000,.075,40] yields the amount of money that would have to be invested at 7 1/2 % compounded annually to provide an ordinary annuity income of \$45,000 per year for 40 years.

creates a table of the amount of money that would have to be invested at 8% compounded annually to provide an ordinary annuity income of \$20,000 + 5000 k per year for 35 years and expresses the result in TableForm.

Notice that the first column corresponds to the annuity income and the second column corresponds to the present value of the annuity.

□ Example: (Deferred Annuities)

Deferred annuities can also be considered :

The **present value** of a deferred annuity of R dollars per period for n periods deferred for k periods with interest

rate j per period is given by:
$$P_{\text{def}} = R \left[\frac{1 - (1 + j)^{-(n+k)}}{j} - \frac{1 - (1 + j)^{-k}}{j} \right]$$

The function which computes the present value of a deferred annuity is given below where
r = the amount of the deferred annuity,
n = the number of years in which in annuity is received,
k = the number of years in which the lump sum investment is made, and
j = the interest rate.

```

Interest
In[1]:=
def[r_,n_,k_,j_]=r ( (1-(1+j)^(-
(n+k))
)/j - (1-(1+j)^(-k))/j )
Out[1]=
      -k                -(k + n)
      1 - (1 + j)      1 - (1 + j)
      (-(------) + -----) r
              j                j
In[2]:=
def[35000,35,30,.15]
Out[2]=
3497.58
In[3]:=
Table[{k,65-k,def[30000,40,65-k,.085]},
{k,25,65,10}] // TableForm
Out[3]//TableForm=
25      40      12988.8
35      30      29367.4
45      20      66399.2
55      10      150127.
65      0       339436.
    
```

def[r,n,k,j] computes

$$r \left[\frac{1-(1+j)^{-(n+k)}}{j} - \frac{1-(1+j)^{-k}}{j} \right]$$

def[35000,35,30,.15]
 computes the lump sum that would have to be invested for 30 years at a rate of 15% compounded annually to provide an ordinary annuity income of \$35,000 per year for 35 years.

creates a table of the lump sums that would have to be invested at a rate of 8 1/2% compounded annually to provide an ordinary annuity income of \$30,000 per year for 40 years.

Current age Present value of the annuity
 Number of years from retirement

□ Example: (Amortization)

A loan is **amortized** if both the principal and interest are paid by a sequence of equal periodic payments. A loan of P dollars at interest rate j per period may be amortized in n equal periodic payments of R dollars made at the end

of each period, where $R = \frac{Pj}{1-(1+j)^{-n}}$.

The function, `amort[p, j, n]`, defined below determines the monthly payment needed to amortize a loan of `p` dollars with an interest rate of `j` % compounded monthly over `n` months. A second function, `totintpaid[p, j, n]`, calculates the total amount of interest paid to amortize a loan of `p` dollars with an interest rate of `j` % compounded monthly over `n` months.

Interest
☰

```

In[32]:=
  amort[p_, j_, n_] =
    (p j) / (1 - (1 + j)^(-n))

Out[32]:=
      j p
-----
      -n
1 - (1 + j)

In[33]:=
  totintpaid[p_, j_, n_] =
    n amort[p, j, n] - p

Out[33]:=
      j n p
-p + -----
      -n
      1 - (1 + j)

In[34]:=
  amort[75000, .095/12, 20 12]

Out[34]:=
699.098

In[35]:=
  Table[{j, amort[80000, j/12, 20 12]},
    {j, .08, .105, .005}] // TableForm

Out[35]:=TableForm=
0.08      669.152

0.085     694.259

0.09      719.781

0.095     745.705

0.1       772.017

0.105     798.704

```

$$p \frac{j}{1 - (1+j)^{-n}}$$

`amort[p, j, n]` computes

$$np \frac{j}{1 - (1+j)^{-n}} - p$$

`totintpaid[p, j, n]` computes

Remember that a space between two numbers denotes multiplication.

calculates the monthly payment necessary to amortize a loan of \$75,000 with interest of 9 1/2% compounded monthly over 20 years.

calculates the monthly payment necessary to amortize a loan of \$80,000 with interest of 8, 8 1/2, 9, 9 1/2, 10, and 10 1/2 percent compounded monthly over a period of twenty years.

The first column corresponds to the annual interest rate and the second column corresponds to the monthly payment.

□ Example:

The first calculation below determines the total amount paid on a loan of \$75,000 at a rate of 9.5% compounded monthly over twenty years while the second shows how much of this amount was paid towards the interest.

The screenshot shows a TI-84 Plus calculator window titled "Interest". It displays two calculations:

```

In[36]:=
  240 amort[75000,.095/12, 240]
Out[36]=
  167784.

In[37]:=
  totintpaid[75000,.095/12,240]
Out[37]=
  92783.6
  
```

Callout boxes provide the following explanations:

- For the first calculation: *calculates the total amount paid to amortize a loan of \$75,000 at a rate of 9 1/2% compounded monthly over a period of twenty years.*
- For the second calculation: *calculates the total interest paid to amortize a loan of \$75,000 at a rate of 9 1/2% compounded monthly over a period of twenty years.*

In many cases, the amount paid towards the principle of the loan and the total amount which remains to be paid after a certain payment need to be computed. This is easily accomplished with the functions `unpaidbalance` and `curprinpaid` defined below using the function `amort[p, j, n]` that was previously defined:

Remark: *Mathematica* does not retain definitions of functions from previous *Mathematica* sessions. This means that in order to use a function definition from a previous *Mathematica* session, the definition must be re-entered.

Interest

```
In[38]:=
unpaidbalance[p_, j_, n_, m_]=
present[amort[p, j, n], j, n-m]
```

Out[38]=

$$\frac{(1 - (1 + j)^{-(-m + n)}) p}{1 - (1 + j)^{-n}}$$

```
In[39]:=
curprinpaid[p_, j_, n_, m_]=
p-unpaidbalance[p, j, n, m]
```

Out[39]=

$$p - \frac{(1 - (1 + j)^{-(-m + n)}) p}{1 - (1 + j)^{-n}}$$

```
In[40]:=
unpaidbalance[60000, .08/12, 120, 60]
```

Out[40]= 35902.1

```
In[41]:=
curprinpaid[60000, .08/12, 120, 60]
```

Out[41]= 24097.9

`unpaidbalance[p, j, n, m]` computes the unpaid balance of a loan of \$p amortized at a rate of j% compounded at each payment for a total of n payments immediately after the mth payment.

`curprinpaid[p, j, n, m]` computes the principal paid immediately after the mth payment.

The unpaid balance of a \$60,000 loan with interest at a rate of 8% compounded monthly scheduled to be amortized over a period of ten years (120 payments) immediately after the 60th payment is \$35,901.10.

Consequently, the total principal paid immediately after the 60th payment is \$60,000-\$35,901.10=\$24,097.9

Mathematica can also be used to determine the total amount of interest paid on a loan using the following function :

Interest

```
In[6]:=
curintpaid[p_, j_, n_, m_] =
  m amort[p, j, n] - curprinpaid[p, j, n, m]
```

curintpaid[p, j, n, m] computes the interest paid on a loan of \$p amortized at a rate of j per period over n periods immediately after the mth payment.

$$-p + \frac{(1 - (1 + j)^{-n}) p}{1 - (1 + j)^{-n}} + \frac{j m p}{1 - (1 + j)^{-n}}$$

```
In[7]:=
curintpaid[60000, .08/12, 120, 60]
```

calculates the interest paid on a loan of \$60,000 with an interest rate of 8% compounded monthly amortized over a period of ten years (120 months) immediately after the 60th payment.

```
Out[7]=
19580.1
```

Tables can be created which show a breakdown of the payments made on a loan (i.e., how much of the total amount paid is allotted to the principle and how much to the interest.) An example is given below :

Interest

```
In[8]:=
amort[45000, .07/12, 15, 12]
```

calculates the monthly payment necessary to amortize a loan of \$45,000 with interest rate 7% compounded monthly over a period of 15 years (15 12= 180 months)

```
Out[8]=
404.473
```

Don't forget that a space between two numbers denotes multiplication.

```
In[9]:=
Table[{t, curprinpaid[45000, .07/12, 15, 12, t],
curintpaid[45000, .07/12, 15, 12, t]},
{t, 0, 15, 3}] // TableForm
```

This table shows the interest paid and principle paid at the end of 0, 3, 6, 9, 12 and 15 years.

```
Out[9]:=TableForm=
```

0	0.	0.	<i>Column 1 represents number of years, column 2 represents principle paid, and column 3 represents interest paid.</i>
3	5668.99	8892.03	
6	12658.4	16463.6	
9	21275.9	22407.2	<i>Thus, at the end of twelve years, \$31,900.60 of the principle has been paid, \$26,343.50 in interest has been paid.</i>
12	31900.6	26343.5	
15	45000	27805.1	

Since `curintpaid[p, j, n, y]` computes the interest paid on a loan of \$`p` amortized at a rate of `j` per period over `n` periods immediately after the `y`th payment, and `curintpaid[p, j, n, y-12]` computes the interest paid on a loan of \$`p` amortized at a rate of `j` per period over `n` periods immediately after the `(y-12)`th payment,

`curintpaid[p, j, n, y]-curintpaid[p, j, n, y-12]` yields the amount of interest paid on a loan of \$`p` amortized at a rate of `j` per period over `n` periods between the `(y-12)`th and `y`th payment.

Consequently, the interest paid and the amount of principle paid over a year can also be computed :

```

Interest
In[10]:=
annualintpaid[p_, j_, n_, y_] :=
  curintpaid[p, j, n, y] - curintpaid[p, j, n, y - 12];
annualprinpaid[p_, j_, n_, y_] :=
  curprinpaid[p, j, n, y] - curprinpaid[p, j, n, y - 12];

In[11]:=
Table[{t, annualintpaid[45000, .07/12, 15, 12, t],
  annualprinpaid[45000, .07/12, 15, 12, t]}]

Out[11]//TableForm= // TableForm
1      3094.26  1759.41  Column 1 represents the number of years
2      2967.08  1886.6   the loan has been held; column 2 represents
3      2830.69  2022.98  the interest paid on the loan during the
4      2684.45  2169.22  year; and column 3 represents the amount
5      2527.64  2326.03  of the principle that has been paid.

```

● **Additional Output Features of Version 2.0**

□ **Example:**

Suppose an investor begins investing at a rate of d dollars per year at an annual rate of $j\%$. Each year the investor increases the amount invested by $i\%$. How much has the investor accumulated after m years?

The following table illustrates the amount invested each year and the value of the annual investment after m years:

Year	Rate of Increase	Annual Interest	Annual Investment	Value after m Years
0		$j\%$	d	$(1 + j\%)^m d$
1	$i\%$	$j\%$	$(1 + i\%)d$	$(1 + i\%)(1 + j\%)^{m-1} d$
2	$i\%$	$j\%$	$(1 + i\%)^2 d$	$(1 + i\%)^2 (1 + j\%)^{m-2} d$
3	$i\%$	$j\%$	$(1 + i\%)^3 d$	$(1 + i\%)^3 (1 + j\%)^{m-3} d$
4	$i\%$	$j\%$	$(1 + i\%)^4 d$	$(1 + i\%)^4 (1 + j\%)^{m-4} d$
5	$i\%$	$j\%$	$(1 + i\%)^5 d$	$(1 + i\%)^5 (1 + j\%)^{m-5} d$
k	$i\%$	$j\%$	$(1 + i\%)^k d$	$(1 + i\%)^k (1 + j\%)^{m-k} d$
m	$i\%$	$j\%$	$(1 + i\%)^m d$	$(1 + i\%)^m d$

It follows that the total value of the amount invested for the first k years after m years is given by:

Year	Total Investment
0	$(1 + j\%)^m d$
1	$(1 + j\%)^m d + (1 + i\%)(1 + j\%)^{m-1} d$
2	$(1 + j\%)^m d + (1 + i\%)(1 + j\%)^{m-1} d + (1 + i\%)^2 (1 + j\%)^{m-2} d$
3	$\sum_{n=0}^3 (1 + i\%)^n (1 + j\%)^{m-n} d$
4	$\sum_{n=0}^4 (1 + i\%)^n (1 + j\%)^{m-n} d$
5	$\sum_{n=0}^5 (1 + i\%)^n (1 + j\%)^{m-n} d$
k	$\sum_{n=0}^k (1 + i\%)^n (1 + j\%)^{m-n} d$
m	$\sum_{n=0}^m (1 + i\%)^n (1 + j\%)^{m-n} d$

o The package `SymbolicSum.m`, contained in the folder `Algebra`, can be used to find a closed form

of the sums $\sum_{n=0}^k (1 + i\%)^n (1 + j\%)^{m-n} d$ and $\sum_{n=0}^m (1 + i\%)^n (1 + j\%)^{m-n} d$

SymbolicSum.m is included with Version 2.0 but not prior versions of *Mathematica*:

FinancialPlanning

```
In[4]:=
  <<SymbolicSum.m
In[47]:=
  closedone=SymbolicSum[(1+i)^n (1+j)^(m-n) d,
    {n,0,k}]
Out[47]=
  d (1 + j)^m (1/(-i + j) + j/(-i + j) - (1 + i)^(1 + k)/((1 + j)^k (-i + j)))
In[48]:=
  Factor[Together[closedone]]
Out[48]=
  (d (1 + j)^(-k + m) ((1 + i)^k + i (1 + i)^k - (1 + j)^k - j (1 + j)^k)) / (i - j)
```

*Finds a closed form for the sum $\sum_{n=0}^k (1+i)^n (1+j)^{m-n} d$ and names the result **closedone**.*

*Writes **closedone** as a single fraction and then factors the numerator.*

In the exact same manner a closed form was found and simplified for $\sum_{n=0}^k (1+i)^n (1+j)^{m-n} d$,

SymbolicSum is used to find a closed form for $\sum_{n=0}^m (1+i)^n (1+j)^{m-n} d$.

In this case, however, the final result is displayed in a print cell in input form with the command **Print[InputForm[%]]**.

```
In[50]:=
  closedtwo=SymbolicSum[(1+i)^n (1+j)^(m-n) d,
    {n,0,m}]
Out[50]=
  d (1 + j)^m (1/(-i + j) + j/(-i + j) - (1 + i)^(1 + m)/((1 + j)^m (-i + j)))
In[51]:=
  Factor[Together[closedtwo]]
Out[51]=
  d (-(1 + i)^m - i (1 + i)^m + (1 + j)^m + j (1 + j)^m) / (-i + j)
In[54]:=
  Print[InputForm[%]]
```

***closedtwo** is the closed form of the sum $\sum_{n=0}^m (1+i)^n (1+j)^{m-n} d$*

*writes **closedtwo** as a single fraction then factors.*

prints the previous output in input form

The above results are used to define the functions `investment[{d,i,j},{k,m}]` and `investmenttot[{d,i,j},m]`. In the second case, notice that print cells can be edited like any other input or text cell. Consequently, we use Macintosh editing features to copy and paste the above result when we define the the function `investmenttot`.

```

investment[{d_,i_,j_},{k_,m_}]=
  (d*(1+j)^(-k+m)*((1+i)^k+i*(1+i)^k-
  (1+j)^k-j*(1+j)^k))/(i-j)

Out[41]=
  (d (1 + j)-k + m ((1 + i)k + i (1 + i)k -
  (1 + j)k - j (1 + j)k) / (i - j)

In[72]:=
investmenttot[{d_,i_,j_},m]=
  (d*(-(1+i)^m-i*(1+i)^m+
  (1+j)^m+j*(1+j)^m)/(-i+j)

Out[72]=
  
$$\frac{d \left( -(1+i)^m - i (1+i)^m + (1+j)^m + j (1+j)^m \right)}{-i + j}$$


```

Finally, `investment` and `investmenttot` are used to illustrate various financial scenarios. In the first example, `investment` is used to compute the value after twenty-five years of investing \$6500 the first year and then increasing the amount invested 5% per year is shown for 5, 10, 15, 20, and 25 years assuming a 15% rate of interest on the amount invested. Version 2.0 contains the built-in function `AccountingForm` which can be used to convert numbers expressed in exponential notation to ordinary notation. In the second example, `investmenttot` is used to compute the value after twenty-five years of investing \$6500 the first year and then increasing the amount invested 5% per year is shown assuming various rates of interest. The results are displayed in `AccountingForm`:

```

In[64]:=
  results=Table[
    {t, investment[ {6500, .05, .15}, {t, 25} ]},
    {t, 5, 25, 5} ] // TableForm

Out[64]//TableForm=
  5    1.03506 106
  10   1.55608 106
  15   1.88668 106
  20   2.09646 106
  25   2.22957 106

  TableForm[ AccountingForm[ results ] ]

Out[65]//TableForm=
  5    1035065.
  10   1556078.
  15   1886680.
  20   2096460.
  25   2229573.

In[82]:=
  scenes=Table[ {i, investmenttot[ {6500, .05, i}, 25] },
    {i, .08, .20, .02} ];
  AccountingForm[ TableForm[ scenes ] ]

Out[82]//AccountingForm=
  0.08  832147.
  0.1   1087126.
  0.12  1437837.
  0.14  1921899.
  0.16  2591636.
  0.18  3519665.
  0.2   4806524.

```

*The command **AccountingForm** can be used to convert numbers expressed in exponential notation to ordinary digit form.*

This table illustrates the total value of investing \$6500 the first year and then increasing the amount invested by 5% per year for 25 years for various rates of interest.

■ **Application:** Graphing Parametric Equations with `ListPlot` and `ParametricPlot`

If `list={{x[1],y[1]},{x[2],y[2]},...,{x[n],y[n]}}` is a list of ordered pairs, `ListPlot[list]` graphs the set of ordered pairs in `list`. The commands `ListPlot` and `Plot` share the same options.

The following example demonstrates how `ListPlot` is used to create a parametric plot. In this case, both coordinates depend on the variable `t`. A list is produced by evaluating the function `f` at values of `t` running from

$t=0$ to $t=3\pi$ using increments of $\frac{3\pi}{150}$.

The ordered pairs obtained are then plotted using `ListPlot`.

```

In[17]:=
Clear[f,list]
f[t_]:= {Cos[t]/(Sqrt[t]+1), t^(1/3) Sin[2t]}
list=Table[f[t],{t,0,3Pi,3Pi/150}];

In[18]:=
ListPlot[list,PlotJoined->True,
AspectRatio->1,Ticks->None]

Out[18]=
-Graphics-

```

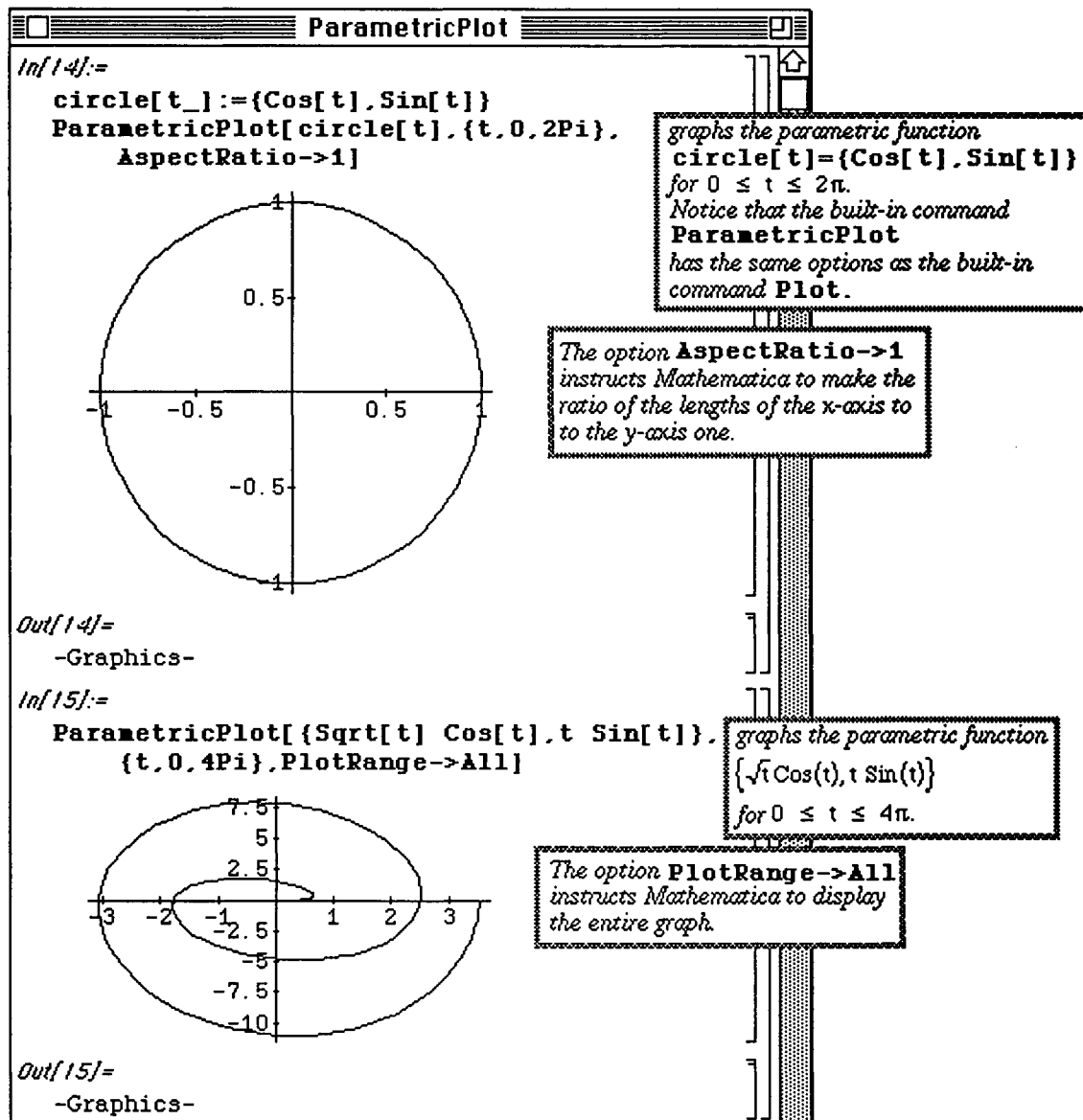
Notice that several commands are combined into a single input cell. The semi-colon is placed at the end to prevent Mathematica from displaying list which consists of 150 ordered pairs.

The commands `ListPlot` and `Plot` share many of the same options.

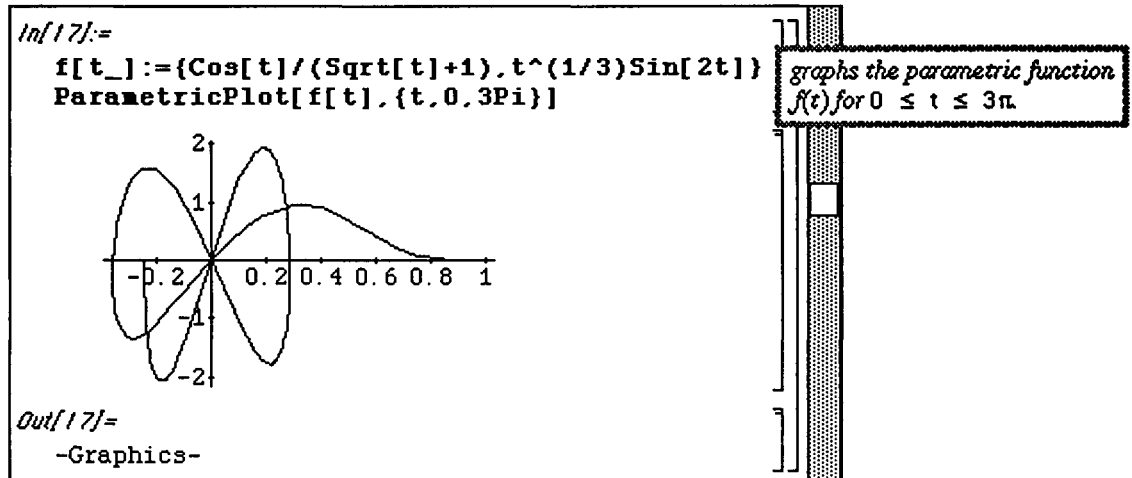
Two-dimensional parametric functions can also be graphed with the built-in function **ParametricPlot**.

□ **Example:**

The unit circle is given by the parametric equation $\{\text{Cos}(t), \text{Sin}(t)\}$, $0 \leq t \leq 2\pi$. To graph the unit circle, proceed as follows:



We can obtain essentially the same result with `ParametricPlot` as we obtained above with `ListPlot`:



■ Application:

Given a function f , the following example illustrates how to create a table of the first, second, third, ... , and n th derivatives (provided they all exist) of f and then graph the resulting table.

□ Example:

Compute and graph the first three derivatives of $f(x) = xe^x$.

Mathematica can produce a table of these derivatives rather easily. This is accomplished through several commands. After defining f , a list of f and its first three derivatives in simplified form is obtained with the command

```
Table[Simplify[D[f[x], {x, n}]], {n, 0, 3}].
```

This list is then placed in the form of a table using the command

```
TableForm[list].
```

Lists can be useful in plotting the graphs of functions. Instead of entering the **GrayLevel** for each function in a multiple plot, these **GrayLevel** assignments can be made with a list. This approach to plotting several functions is shown below :

```

In[37]:=
Clear[f,list1,list2]
f[x_]=x Exp[x]
list1=Table[Simplify[D[f[x],{x,n}]],{n,0,3}]
TableForm[list1]

Out[37]//TableForm=
      x
E x list1[[1]] is f(x).

      x
E (1 + x) list1[[2]] is f'(1)(x).

      x
E (2 + x) list1[[3]] is f'(2)(x).

      x
E (3 + x) list1[[4]] is f'(3)(x).

In[42]:=
list2=Table[GrayLevel[i],{i,0,.75,.75/4}]

Out[42]=
{GrayLevel[0], GrayLevel[0.1875],

  GrayLevel[0.375],

  GrayLevel[0.5625],

  GrayLevel[0.75]}

```

Notice that the following result could have been accomplished with the command `Plot[{f[x], f'[x], f''[x], f'''[x]}, {x, -1, 1}, PlotStyle->{GrayLevel[0], GrayLevel[.1875], GrayLevel[.375], GrayLevel[.5625]}].`

However, since the use of lists simplifies the commands needed, this alternate approach is used. In order to make use of `list1` and `list2` given above, the **Release** command must be used. The command `Release[argument]` causes `argument` to be evaluated immediately.

Hence, the command

```
Plot[Release[list1], {x, -1, 1}]
```

given below causes *Mathematica* to first produce the list of functions in `list1` and then evaluate the functions in the list at the values of `x` between `-1` and `1` in order to plot the functions. Otherwise, a new list would be created for each value of `x`. **Release** is used similarly with the list of **GrayLevel** values.

- o In Version 2.0, **Release** has been replaced by the command **Evaluate**. Consequently, when using Version 2.0, be sure to use **Evaluate** instead of **Release**.

A list of functions can be created and plotted in a single command. The second example below shows how the lines $y=mx$ (where m varies from $m = -4$ to $m = 2$ in increments of $6/5$) are plotted using the same **GrayLevel** list used in the first example.

OperationsOnLists

```
In[43]:=
Plot[Release[list1], {x, -1, 1},
PlotStyle->Release[list2]]
```

Here we graph $f(x)$ and its first three derivatives.

list1[[4]] in shade list2[[4]]

list1[[3]] in shade list2[[3]]

list1[[2]] in shade list2[[2]]

list1[[1]] in shade list2[[1]]

When graphing lists of functions, the command Release must be used.

```
Out[43]=
-Graphics-
```

```
In[45]:=
Plot[Release[Table[m x, {m, -4, 2, 6/5}]],
{x, -1, 1}, PlotStyle->Release[list2]]
```

Here we graph several lines passing through the origin for various slopes.

m=2

m=-4

```
Out[45]=
-Graphics-
```

■ **Application:** Graphing Equations

Often when working problems, the ability to extract a particular element from a list is quite useful. The following example considers the equation $4x^2 + 9y^2 = 81$.

Solving this equation for y yields the two solutions $y = \pm \frac{\sqrt{81 - 4x^2}}{3}$.

Using *Mathematica*, these y -values appear in the form of a list. Notice below how each element of a list can be extracted for later use. Since many results are rather complicated, this technique can save a great deal of time used on typing and limit careless mistakes.

□ **Example:**

Graph the equation $4x^2 + 9y^2 = 81$.

Begin by defining **equation** to be the equation $4x^2 + 9y^2 == 81$. *Be sure* to include the double equals sign so that *Mathematica* interprets **equation** as a mathematical equation.

Notice that the elements of the list **ycoords** are lists. Notice that if

table={list1, list2, ..., listn}, where list1, ..., listn are lists, **table[[1]]** yields the first element of **table** which is list1; **table[[3,2]]** yields the second element of the third element of **table**. In general, **table[[i,j]]** yields the j th element of the i th element of **table**. Lists of lists, or equivalently, nested lists will be discussed in further detail in Chapter 4.

The screenshot shows a Mathematica notebook window with the following content:

```

In[1]:=
equation= 4 x^2+ 9y^2==81
Out[1]=
      2      2
4 x  + 9 y  == 81
In[2]:=
ycoords=Solve[equation, y]
Out[2]=
      2
      Sqrt[81 - 4 x ]
{{y -> -----},
      3
      2
      -Sqrt[81 - 4 x ]
{y -> -----}}
      3
In[3]:=
ycoords[[1]]
Out[3]=
      2
      Sqrt[81 - 4 x ]
{y -> -----}
      3
  
```

Annotations in the notebook:

- A box around the equation in Out[1] says: "Don't forget to include the double-equals between the left- and right-hand side to designate equations."
- A box around the extraction code in In[3] says: "Remember that double-square brackets '[[]]' are used to extract elements of lists."

equation
is the name of the
equation

$$4x^2 + 9y^2 = 81.$$

Solve[equation, y]
solves **equation**
for y . The resulting list
is named
ycoords.

ycoords[[1]]
is the first element of the
list **ycoords**;
it is a list which contains
one element.

The first (and only) element of `ycoords[[1]]` is found below with `ycoords[[1,1]]`. This expression is made up of two parts: the part in front of the arrow and the part following the arrow. Therefore, to obtain the desired formula (the second part), `ycoords[[1,1,2]]` is used:

<pre>In[4]:= ycoords[[1,1]] Out[4]= 2 Sqrt[81 - 4 x] y -> ----- 3 In[5]:= ycoords[[1,1,2]] Out[5]= 2 Sqrt[81 - 4 x] ----- 3</pre>		<p>ycoords[[1,1]] <i>is the single element of the list ycoords[[1]]; it is a list which contains two elements.</i></p> <p>ycoords[[1,1,2]] <i>is the second element of the list ycoords[[1,1]] notice that it is a function of x and hence can be graphed.</i></p>
---	--	--

After extracting the appropriate elements from the list, they can be used below in other commands to determine where the curve intersects the x-axis and then plot the curve.

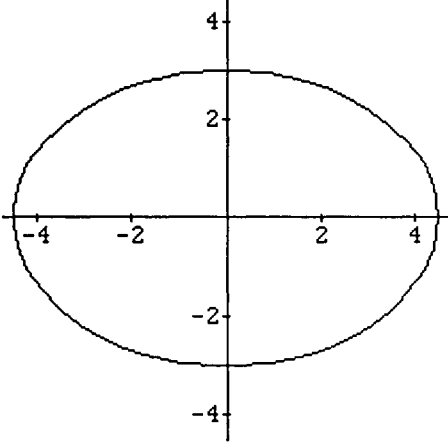
ExtractingElementsofLists

```

In[6]:=
  xvalues=Solve[ycords[[1,1,2]]==0,x]
Out[6]=
  {{x -> - $\frac{9}{2}$ }, {x ->  $\frac{9}{2}$ }}
In[7]:=
  Plot[ycords[[1,1,2]],ycords[[2,1,2]],
  {x,-9/2,9/2},PlotRange->{-9/2,9/2},
  AspectRatio->1]
  
```

solves the equation $\frac{\sqrt{81-4x^2}}{3} = 0$.
 Notice that if $x < -9/2$ or $x > 9/2$,
 $\sqrt{81-4x^2}$ is not real. Hence, the
 domain of `ycords[[1,1,2]]`
 (and `ycords[[2,1,2]]`) is
 $[-9/2,9/2]$.

Graph `ycords[[1,1,2]]`
 and `ycords[[2,2,2]]`
 on the domain $[-9/2,9/2]$. The option
`PlotRange->{-9/2,9/2}`
 specifies that the range of y-values
 shown is $[-9/2,9/2]$; the option
`AspectRatio->1`
 specifies that the ratio of the lengths of
 the x-axis to y-axis is one.



```

Out[7]=
  -Graphics-
  
```

The equation $4x^2 + 9y^2 = 81$ is equivalent to the parametric equation $\left\{ \frac{9}{2} \cos(t), 3 \sin(t) \right\}, 0 \leq t \leq 2\pi$.

Consequently, the same result could have been obtained with the command
`ParametricPlot[{9/2 Cos[t], 3Sin[t]}, {t, 0, 2Pi}]`.

- Moreover, Version 2.0 includes the package `ImplicitPlot.m` which contains the command `ImplicitPlot`. The command `ImplicitPlot` can be used to graph the previous example as well as the next example. For additional information on `ImplicitPlot`, see Chapter 8. Unfortunately, `ImplicitPlot` is not available in versions of *Mathematica* released prior to Version 2.0.

The following example deals with a slightly more complicated curve :

□ Example:

Graph the curve $y^2 - x^4 + 2x^6 - x^8 = 0$.

ExtractingElementsofLists

```

In[8]:=
equation2=y^2-x^4+2x^6-x^8==0
Out[8]=
      4      6      8      2
      -x  + 2x  - x  + y  == 0
In[9]:=
yvalues=Solve[equation2,y]
Out[9]=
      2      2      4
      {{y -> x Sqrt[1 - 2x  + x ]},
      {y -> -(x Sqrt[1 - 2x  + x ])}}
In[10]:=
yvalues[[2]]
Out[10]=
      2      2      4
      {y -> -(x Sqrt[1 - 2x  + x ])}
In[11]:=
yvalues[[2,1]]
Out[11]=
      2      2      4
      y -> -(x Sqrt[1 - 2x  + x ])
In[12]:=
yvalues[[2,1,2]]
Out[12]=
      2      2      4
      -(x Sqrt[1 - 2x  + x ])
In[13]:=
Factor[1-2x^2+x^4]
Out[13]=
      2      2
      (-1 + x) (1 + x)

```

Define the equation
 $y^2 - x^4 + 2x^6 - x^8 = 0$
 to be **equation2**.

Be sure to use two
 equals signs to denote
 equations.

Solve[equation2,y]
 solves the equation
equation2 for **y**; the
 list of solutions is named
yvalues.

yvalues[[2]]
 is the second element of
 the list **yvalues**;
 it is a list with one element.

Don't forget to use double
 square brackets when extracting
 elements of lists.

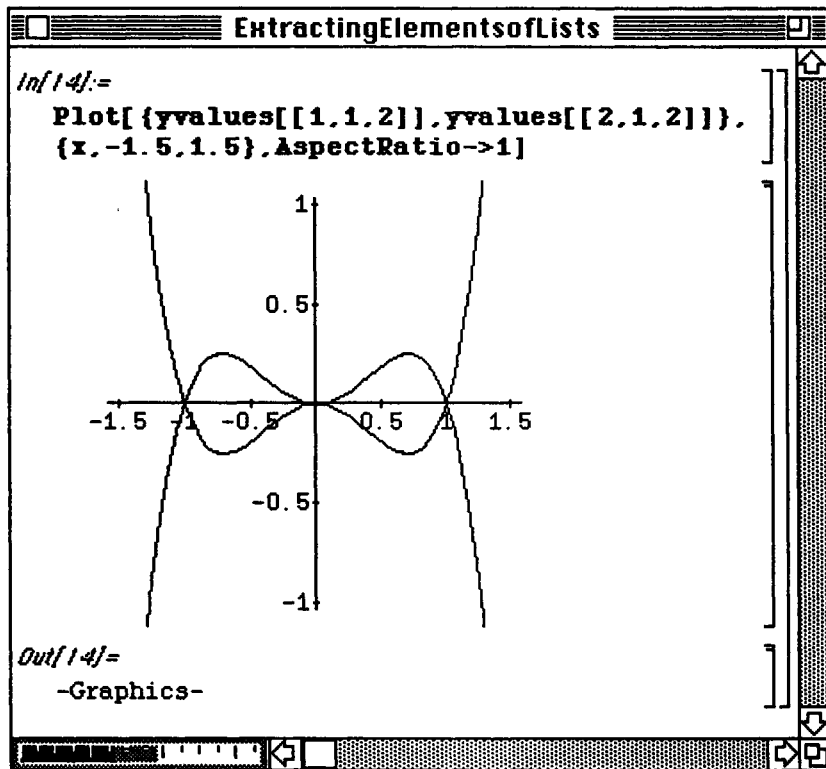
yvalues[[2,1]]
 is the element of the list
yvalues[[2]];
 it is a list with two
 elements.

yvalues[[2,1,2]]
 is the second element of
 the list
yvalues[[2,1]];
 it is a function of **x**.

yvalues[[2,1,2]]
 has domain all real numbers
 since
 $1 - 2x^2 + x^4$
 is a perfect square.

The curve is then easily plotted using the elements of the **yvalues** list obtained above. Notice that this curve passes through the x-axis at $x = -1$ and $x = 1$ as expected from the results of the previous command:

```
Factor[1-2x^2+x^4].
```



*Graphing
yvalues[[1,1,2]]
and
yvalues[[2,1,2]]
simultaneously is
equivalent to graphing
the equation
 $y^2 - x^4 + 2x^6 - x^8 = 0$.*

■ **Application:** Tangent Lines and Animations

The following example illustrates how the tangent line to the graph of $f(x) = x^3 - \frac{9}{2}x^2 + \frac{23}{4}x - \frac{15}{8}$ at many values of x on the interval $[0,3]$ can be determined and plotted through the use of lists. Provided f is differentiable when $x=a$, the line tangent to the graph of f at $x=a$ is given by $y-f(x)=f'(x)(x-a)$.

Hence, this line can be defined as a function of x and a with

```
tanline [x_, a_] := f'[a](x - a) + f[a]
```

The function `tangraph[a]` defined below plots the tangent line to f at $x = a$ for values of x between 0 and 3. In plotting these lines, a list of values of a is needed. This list is created in `table` below :

```

OperationsOnLists

In[50]:=
Clear[f, table, tanline, tangraph]
f[x_]=x^3-9/2 x^2+23/4 x-15/8;
plotf=Plot[f[x], {x, 0, 3},
PlotRange->{-1.5, 1.5}]

Out[50]=
-Graphics-

In[51]:=
tanline[a_, x_]:=
f'[a] (x-a)+f[a]

In[53]:=
tangraph[a_]:=
Plot[{tanline[a, x], f[x]}, {x, 0, 3},
PlotRange->{-1.5, 1.5},
PlotStyle->{GrayLevel[.3], GrayLevel[0]}]

In[54]:=
table=Table[j, {j, 0, 3, 3/20}]

Out[54]=


|     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3   | 3   | 9   | 3   | 3   | 9   | 21  | 6   |     |
| {0, | --, | --, | --, | --, | --, | --, | --, |     |
| 20  | 10  | 20  | 5   | 4   | 10  | 20  | 5   |     |
| 27  | 3   | 33  | 9   | 39  | 21  | 9   | 12  | 51  |
| --, | --, | --, | --, | --, | --, | --, | --, | --, |
| 20  | 2   | 20  | 5   | 20  | 10  | 4   | 5   | 20  |
| 27  | 57  |     |     |     |     |     |     |     |
| --, | --, | 3}  |     |     |     |     |     |     |
| 10  | 20  |     |     |     |     |     |     |     |


```

Since we will name objects f , `table`, `tanline`, and `tangraph`, clear all prior definitions first. Then define

$$f(x) = x^3 - \frac{9}{2}x^2 + \frac{23}{4}x - \frac{15}{8}$$

and graph $f(x)$ on the interval $[0, 3]$. The option `PlotRange->{-1.5, 1.5}` specifies that the range displayed consists of the y -values between -1.5 and 1.5 .

For a given value of a , the function `tanline[a, x]` is the line tangent to the graph of f at the point $(a, f(a))$.

For a given value of a , the function `tangraph[a]` graphs `tanline[a, x]` (in gray) and `f[x]` (in black) on the interval $[0, 3]$. The y -values displayed are between -1.5 and 1.5 .

`table` is a list of the values $0 + n \frac{3}{20}$ for $n = 0, 1, 2, \dots, 30$.

Once the list of **a** values is established in **table**, **Map[tangraph, table]** evaluates **tangraph** at each value of **a** in **table**. Hence, a list of the graphs of the lines tangent to the graph of **f** when $x = a$ is produced. Only the first graph in this list is shown below. The others are hidden but can be seen by double-clicking on the outer ("half-arrow") cell containing the first graph. These graphs can also be viewed using *Mathematica's* animation capabilities.

OperationsOnLists

```
In[2]:=
table=Table[j, {j, 0, 3, 3/20}]

In[5]:=
Map[tangraph, table]
```

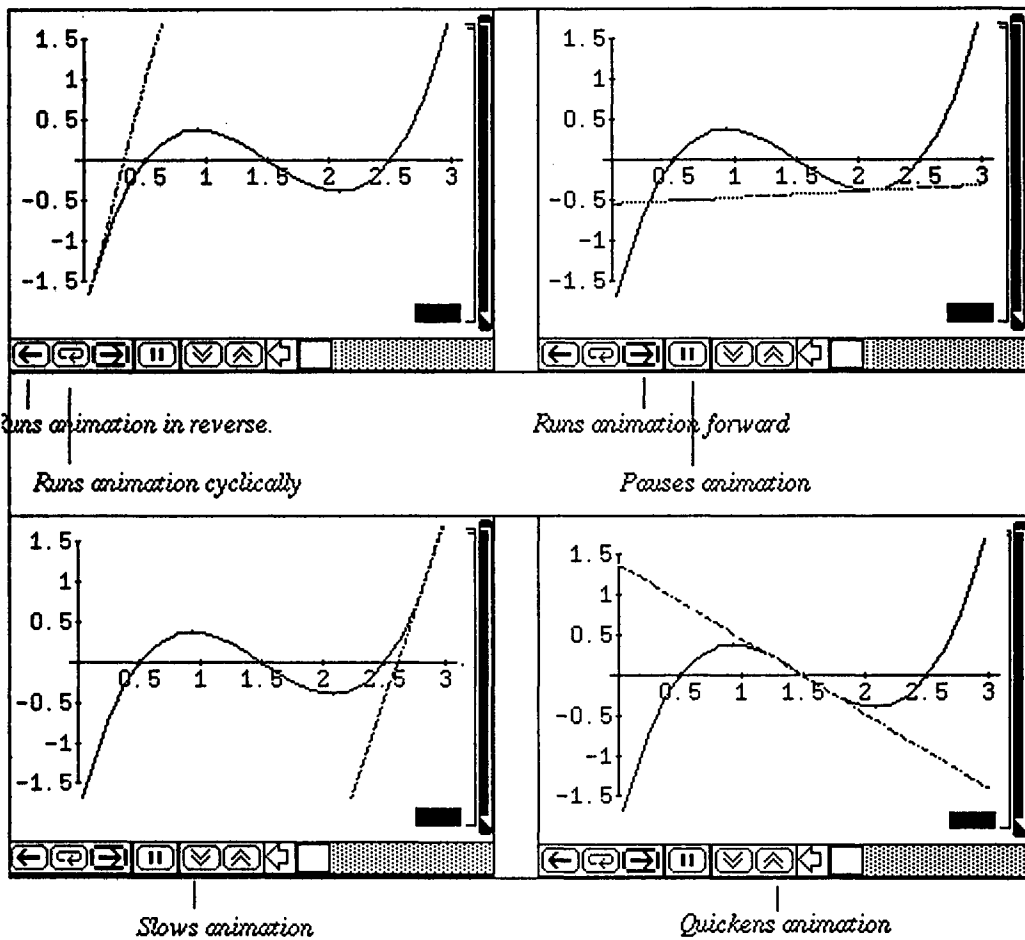
Notice that the arrow and box indicate the cells are grouped and closed. To open a group of cells, move the cursor to either the box or the arrow and click twice.

Map[tangraph, table]
computes
tangraph[table[[i]]]
for each $i=1, 2, \dots, 21$.

Notice that the resulting group of graphics cells are closed.

□ Animation :

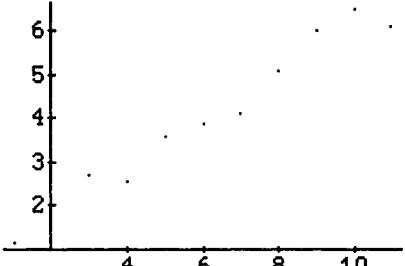
Through using animation, many graphs can be seen in succession. Hence, the graphs can be compared quite easily. The first step towards animation is to click once on the outer cell which encloses all of the graphs. This selects all of the graphs contained in the list. Next, select **Animate Selected Graphics** found under the **Graph** heading in the Menu at the top of the screen. Animation can be halted by clicking once anywhere on the screen. Some of the animation options are demonstrated below :



■ Application: Approximating Lists with Functions

Another interesting application of lists is that of curve-fitting. The command `Fit[data, functionset, variables]` fits a list of data points found in `data` using the functions in `functionset` by the method of least-squares. The functions in `functionset` are functions of the variables listed in `variables`.

An example is shown below which gives a quadratic fit to the data points in `datalist`.

ListsandFunctions	
<pre> In[46]:= datalist={1.14479, 1.5767, 2.68572, 2.5199, 3.58019, 3.84176, 4.09957, 5.09166, 5.98085, 6.49449, 6.12113}; </pre>	<p><i>datalist is the list of values</i> <code>{1.14479, 1.5767, 2.68572, 2.5199, 3.58019, 3.84176, 4.09957, 5.09166, 5.98085, 6.49449, 6.12113}</code>.</p>
<pre> In[47]:= p1=ListPlot[datalist] </pre> 	<p><i>p1 graphs the points</i> <code>{i, datalist[[i]]}</code> for $i = 1, 2, 3, \dots, 10, 11$.</p>
<pre> Out[47]= -Graphics- </pre>	
<pre> In[48]:= Clear[y] y[x_]=Fit[datalist, {1, x, x^2}, x] </pre>	<p><code>Fit[datalist, {1, x, x^2}, x]</code> <i>attempts to find a linear combination of</i> $1, x,$ and x^2 <i>which approximates the list of numbers</i> datalist <i>as closely as possible.</i> <i>The resulting function is named y.</i></p>
<pre> Out[48]= 0.508266 + 0.608688 x - 0.00519281 x² </pre>	

The approximating function obtained above via the least-squares method can be plotted along with the data points. This is demonstrated below. Notice that many of the data points are not very close to the approximating function. Hence, a better approximation is obtained below using a polynomial of higher degree (4).

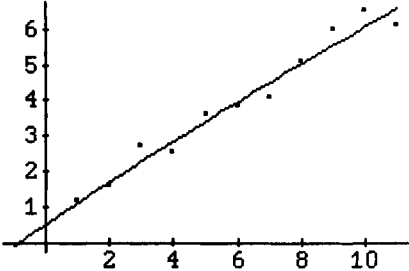
Lists and Functions
⌵

```

In[49]:=
  p2=Plot[y[x].{x,-1,11}.
  DisplayFunction->Identity]
Out[49]=
  -Graphics-
In[50]:=
  Show[p1,p2,DisplayFunction->
  $DisplayFunction]

```

The option `DisplayFunction->Identity` prevents Mathematica from showing the graph.



```

Out[50]=
  -Graphics-
In[51]:=
  Clear[y]
  y[x_]=Fit[datalist,
  {1,x,x^2,x^3,x^4},x]
Out[51]=
  -0.54133 + 2.02744 x - 0.532282 x2 +
  0.0709201 x3 - 0.00310985 x4

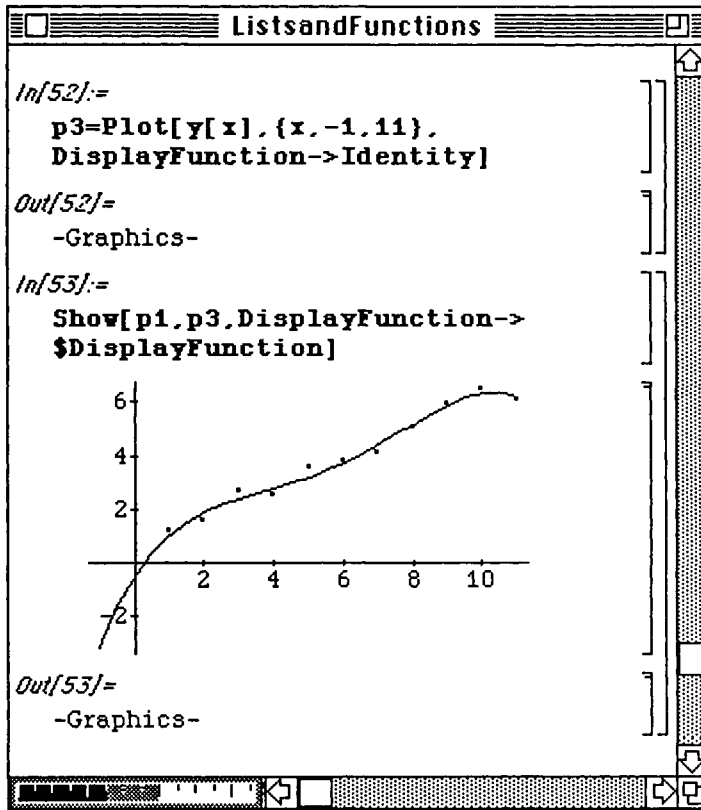
```

`p2=Plot[y[x].{x,-1,11}.
DisplayFunction->Identity]`
graphs *y* on the interval $[-1,11]$ and names the resulting graph *p2*.

`Show[p1,p2,
DisplayFunction->
$DisplayFunction]`
shows the graphs of *p1* and *p2* simultaneously. Hence, we can see how well the fit approximates the data.

`Clear[y]
y[x_]=Fit[datalist,
{1,x,x^2,x^3,x^4},x]`
attempts to find a linear combination of $1, x, x^2, x^3,$ and x^4 to approximate *datalist* as well as possible.

To check its accuracy, this second approximation is simultaneously with the data points.



As before, we can see how well the fit approximates the data.

In this case, the fourth degree fit is a much better approximation of the data than the second degree fit.

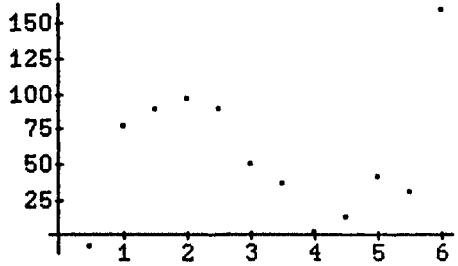
Next, consider a list of data points made up of ordered pairs. These points are plotted below with `ListPlot`, and fitted with a polynomial of degree 3 using `Fit`. (Note that, in this case, since the data is given as ordered pairs, `ListPlot` plots the points as they are given in `datalist`.)

ListsandFunctions

```
In[2]:=
datalist={{.5, -8.4074},
{1, 76.8141},{1.5, 88.6214},{2, 96.127},
{2.5, 88.3714}, {3, 49.797},
{3.5, 35.978}, {4, 2.1036},
{4.5, 12.2676}, {5, 40.4741},
{5.5, 30.0686}, {6, 158.8401}};
```

datalist consists of twelve ordered pairs. The semi-colon is placed at the end of the list to prevent Mathematica from displaying it.

```
In[3]:=
pointgraph=ListPlot[datalist]
```



pointgraph is the graph of the ordered pairs in datalist.

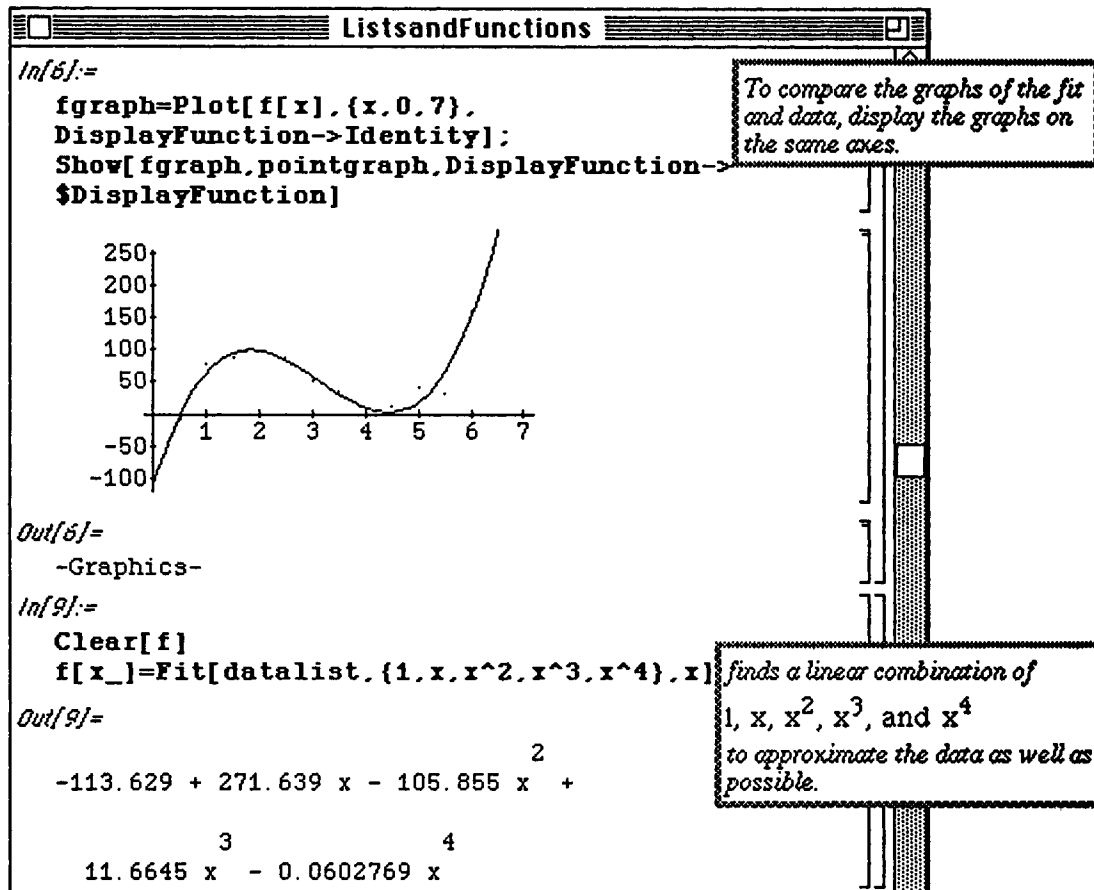
```
Out[3]=
-Graphics-
```

```
In[5]:=
Clear[f]
f[x_]=Fit[datalist,{1,x,x^2,x^3},x]
```

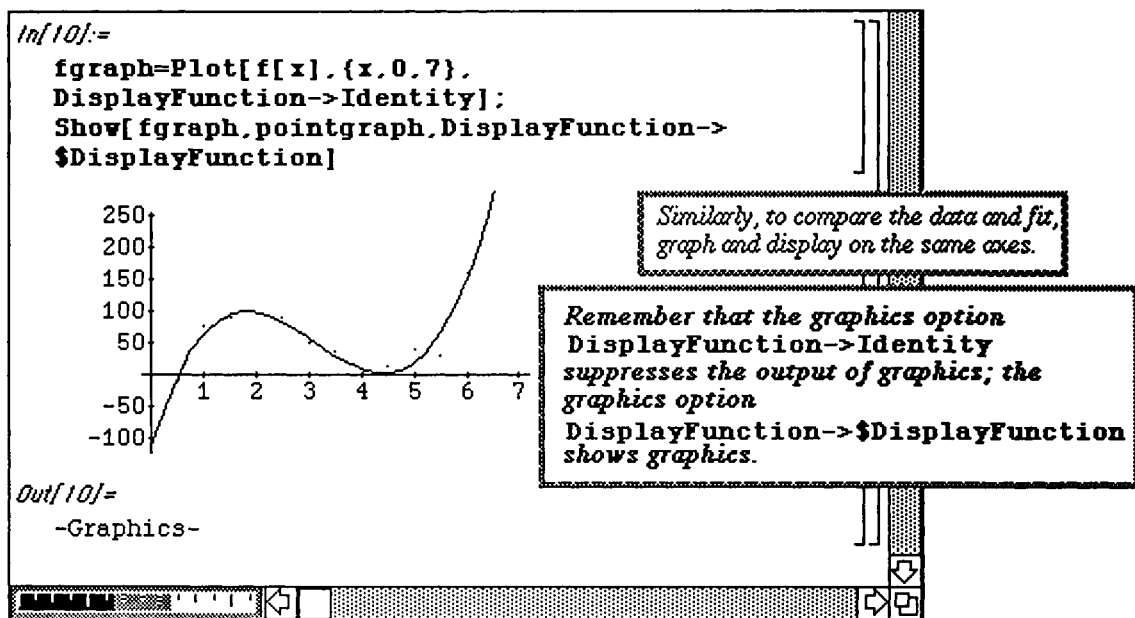
Fit[datalist,{1,x,x^2,x^3},x] finds a linear combination of 1, x, x², and x³ to approximate datalist.

```
Out[5]=
-111.279 + 266.294 x - 102.486 x2 +
10.8809 x3
```


The list of data and the approximating curve $f(x)$ are plotted together to check the accuracy. Then, a polynomial of degree 4 is used to fit the points.



In this case, the fit resulting from `Fit[dataList, {1, x, x^2, x^3, x^4}, x]` does not appear to be much more accurate than the fit resulting from `Fit[dataList, {1, x, x^2, x^3}, x]`:



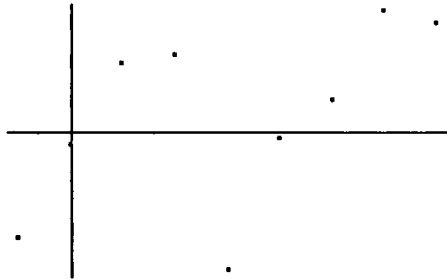
Again, consider a set of data points composed of ordered pairs. These points are listed and plotted below using `ListPlot`. In addition to curve-fitting with polynomials, *Mathematica* can also fit the data with trigonometric functions of the form $C_1 + C_2 \cos x + C_3 \sin x + C_4 \cos 2x + C_5 \sin 2x + \dots$

The approximating function, called *g*, using the first three terms of the above expression is determined below:

```
In[50]:=
datapairs={{.25,-.70499},{.5,-.08331},
           {.75,.46172},{1.,.51234},{1.25,-.91532},
           {1.5,-.04229},{1.75,.21506},{2.,.81278},
           {2.25,.72455}};
```

*In this example,
datapairs
is a list of ordered pairs.*

```
In[51]:=
dataplot=ListPlot[datapairs,Ticks->None]
```



*Remember that the command
ListPlot
has many of same options as
the command
Plot.*

*Here we name the
graph of datapairs
dataplot
and show the graph without
tick marks on either axis.*

```
Out[51]=
-Graphics-
```

```
In[52]:=
Clear[g]
g[x_]=Fit[datapairs,{1,Sin[x],Cos[x]},x]
```

```
Out[52]=
0.458644 - 0.642252 Cos[x] - 0.244861 Sin[x]
```

*computes a linear combination of
1, Sin(x), and Cos(x) to approximate
datapairs
as well as possible.*

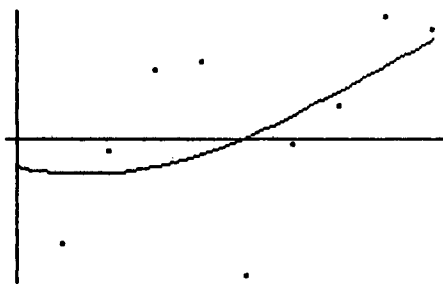
The trigonometric function *g* and the data are then plotted on the same graph. Afterwards, another approximation is calculated using a function of the form

$$C_1 + C_2 \cos x + C_3 \sin x + C_4 \cos 2x + C_5 \sin 2x + C_6 \cos 3x + C_7 \sin 3x.$$

This function is later plotted together with the set of data points and proves to be a much better fit.

```
In[53]:=
```

```
graphg=Plot[g[x].{x,0,2.25},DisplayFunction->
Identity]
Show[dataplot,graphg,DisplayFunction
->$DisplayFunction]
```



```
Out[53]=
```

```
-Graphics-
```

```
In[54]:=
```

```
Clear[g]
g[x_]=Fit[datapairs,{1,Sin[x],Cos[x],
Sin[2 x],Cos[2 x],Sin[3 x],Cos[3 x]},x]
```

```
Out[54]=
```

```
109.244 - 46.7349 Cos[x] -
72.2217 Cos[2 x] + 13.7625 Cos[3 x] -
167.415 Sin[x] + 42.4829 Sin[2 x] +
```

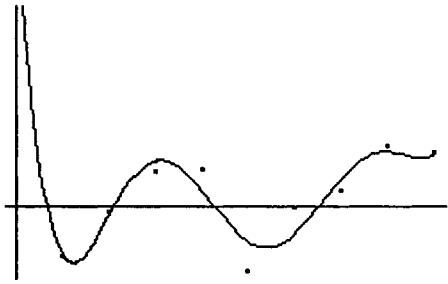
Compare the fit g and the data by showing their graphs on the same axes.

calculates a linear combination of $1, \sin(x), \cos(x), \sin(2x), \cos(2x), \sin(3x),$ and $\cos(3x)$ to approximate the data as well as possible.

In this case, the fit, g , resulting from

`Fit[datapairs, {1, Sin[x], Cos[x], Sin[2x], Cos[2x], Sin[3x], Cos[3x]}, x]` is much better than the fit resulting from `Fit[datapairs, {1, Sin[x], Cos[x]}, x]`:

```
In[55]:=
graphg=Plot[g[x], {x, 0, 2.25}, DisplayFunction->
Identity]
Show[dataplot, graphg, DisplayFunction
-> $DisplayFunction]
```



```
Out[55]=
-Graphics-
```

Notice that this fit g is a much better approximation of the data than the previous fit.

Mathematica supplies several packages which can be used to fit data using different techniques. For additional information regarding the different packages, see Chapter 7.

■ **Application:** Introduction to Fourier Series

Many problems in applied mathematics are solved through the use of Fourier series. *Mathematica* assists in the computation of these series in several ways. First, recall the definition :

The **Fourier series** of a periodic function $f(x)$ with period $2L$ is the trigonometric series

$$a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right], \quad \text{where } a_0 = \frac{1}{2L} \int_{-L}^L f(x) dx, \quad a_n = \frac{1}{L} \int_{-L}^L f(x) \cos\left(\frac{n\pi x}{L}\right) dx,$$

$$\text{and } b_n = \frac{1}{L} \int_{-L}^L f(x) \sin\left(\frac{n\pi x}{L}\right) dx.$$

The **kth term of the Fourier series**

$$a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right] \text{ is } a_k \cos\left(\frac{k\pi x}{L}\right) + b_k \sin\left(\frac{k\pi x}{L}\right)$$

The **kth partial sum of the Fourier series**

$$a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right] \text{ is } a_0 + \sum_{n=1}^k \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right].$$

It is a well-known theorem that if $f(x)$ is a periodic function with period $2L$ and $f(x)$ is continuous on $[-L, L]$ except at most finitely many points, then at each point x the Fourier series corresponding to f converges and

$$a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right] = \frac{\lim_{y \rightarrow x^+} f(y) + \lim_{y \rightarrow x^-} f(y)}{2}.$$

In fact, if the series $\sum_{n=1}^{\infty} (|a_n| + |b_n|)$ converges, then the Fourier series

$$a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right] \text{ converges uniformly on } \mathfrak{R}.$$

Mathematica simplifies the process of determining the coefficients as well as assists in verifying the convergence of the series. Additional applications discussing Fourier Series will be discussed in **Chapter 5**. Consider the following example.

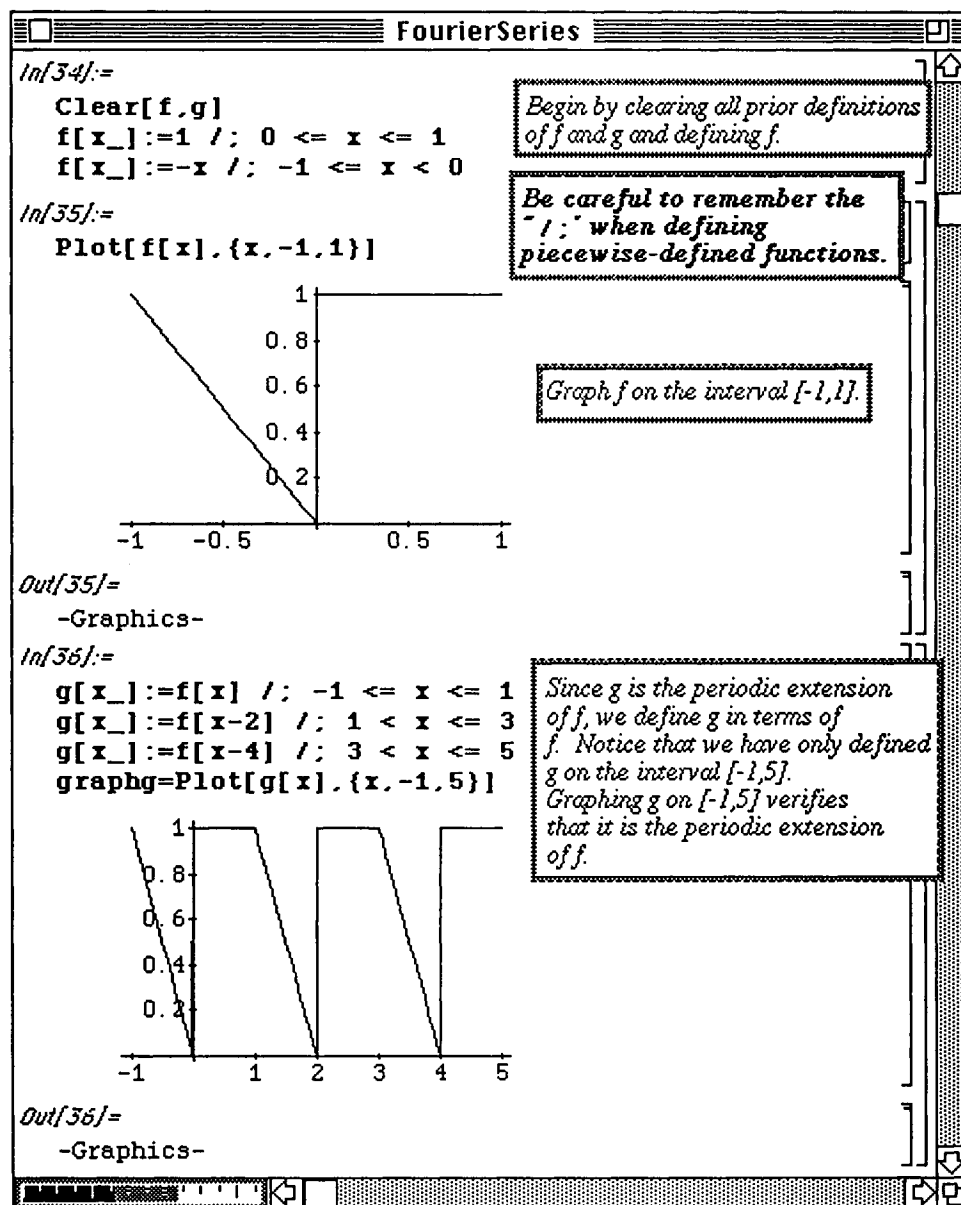
o Version 2.0 includes the package **FourierTransform.m** in the **Calculus** folder.

FourierTransform.m contains several commands which can be used to compute exact or approximate Fourier series of some functions. The package **FourierTransform.m** is discussed in more detail in **Chapter 7**.

□ Example:

Let $f(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ -x & \text{if } -1 \leq x < 0 \end{cases}$ and let g be the periodic extension of f of period 2.

Notice how the piecewise-defined function f is entered. The functions f and g are then plotted. The graph of g is named **graphg** for later use.



The Fourier series coefficients are computed with the integral formulae given earlier. Executing the command

L=1
a[0]=1/(2L) NIntegrate[f[x], {x, -L, L}] defines **L** to be 1 and **a[0]** to be an approximation of

the integral $\frac{1}{2L} \int_{-L}^L f(x) dx$. Executing the command

a[n_]:=1/L NIntegrate[f[x] Cos[n Pi x/L], {x, -L, L}]
b[n_]:=1/L NIntegrate[f[x] Sin[n Pi x/L], {x, -L, L}]

defines **a[n]** to be an approximation of the integral

$\frac{1}{L} \int_{-L}^L f(x) \cos\left(\frac{n\pi x}{L}\right) dx$ and **b[n]** to be an approximation of the integral $\frac{1}{L} \int_{-L}^L f(x) \sin\left(\frac{n\pi x}{L}\right) dx$.

The screenshot shows a Mathematica notebook window titled "FourierSeries". The input area contains the following code:

```
In[38]:=
Clear[a,b,f s,L]
L=1
a[0]=1/(2L) NIntegrate[f[x], {x, -L, L}]
```

The output area shows the result of the first command:

```
Out[38]=
0.750409
```

The input area also contains the following code:

```
In[5]:=
a[n_]:=1/L NIntegrate[f[x] Cos[n Pi x/L], {x, -L, L}]
b[n_]:=1/L NIntegrate[f[x] Sin[n Pi x/L], {x, -L, L}]
```

Annotations in the notebook provide context:

- A box next to `L=1` states: "For this example, L is assigned the value 1."
- A box next to the `NIntegrate` command in the first input block states: "To approximate the Fourier series for f, we need to compute a₀ and various values of a_n and b_n."
- A box next to the `NIntegrate` commands in the second input block states: "a_n and b_n will be approximated using NIntegrate."

A table containing the coefficients $a[i]$ and $b[i]$ for $i = 1, 2, 3, \dots, 10$ is created. Notice how the elements of the table are extracted using double brackets with `coeffs`, the name of the table :

coeffs
is the table of ordered pairs
 $\{a[i], b[i]\}$ for $i = 1, 2, \dots, 9, 10$.

TableForm[coeffs]
expresses coeffs in a column.
Notice that the first column
corresponds to $a[i]$; the second
column corresponds to $b[i]$.

Notice that the command TableForm[coeffs] produces the same result as the command coeffs // TableForm.

Be sure to use double-square brackets to extract elements of tables and lists!

coeffs[[1]] yields the first element of coeffs which is an ordered pair, a list with two elements.

coeffs[[2,1]] yields the first member of the second element of coeffs.

coeffs[[3,2]] yields the second member of the third element of coeffs.

```

In[6]:=
  coeffs=Table[{a[i],b[i]},{i,1,10}];
In[7]:=
  TableForm[coeffs]
Out[7]//TableForm=
-0.201825    0.318309
0.000817324 0.159153
-0.0216985  0.1061
0.000817313 0.0795733
-0.00728839 0.0636567
0.000817295 0.0530454
-0.00331828 0.0454655
0.00081727  0.0397803
-0.0016845  0.0353583
0.000817237 0.0318205
In[9]:=
  coeffs[[1]]
Out[9]=
{-0.201825, 0.318309}
In[10]:=
  coeffs[[2,1]]
Out[10]=
0.000817324
In[11]:=
  coeffs[[3,2]]
Out[11]=
0.1061

```

The command `Sum[f[i], {i, 1, n}]` computes the sum

$$f[1]+f[2]+\dots+f[n-1]+f[n]=\sum_{i=1}^n f[i].$$

Once the coefficients are calculated, the k th partial sum of the Fourier series is obtained with **Sum**.

The k th term of the series is:

$$fs[k,x] := coeffs[[k,1]] \text{ Cos}[k \text{ Pi } x] + coeffs[[k,2]] \text{ Sin}[k \text{ Pi } x]$$

where $k=1, 2, \dots$, $a_k = coeffs[[k,1]]$, and $b_k = coeffs[[k,2]]$.

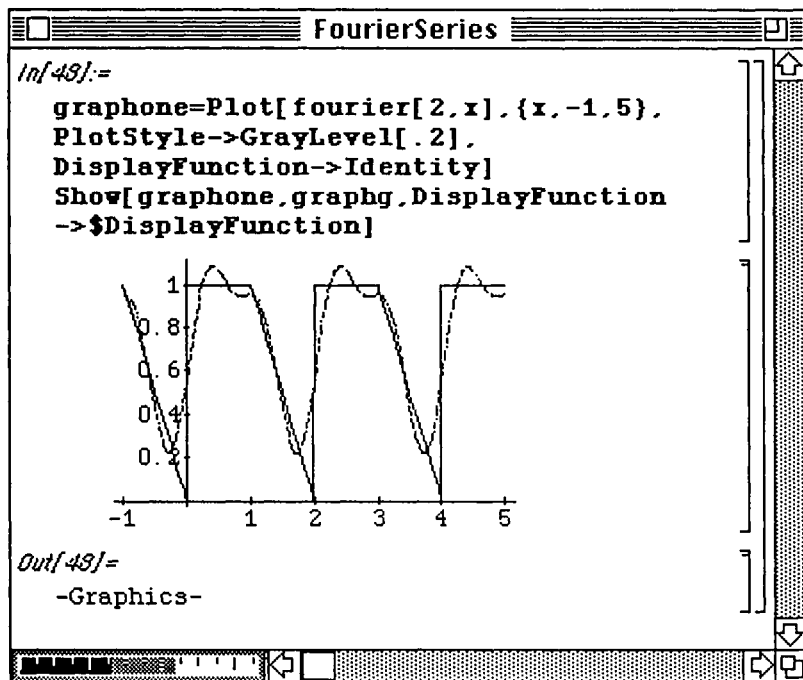
Therefore, the n th partial sum (the sum of the first n terms) of the Fourier series is obtained by summing the **fs[k,x]** over k from $k=1$ to $k=n$ and adding to this summation the coefficient **a[0]**:

$$\text{nth partial sum} = a[0] + fs[1,x] + \dots + fs[n-1,x] + fs[n,x] = a[0] + \sum_{k=1}^n fs[k,x].$$

This is defined below with the function **fourier[n,x]**. Several examples of this function are then given. Note that the largest value of n which can be used is $n=10$ since the coefficients for the Fourier series have not been calculated for larger values of n .

FourierSeries	
<i>In[44]:=</i>	fs[k,x] is the term $a_k \text{Cos}(k\pi x) + b_k \text{Sin}(k\pi x)$.
fs[k_,x_] := coeffs[[k,1]] Cos[k Pi x] + coeffs[[k,2]] Sin[k Pi x]	
<i>In[45]:=</i>	fourier[n,x] computes $a_0 + \sum_{k=1}^n fs[k,x]$.
fourier[n_,x_] := a[0] + Sum[fs[k,x], {k,1,n}]	
<i>In[46]:=</i>	fourier[2,x] is the function $a_0 + \sum_{k=1}^2 fs[k,x]$.
fourier[2,x]	
<i>Out[46]=</i>	fourier[2,x] is the function $a_0 + \sum_{k=1}^2 fs[k,x]$.
0.750409 - 0.201825 Cos[Pi x] + 0.000817324 Cos[2 Pi x] + 0.318309 Sin[Pi x] + 0.159153 Sin[2 Pi x]	
<i>In[47]:=</i>	fourier[3,x] is the function $a_0 + \sum_{k=1}^3 fs[k,x]$.
fourier[3,x]	
<i>Out[47]=</i>	
0.750409 - 0.201825 Cos[Pi x] + 0.000817324 Cos[2 Pi x] - 0.0216985 Cos[3 Pi x] + 0.318309 Sin[Pi x] + 0.159153 Sin[2 Pi x] + 0.1061 Sin[3 Pi x]	

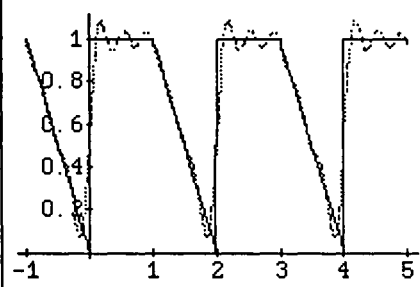
To see how the Fourier series approximates the function, the function g is plotted together with the graph of the Fourier series with $n = 2$. This does not appear to be a very good approximation. (The resulting graph is named **graphone**.)



graphone is the graph of **fourier[2,x]** on the interval $[-1,5]$. Notice that the **Plot** and **Show** command are combined into a single input cell to show **graphg** and **graphone** simultaneously.

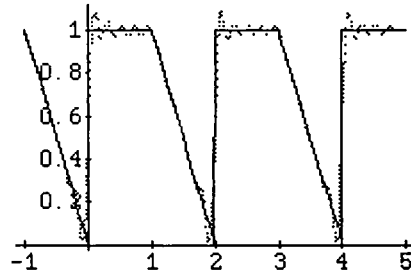
The use of more terms in the Fourier series yields a more accurate approximation of g . Graphs are shown for $n = 5$ and $n = 9$. These are named **graphfive** and **graphnine**, respectively.

```
graphfive=Plot[fourier[5,x],{x,-1,5},
PlotStyle->GrayLevel[.4],
DisplayFunction->Identity]
Show[graphfive,graphg,DisplayFunction
->{$DisplayFunction}]
```



In the same manner, we can compare **fourier[5,x]**, **fourier[9,x]**, and **g**.

```
graphnine=Plot[fourier[9,x],{x,-1,5},
PlotStyle->GrayLevel[.6],
DisplayFunction->Identity]
Show[graphnine,graphg,DisplayFunction
->{$DisplayFunction}]
```



□ **Example: (One-Dimensional Heat Equation)**

A typical problem in applied mathematics which involves the use of Fourier series is that of the one-dimensional heat equation. This initial-value problem which describes the temperature in a uniform rod with insulated surface is given by:

- (1) $k \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}, \quad 0 < x < a, \quad t > 0;$
- (2) $u(0, t) = T_0; \quad t > 0;$
- (3) $u(a, t) = T_a; \quad t > 0; \quad \text{and}$
- (4) $u(x, 0) = f(x), \quad 0 < x < a.$

The solution to the problem is well-known:

$$u(x, t) = \underbrace{T_0 + \frac{x(T_a - T_0)}{a}}_{v(x)} + \sum_{n=1}^{\infty} b_n \sin(\lambda_n x) e^{-\lambda_n^2 kt}, \quad \text{where } \lambda_n = \frac{n\pi}{a} \text{ and } b_n = \frac{2}{a} \int_0^a [f(x) - v(x)] \sin\left(\frac{n\pi x}{a}\right) dx$$

and is obtained through separation of variables techniques. The coefficient b_n in the solution $u(x, t)$ is the Fourier series coefficient b_n of the function $f(x) - v(x)$, where $v(x)$ is the steady-state temperature.

Consider the heat equation with initial temperature distribution $f(x) = -(x-1) \cos(\pi x)$.

The steady-state temperature for this problem is $v(x) = 1 - x$, and the eigenvalue,

λ_n , is given by $\frac{n\pi}{2}$.

The function f is defined and plotted below. Also, the steady-state temperature, $v(x)$, and the eigenvalue are defined. Finally, **NIntegrate** is used to define a function which will be used to calculate the coefficients of the solution.

FourierSeries

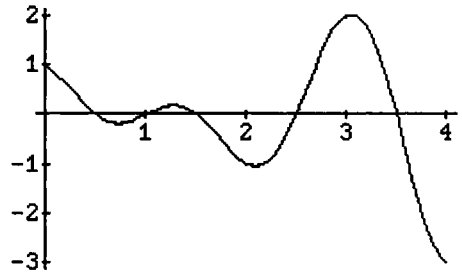
```

In[10]:=
f[x_]:=-(x-1) Cos[Pi x]
Plot[f[x],{x,0,4}]

Out[10]=
-Graphics-

In[24]:=
v[x_]:=1-x
lambda[n_]:=n Pi/2
b[n_]:=NIntegrate[(f[x]-v[x])
Sin[n Pi x/2],{x,0,4}]

```



First define and graph f to visualize the initial temperature distribution.

In this example,
 $v(x) = 1-x$,
 $\lambda_n = \frac{n\pi}{2}$, and
 $b_n = \int_0^4 (f(x) - v(x)) \sin\left(\frac{n\pi x}{2}\right) dx$

We use numerical integration to save time in calculations

Let $S_m = b_m \sin(\lambda_m x) e^{-\lambda_m^2 kt}$. Then the desired solution $u(x,t)$ is given by $u(x,t) = v(x) + \sum_{m=1}^{\infty} S_m$.

Let $u(x,t,n) = v(x) + \sum_{m=1}^n S_m$.

Notice that $u(x,t,n) = u(x,t,n-1) + S_n$. Consequently approximations of the solution to the heat equation are obtained recursively. The solution is first defined for $n = 1$, $u[x,t,1]$.

Subsequent partial sums, $u[x,t,n]$, are obtained by adding the n th term of the series,

$S_n = b_n \sin(\lambda_n x) e^{-\lambda_n^2 kt}$ to $u[x,t,n-1]$.

```

In[25]:=
u[x_,t_,1]:=v[x]+b[1] Sin[lambda[1] x
]Exp[-lambda[1]^2 1/4 t]
u[x_,t_,n_]:=u[x,t,n-1]+b[n] Sin[
lambda[n] x]Exp[-lambda[n]^2 1/4 t]

```

Notice $u(x,t,n)$ is obtained via adding the n th term to $u(x,t,n-1)$. Hence, we take advantage of Mathematica's ability to compute recursively.

By defining the solution in this manner a table can be created which includes the partial sums of the solution :

FourierSeries

```
In[19]:=
  Table[u[x,t,n],{n,1,3}] // TableForm
Out[19]//TableForm=
```

$1 - x - \frac{3.39531 \sin\left(\frac{\pi x}{2}\right)}{E^{(\pi t)/16}}$	$1 - x - \frac{3.39531 \sin\left(\frac{\pi x}{2}\right)}{E^{(\pi t)/16}} - \frac{0.95493 \sin[\pi x]}{E^{(\pi t)/4}}$	$1 - x - \frac{3.39531 \sin\left(\frac{\pi x}{2}\right)}{E^{(\pi t)/16}} - \frac{0.95493 \sin[\pi x]}{E^{(\pi t)/4}} + \frac{0.679061 \sin\left(\frac{3\pi x}{2}\right)}{E^{(9\pi t)/16}}$
---	---	--

This term is $u(x,t,1)$

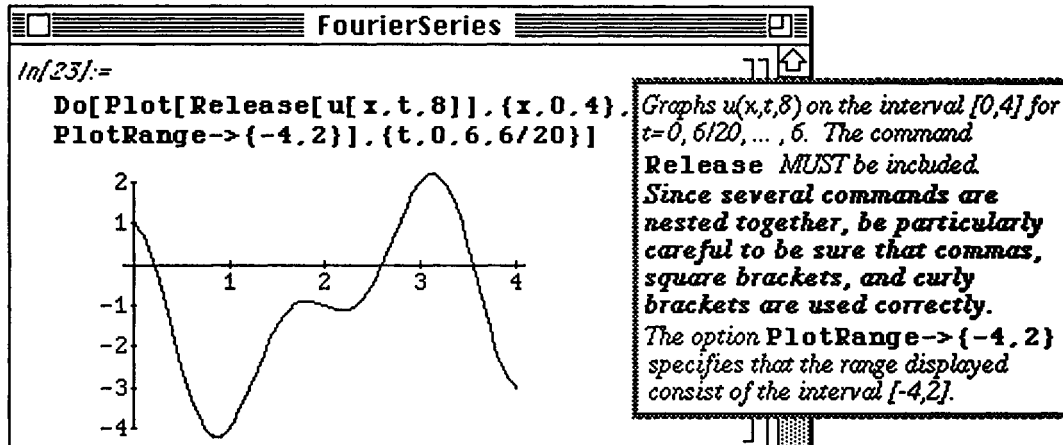
This term is $u(x,t,2)$

This term is $u(x,t,3)$

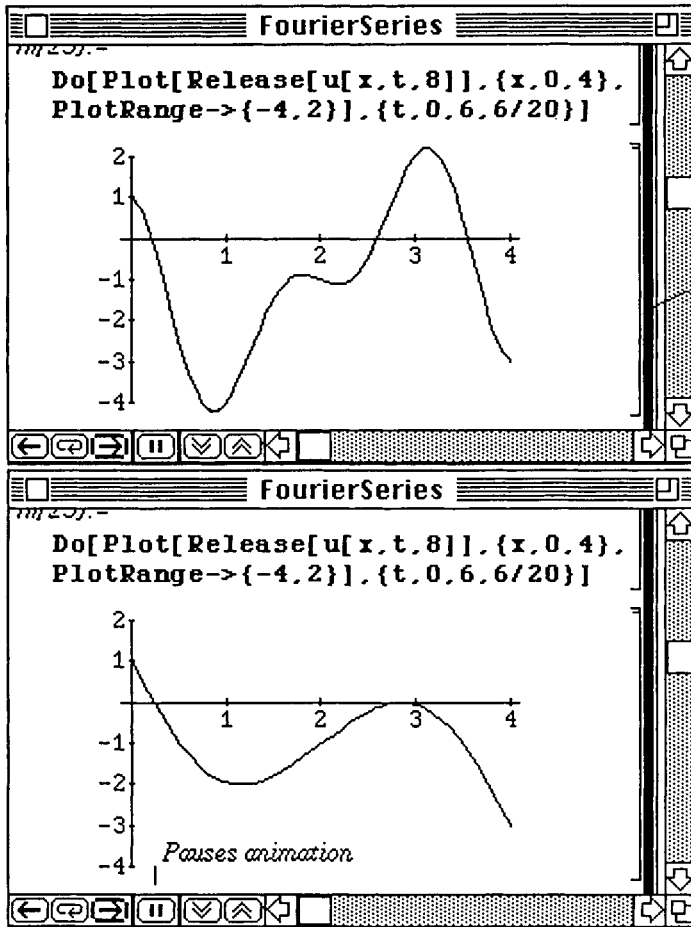
creates a table of the functions $u(x,t,1)$, $u(x,t,2)$ and $u(x,t,3)$. The result is expressed in a column.

The syntax of the *Mathematica* command **Do** is similar to the syntax of the command **Table**. The command **Do**[statement[i], {i, istart, istop, istep}] instructs *Mathematica* to execute statement[i] for values of i beginning with istart and continuing through istop in increments of istep.

The solution with $n = 8$ is plotted below for $t = 0$ to $t = 6$ using a step-size in t of $6/20$. Remember that $u[x, t, n]$ is determined with a **Table** command. Therefore, **Release** must be used in the **Do** command below so that *Mathematica* first computes the solution u and then evaluates u at the particular values of x . Otherwise, u is recalculated for each value of x .



The plots of the solution obtained above can be animated :



To animate graphics:

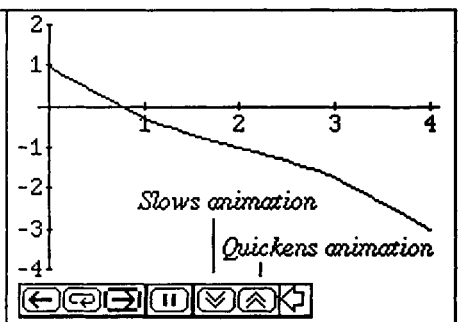
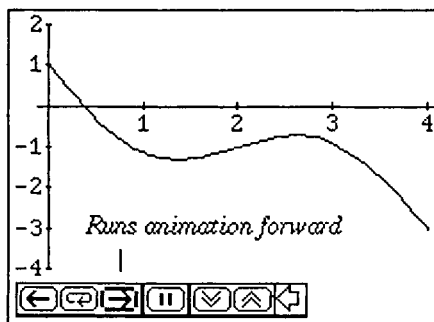
1) Select the group of graphics cells to be animated by using the mouse to move the arrow to the outermost cell brackets and clicking once. The group of cells will become highlighted.

2) Either go to Graphics and select **Animate Selected Graphics** OR press the **Open-Apple** and **y-key** simultaneously.

These four diagrams are four of the twenty graphics cell generated by the Do command.

Notice that animations can be modified while they are in progress by using clicking the six buttons that appear at the bottom of the screen while an animation is running.

Runs animation cyclically
Runs animation in reverse



Chapter 5

Introduction to Nested Lists: Matrices and Vectors

- Chapter 5 discusses operations on matrices and vectors, including vector calculus and systems of equations. Several linear programming examples are discussed.
- Commands introduced and discussed in this chapter from Version 1.2 include:

Operations on Matrices

```

matrixa+matrixb
matrixa.matrixb
Det[matrix]
Eigenvalues[matrix]
Eigenvectors[matrix]
IdentityMatrix[positiveinteger]
Inverse[matrix]
Transpose[matrix]
LinearSolve[matrix,vector]
MatrixPower[matrix,positiveinteger]
MatrixForm[matrix]

```

Operations to Create Lists and Tables

```

Table[expression,{positiveinteger}]
Array[variable,positiveinteger]

```

Vector Calculus

```

<<VectorAnalysis.m
Grad[scalarfield,coordinatesystem]
Laplacian[scalarfield,coordinatesystem]
Div[vectorfield,coordinatesystem]
Curl[vectorfield,coordinatesystem]
SetCoordinates[System]

```

Linear Programming

```

ConstrainedMin[function,{inequalities},{variables}]
ConstrainedMax[function,{inequalities},{variables}]
LinearProgramming[vectorc,matrixa,vectorb]

```

Other Commands

```

TrigExpand[expression]
BesselJ[alpha,x]
Print[expression]

```

Saving and Appending Output for Future Mathematica Sessions

```

>>filename
>>>filename

```

Commands introduced and discussed in this chapter from Version 2.0 include:

```

Expand[Trig->True]

```

- Applications in this chapter include linear programming, vector calculus, and saving results for future *Mathematica* sessions.

■ 5.1 Nested Lists: Introduction to Matrices, Vectors and Matrix Operations

■ Defining Matrices and Vectors

Matrix algebra can be performed with *Mathematica*. Before introducing the operations involved in matrix algebra, the method by which a matrix is entered must first be discussed. In *Mathematica*, a matrix is simply a list of lists where each list represents a row of the matrix. Therefore, the $m \times n$ matrix

$$A = [a_{i,j}] = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{bmatrix} \text{ is entered in the following manner:}$$

$$A = \{\{a_{1,1}, a_{1,2}, a_{1,3}, \dots, a_{1,n}\}, \{a_{2,1}, a_{2,2}, a_{2,3}, \dots, a_{2,n}\}, \dots, \{a_{m,1}, a_{m,2}, a_{m,3}, \dots, a_{m,n}\}\}.$$

For example, to use *Mathematica* to define **m** to be the matrix $\begin{bmatrix} a[1,1] & a[1,2] \\ a[2,1] & a[2,2] \end{bmatrix}$ execute

the command **m = {{a[1,1], a[1,2]}, {a[2,1], a[2,2]}}**.

Another way to create a matrix is to use the command **Array**. The command **m=Array[a, {2,2}]** produces the same result as above.

The following examples illustrate the definition of a 3 x 3 matrix.

□ Example:

`matrixa=Table[a[i,j],{i,1,3},{j,1,3}]` and `matrixaprime=Array[a,{3,3}]` produce the same result.

CreatingMatrices *Begin by clearing all prior definitions of objects to be used in this example.*

```

In[1]:=
  Clear[a,b,matrixa,matrixaprime,matrixb]

In[2]:=
  matrixa=Table[a[i,j],{i,1,3},{j,1,3}]

Out[2]=
  {{a[1, 1], a[1, 2], a[1, 3]},
   {a[2, 1], a[2, 2], a[2, 3]},
   {a[3, 1], a[3, 2], a[3, 3]}}

In[3]:=
  MatrixForm[matrixa]

Out[3]//MatrixForm=
  a[1, 1] a[1, 2] a[1, 3]
  a[2, 1] a[2, 2] a[2, 3]
  a[3, 1] a[3, 2] a[3, 3]

In[4]:=
  matrixaprime=Array[a,{3,3}]

Out[4]=
  {{a[1, 1], a[1, 2], a[1, 3]},
   {a[2, 1], a[2, 2], a[2, 3]},
   {a[3, 1], a[3, 2], a[3, 3]}}

In[5]:=
  MatrixForm[matrixaprime]

Out[5]//MatrixForm=
  a[1, 1] a[1, 2] a[1, 3]
  a[2, 1] a[2, 2] a[2, 3]
  a[3, 1] a[3, 2] a[3, 3]

```

defines matrixa to be the matrix

a[1,1]	a[1,2]	a[1,3]
a[2,1]	a[2,2]	a[2,3]
a[3,1]	a[3,2]	a[3,3]

MatrixForm[matrixa] displays matrixa in traditional matrix form. The command matrixa[[2,3]] yields a[2,3].

defines matrixaprime to be the matrix

a[1,1]	a[1,2]	a[1,3]
a[2,1]	a[2,2]	a[2,3]
a[3,1]	a[3,2]	a[3,3]

MatrixForm[matrixaprime] displays matrixaprime in traditional matrix form.

In addition, non-square matrices may also be defined using *Mathematica*. For example,

to define `matrixb` to be the matrix $\begin{bmatrix} b[1,1] & b[1,2] & b[1,3] & b[1,4] \\ b[2,1] & b[2,2] & b[2,3] & b[2,4] \end{bmatrix}$, enter either

`matrixb=Table[b[i,j],{i,1,2},{j,1,4}]` or `matrixb=Array[b,{2,4}]`.

```
In[6]:=
  matrixb=Array[b,{2,4}]
```

```
Out[6]=
  {{b[1, 1], b[1, 2], b[1, 3], b[1, 4]},
   {b[2, 1], b[2, 2], b[2, 3], b[2, 4]}}
```

```
In[7]:=
  MatrixForm[matrixb]
```

```
Out[7]//MatrixForm=
  b[1, 1] b[1, 2] b[1, 3] b[1, 4]
  b[2, 1] b[2, 2] b[2, 3] b[2, 4]
```

```
Array[b,{2,4}]
```

yields the 2 x 4 matrix

```
b[1,1] b[1,2] b[1,3] b[1,4]
b[2,1] b[2,2] b[2,3] b[2,4]
```

The same result would have been
obtained using the command

```
Table[b[i,j],{i,1,2},{j,1,4}]
```

□ Example:

Use *Mathematica* to create the matrix **matrixc**,

$$\begin{bmatrix} c[1,1] & c[1,2] & c[1,3] & c[1,4] \\ c[2,1] & c[2,2] & c[2,3] & c[2,4] \\ c[3,1] & c[3,2] & c[3,3] & c[3,4] \end{bmatrix}, \text{ where } c(i,j) \text{ is the numerical value of } \text{Cos}(j^2 - i^2) \text{Sin}(i^2 - j^2).$$

```

In[17]:=
  Clear[c,matrixc]
  c[i_,j_]=N[Cos[j^2-i^2] Sin[i^2-j^2]]
Out[17]=
      2      2      2      2
Cos[-1. i  + j ] Sin[i  - 1. j ]
In[18]:=
  matrixc=Array[c,{3,4}]
Out[18]=
  {{0., 0.139708, 0.143952, 0.494016},
   {-0.139708, 0., 0.272011, 0.452789},
   {-0.143952, -0.272011, 0., -0.495304}}
In[19]:=
  MatrixForm[matrixc]
Out[19]//MatrixForm=
  0.      0.139708  0.143952  0.494016
 -0.139708  0.      0.272011  0.452789
 -0.143952 -0.272011  0.      -0.495304
  
```

After clearing all prior definitions of *c* and *matrixc*, define *c(i,j)* to be the numerical value of $c(i,j) = \text{Cos}(j^2 - i^2) \text{Sin}(i^2 - j^2)$.

Array[c,{3,4}]
computes the 3 x 4 matrix

c[1,1]	c[1,2]	c[1,3]	c[1,4]
c[2,1]	c[2,2]	c[2,3]	c[2,4]
c[3,1]	c[3,2]	c[3,3]	c[3,4]

and names the result **matrixc**.

A matrix is a nested list. For the 2×2 matrix $m = \{\{a[1, 1], a[1, 2]\}, \{a[2, 1], a[2, 2]\}\}$ defined earlier, `m[[1]]` yields the first element of matrix m which is the list `{a[1, 1], a[1, 2]}`; `m[[2, 1]]` yields the first element of the second element of matrix m which is `a[2, 1]`. In general, if `matrixm` is an $m \times n$ matrix, `matrixm[[i, j]]` yields the unique element in the i th row and j th column.

Once a matrix has been entered, it can be placed in the usual form (with rows and columns) using the command `MatrixForm[A]`.

In *Mathematica*, a vector is a list of numbers. For example, to use *Mathematica*

to define the row vector `vectorv` to be $[v[1], v[2], v[3]]$,

enter `vectorv={v[1], v[2], v[3]}`.

Similarly, to define the column vector `vectorv` to be $\begin{bmatrix} v[1] \\ v[2] \\ v[3] \end{bmatrix}$

enter `vectorv={v[1], v[2], v[3]}`. Thus, *Mathematica* does not distinguish between row and column vectors. Nevertheless, *Mathematica* performs computations with vectors and matrices correctly.

■ Extracting Elements of Matrices:

Once `matrixa` has been defined via either

```
matrixa=
{{a[1, 1], ..., a[1, n]}, {a[2, 1], ..., a[2, n]}, ..., {a[m, 1], ..., a[m, n]}}
```

or `matrixa=Array[a, {m, n}]`, the unique element of `matrixa` in the i th row and j th column is obtained with `matrixa[[i, j]]`.

□ Example:

In the previous examples `mb` was defined to be the matrix $\begin{bmatrix} 10 & -6 & -9 \\ 6 & -5 & -7 \\ -10 & 9 & 12 \end{bmatrix}$.

`mb[[i, j]]` yields the (unique) number in the i th row and j th column of `mb`. The determinant of `mb` can be calculated with `Det[mb]`. Observe how various components of `mb` (rows, elements) can be extracted and how `mb` is placed in `MatrixForm`.

<pre>In[128]:= mb[[3]] Out[128]= {-10, 9, 12} In[129]:= mb[[1,3]] Out[129]= -9 In[130]:= MatrixForm[mb] Out[130]//MatrixForm= 10 -6 -9 6 -5 -7 -10 9 12 In[131]:= Det[mb] Out[131]= 6</pre>	<div style="border: 1px dashed black; padding: 5px; width: fit-content; margin: 10px auto;"> <p><i>Remember that elements of lists are extracted using double square brackets.</i></p> </div>	<pre>mb[[3]] yields the third element of mb; this is the same as the third row of the matrix. mb[[1,3]] yields the third element of the first element of mb; notice that this is the number in the first row and third column of the matrix. MatrixForm[mb] displays mb in traditional matrix form. The same result would be obtained by entering either mb // MatrixForm or TableForm[mb]. Det[mb] computes the determinant of mb.</pre>
--	---	--

■ Basic Computations with Matrices and Vectors:

Mathematica performs all of the usual operations on matrices. Matrix addition ($\mathbf{A+B}$), scalar multiplication ($k\mathbf{A}$), matrix multiplication ($\mathbf{A.B}$), and combinations of these operations are all possible. In addition, the transpose of a matrix \mathbf{A} is found with the built-in command `Transpose[A]`.

If \mathbf{A} and \mathbf{B} are $n \times n$ matrices satisfying $\mathbf{AB = BA = I}$. Then **\mathbf{B} is the inverse of \mathbf{A}** and \mathbf{B} is denoted by

\mathbf{A}^{-1} . Then $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$.

The inverse of a matrix \mathbf{A} , provided it exists, is found with the built-in command `Inverse[A]`.

Recall that if $\mathbf{A} = [a_{ij}]$ then the transpose of \mathbf{A} is denoted \mathbf{A}^t where $\mathbf{A}^t = [a_{ji}]$.

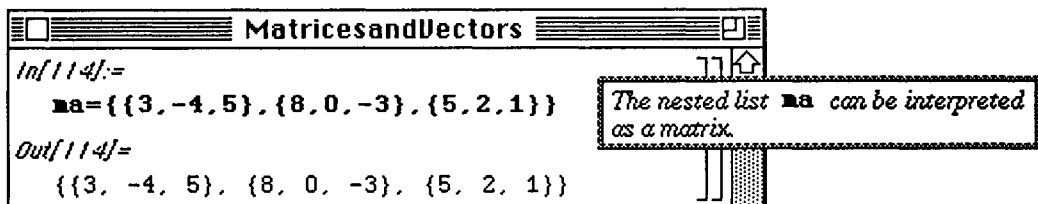
Equivalently, the transpose of A is obtained by interchanging the rows and columns of A .

□ Example:

Use *Mathematica* to define matrix \mathbf{ma} to be $\begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix}$ and matrix \mathbf{mb} to be $\begin{bmatrix} 10 & -6 & -9 \\ 6 & -5 & -7 \\ -10 & 9 & 12 \end{bmatrix}$

Compute (i) $\mathbf{ma}+\mathbf{mb}$; (ii) $\mathbf{mb}-4\mathbf{ma}$; (iii) the inverse of $\mathbf{ma} \cdot \mathbf{mb}$; and (iv) the transpose of $(\mathbf{ma}-2\mathbf{mb}) \cdot \mathbf{mb}$.

As described above, we enter \mathbf{ma} and \mathbf{mb} as nested lists where each element corresponds to a row of the matrix:



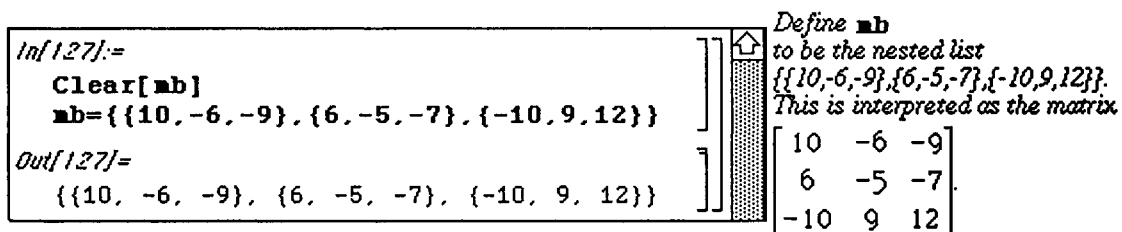
```

In[114]:=
  ma={{3,-4,5},{8,0,-3},{5,2,1}}
Out[114]=
  {{3,-4,5},{8,0,-3},{5,2,1}}

```

The nested list \mathbf{ma} can be interpreted as a matrix.

\mathbf{mb} is defined similarly:



```

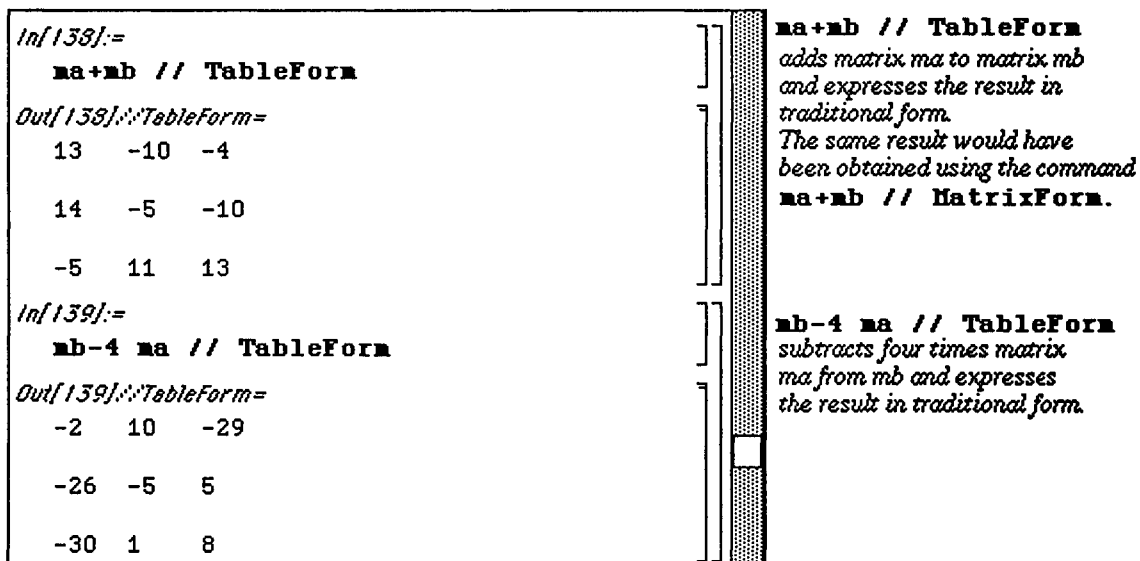
In[127]:=
  Clear[mb]
  mb={{10,-6,-9},{6,-5,-7},{-10,9,12}}
Out[127]=
  {{10,-6,-9},{6,-5,-7},{-10,9,12}}

```

Define \mathbf{mb} to be the nested list $\{\{10,-6,-9\},\{6,-5,-7\},\{-10,9,12\}\}$. This is interpreted as the matrix

$$\begin{bmatrix} 10 & -6 & -9 \\ 6 & -5 & -7 \\ -10 & 9 & 12 \end{bmatrix}$$

Matrices can be expressed in traditional matrix form using either the command **MatrixForm** or **TableForm**. The operations are performed and the resulting matrix is expressed in traditional matrix form:



```

In[138]:=
  ma+mb // TableForm
Out[138]//TableForm=
  13  -10  -4
  14  -5   -10
  -5  11   13

In[139]:=
  mb-4 ma // TableForm
Out[139]//TableForm=
  -2  10  -29
  -26 -5   5
  -30 1   8

```

$\mathbf{ma}+\mathbf{mb}$ // **TableForm** adds matrix \mathbf{ma} to matrix \mathbf{mb} and expresses the result in traditional form. The same result would have been obtained using the command $\mathbf{ma}+\mathbf{mb}$ // **MatrixForm**.

$\mathbf{mb}-4 \mathbf{ma}$ // **TableForm** subtracts four times matrix \mathbf{ma} from \mathbf{mb} and expresses the result in traditional form.

The screenshot shows the Mathematica interface with two input-output pairs. The first input is `In[142]:= Inverse[ma.mb] // TableForm`, and the output is a 3x3 matrix:

$$\begin{pmatrix} 59 & 53 & 167 \\ --- & --- & -(---) \\ 380 & 190 & 380 \\ \\ 223 & 92 & 979 \\ -(---) & -(---) & --- \\ 570 & 95 & 570 \\ \\ 49 & 18 & 187 \\ --- & -- & -(---) \\ 114 & 19 & 114 \end{pmatrix}$$
 A callout box explains: *first computes the product of matrix ma and mb and then find the inverse.*

The second input is `In[143]:= Transpose[(ma-2 mb).mb] // TableForm`, and the output is a 3x3 matrix:

$$\begin{pmatrix} -352 & -90 & 384 \\ \\ 269 & 73 & -277 \\ \\ 373 & 98 & -389 \end{pmatrix}$$
 A callout box explains: *first subtracts 2 times matrix mb from matrix ma, then multiplies the result by matrix mb and finally computes the transpose.*

All of the basic operations on matrices can be performed with *Mathematica*. These include the determinant, `Det[A]`, as shown earlier, as well as the computation of the eigenvalues and corresponding eigenvectors of the matrix.

Recall that a nonzero vector x is an **eigenvector** for the square matrix A means there exists

a scalar λ such that $Ax = \lambda x$. λ is called an **eigenvalue** (or **characteristic value**) for A .

The command `Eigenvalues[m]` gives a list of the eigenvalues of the square matrix m .

The command `Eigenvectors[m]` gives a list of the eigenvectors of the square matrix m .

Several examples are shown below. (Notice that by naming the matrix `ma`, the matrix operations involving `ma` are easier to perform.)

□ Example:

Use *Mathematica* to compute (i) the determinant of $\begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix}$;

(ii) numerical approximations of the eigenvalues; and

(iii) numerical approximations of the eigenvectors

In the previous example, *Mathematica* was used to define **ma** to be $\begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix}$

The screenshot shows a Mathematica notebook interface with the following content:

```

In[115]:=
MatrixForm[ma] displays ma in traditional matrix format.
Out[115]:=MatrixForm=
  3  -4  5
  8  0  -3
  5  2  1

In[116]:=
Det[ma]
Out[116]=
  190

In[117]:=
vals=Eigenvalues[ma]
Short[vals]
Out[117]:=Short=
  4 4
{- + <<2>>, <<1>>, - + <<2>>}
  3 3

In[118]:=
vals // N
Out[118]=
{6.27524, -1.13762 + 5.38363 I,
-1.13762 - 5.38363 I}

```

Annotations on the right side of the notebook window:

- Det[ma]** computes the determinant of **ma**.
- vals=Eigenvalues[ma]** computes the exact eigenvalues of **ma**.
- Short[vals]** displays a portion of the list **vals** on no more than one line.
- vals // N** computes numerical approximations of the exact values of the eigenvalues.

Sometimes the matrix in which each element is numerically approximated is more useful than the matrix in its original form. This is obtained below for the matrix **ma** with **N[ma]**. *Mathematica* can also be used to compute the eigenvectors of the matrix with **Eigenvectors[ma]**.

Notice that this command results in a list of eigenvectors as was the case with eigenvalues. Hence, individual eigenvectors can be extracted from the list of eigenvectors, `vector`, using `vector[[i]]` which gives the *i*th member of the list of eigenvectors. A similar command yields individual eigenvalues. (This method of extracting elements of lists was discussed earlier in Chapter 4.) Once obtained from the list of eigenvectors, certain operations can be performed with the extracted eigenvector. For example, the *i*th eigenvector can be multiplied by the matrix, `ma`. This is accomplished with the command

```
ma.vector[[i]].
```

The above command can be used with `N[vals[[j]]] vector[[i]]` to verify that the numerical approximation of the eigenvalue `vals[[j]]` corresponds to the eigenvector `vector[[i]]`. If this pair corresponds, then `ma.vector[[i]] = N[vals[[j]]] vector[[i]]` according to the definition given above (i.e., $Ax = \lambda x$).

Notice that the lists of eigenvalues and eigenvectors are not given in corresponding order. That is to say, the *i*th eigenvalue in the list of eigenvalues does not necessarily correspond with the *i*th eigenvector in the list of eigenvectors.

Examples of these operations are demonstrated below :

□ Example:

Compute numerical approximations of the eigenvectors of $\mathbf{ma} = \begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix}$.

MatricesandVectors

```

In[119]:=
  vectors=Eigenvectors[N[ma]]
Out[119]=
  {{-0.69216, -0.481495, -0.838594},
   {0.000471861 + 0.767318 I,
    1.51846 - 0.105255 I,
    0.388183 - 0.71867 I},
   {0.000471861 - 0.767318 I,
    1.51846 + 0.105255 I,
    0.388183 + 0.71867 I}}
In[120]:=
  vectors[[1]]
Out[120]=
  {-0.69216, -0.481495, -0.838594}
In[121]:=
  ma.vectors[[1]]
Out[121]=
  {-4.34347, -3.0215, -5.26238}
In[122]:=
  N[vals[[1]]] vectors[[1]]
Out[122]=
  {-4.34347, -3.0215, -5.26238}
  
```

Eigenvectors[N[ma]]
computes a numerical approximation of the eigenvectors of ma. The resulting list is named vectors.

Remember that elements of lists are obtained via double square brackets "[[]]"

To verify that vectors[[1]] is the eigenvector corresponding to the eigenvalue vals[[1]], it is necessary to verify that ma.vectors[[1]] = vals[[1]] vectors[[1]].

vectors[[1]] *gives the first element of the list vectors.*

ma.vectors[[1]] *performs the computation*

$$\begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} -0.692 \\ -0.481 \\ -0.839 \end{bmatrix}$$

N[vals[[1]]] vectors[[1]] *performs the computation*

$$6.27524 \begin{bmatrix} -0.692 \\ -0.481 \\ -0.839 \end{bmatrix}$$

Recall that the $n \times n$ identity matrix is the matrix $I = [b_{i,j}]$ where $b_{i,j} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases}$.

The command `IdentityMatrix[n]` yields the $n \times n$ identity matrix.

□ Example:

Compute the inverse, \mathbf{ma}^{-1} , of the matrix $\mathbf{ma} = \begin{bmatrix} 3 & -4 & 5 \\ 8 & 0 & -3 \\ 5 & 2 & 1 \end{bmatrix}$ and verify that $\mathbf{ma}^{-1}\mathbf{ma} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

The inverse of the matrix \mathbf{ma} is found with the command `Inverse[ma]` and is named `mai` for easier use. The matrix \mathbf{ma} is then multiplied by its inverse `mai` and placed in `TableForm` to verify that the identity matrix is obtained.

```

MatricesandVectors

In[123]:=
  mai=Inverse[ma]
Out[123]=
  3 7 6      23 11 49
  {--, --, --}, {-(---), -(---), ---},
  95 95 95    190 95 190

  8 13 16
  {--, -(---), --}
  95 95 95

In[124]:=
  mai.ma // TableForm
Out[124]::TableForm=
  1 0 0
  0 1 0
  0 0 1
  
```

Inverse[ma]
computes the inverse of the matrix `ma`; the inverse matrix is named `mai`.

mai.ma // TableForm
multiplies matrix `mai` by matrix `ma` and expresses the result as a table. Notice the same output would have been obtained by the command `mai.ma // MatrixForm`.

□ Example:

Define `matrixb` to be the matrix $\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}$. Compute (i) $\text{Det} \begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}$;

(ii) $\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^2$; (iii) $\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^3$; and (iv) $\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^4$.

MatricesandVectors

```
In[3]:=
matrixb={{-2,3,4,0},{-2,0,1,3},{-1,4,-6,5},
{4,8,11,-4}}
```

defines `matrixb` to be the 4 x 4 matrix

$$\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}$$

```
Out[3]=
{{-2, 3, 4, 0}, {-2, 0, 1, 3},
{-1, 4, -6, 5}, {4, 8, 11, -4}}
```

```
In[4]:=
Det[matrixb]
```

`Det[matrixb]` computes the determinant of `matrixb`.

```
Out[4]=
-855
```

```
In[9]:=
matrixb.matrixb // MatrixForm
```

computes the product of `matrixb` multiplied by `matrixb`. This is the same as computing

$$\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^2$$

```
Out[9]//MatrixForm=
-6    10   -29   29
15    22    19   -7
20    13    91  -38
-51   24   -86   95
```

Special attention must be given to the notation which must be used in taking the product of a matrix with itself. The following example illustrates how *Mathematica* interprets the expression `(matrixb)^3`. Usually, the matrix product

`matrixb matrixb matrixb` is represented as `(matrixb)^3`.

However, the command `(matrixb)^3` cubes each element of the matrix `matrixb`.

```
In[14]:=
matrixb.matrixb.matrixb // MatrixForm
```

```
Out[14]//MatrixForm=
 137  98  479  -231
-121  65  -109  189
-309 120  -871  646
 520 263 1381  -738
```

```
In[19]:=
(matrixb)^3 // MatrixForm
```

```
Out[19]//MatrixForm=
 -8  27  64  0
 -8  0  1  27
 -1  64 -216 125
 64 512 1331 -64
```

computes

$$\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^3$$

and expresses the result in traditional matrix form.

Notice that a different result is obtained when the command `(matrixb)^3` is executed. The command `(matrixb)^3` cubes each element of `matrixb`.

The built-in command `MatrixPower` may be used to compute products of matrices:

`MatrixPower[matrix,n]` computes $\underbrace{\text{matrix} \cdot \text{matrix} \cdot \dots \cdot \text{matrix}}_{n\text{-times}}$.

However, to illustrate *Mathematica*'s recursive abilities, we define a function `matrixpower` that performs the same calculations as the built-in function `MatrixPower`.

We define the function `matrixpower` by first defining the matrix with `matrixpower[a_, 1]` and then the matrix product `matrixpower[a_, n_] := a.matrixpower[a, n-1]`.

Hence, *Mathematica* computes the desired power of the matrix.

MatricesandVectors

```
In[26]:=
matrixpower[a_, 1] := a
matrixpower[a_, n_] := a.matrixpower[a, n-1]
```

Using Mathematica's recursion ability, for a given matrix a, define matrixpower[a, 1]=a and define, for n a positive integer,

```
In[28]:=
matrixpower[matrixb, 3]
```

Then, matrixpower[a, n] computes a.a....a ntimes

```
Out[28]=
{{137, 98, 479, -231},
{-121, 65, -109, 189},
{-309, 120, -871, 646},
{520, 263, 1381, -738}}
```

matrixpower[matrixb, 3] computes

$$\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^3$$

```
In[29]:=
matrixpower[matrixb, 4] // MatrixForm
```

computes

$$\begin{bmatrix} -2 & 3 & 4 & 0 \\ -2 & 0 & 1 & 3 \\ -1 & 4 & -6 & 5 \\ 4 & 8 & 11 & -4 \end{bmatrix}^4$$

and expresses the result in matrix form.

```
Out[29] // MatrixForm =
-1873   479   -4769   3613
  977    713    2314  -1106
 3833   757   11216  -6579
-5899  1180  -14061  10646
```


■ 5.2 Linear Systems of Equations

■ Calculating Solutions of Linear Systems of Equations

To solve the system of linear equations $Ax=b$, where A is the coefficient matrix, b is a known vector and x is the unknown vector, we proceed in the usual manner: If A^{-1} exists, then $A^{-1}Ax = A^{-1}b$ so $x = A^{-1}b$.

Mathematica offers several commands for solving systems of linear equations, however, which do not depend on the computation of the inverse of A . These commands are discussed in the following examples.

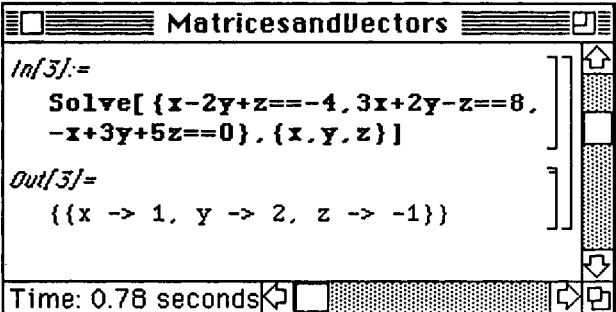
□ Example:

Solve the system of three equations
$$\begin{cases} x - 2y + z = -4 \\ 3x + 2y - z = 8 \\ -x + 3y + 5z = 0 \end{cases}$$
 for x , y , and z .

In order to solve an $n \times n$ system of equations (n equations and n unknown variables), the command `Solve[{eqn1, eqn2, ..., eqnn}, {var1, var2, ..., varn}]` is used. In other words, the equations as well as the variables are entered as lists. If one wishes to solve for all variables that appear in a system, the command `Solve[{eqn1, eqn2, ..., eqnn}]` attempts to solve `eqn1`, `eqn2`, ..., `eqnn` for all variables that appear in them. The system given above with 3 equations and 3 unknowns is solved below. (Remember that a double equals sign must be used in each equation.) The time required to perform the calculation is also displayed. The steps necessary to have the time displayed are given later in this section.

In this case, entering either `Solve[{x-2y+z==-4, 3x+2y-z==8, -x+3y+5z==0}]` or `Solve[{x-2y+z, 3x+2y, -x+3y+5z]=={-4, 8, 0}]` yield the same result.

Remark: Be sure to include the double equals signs between the left- and right-hand sides of each equation.



The screenshot shows the Mathematica interface with the following content:

```

In[3]:=
Solve[{x-2y+z==-4, 3x+2y-z==8,
-x+3y+5z==0}, {x, y, z}]

Out[3]=
{{x -> 1, y -> 2, z -> -1}}

Time: 0.78 seconds

```

To the right of the window, the command is repeated with a descriptive text and the system of equations:

```

Solve[{x-2y+z==-4, 3x+2y-z==8,
-x+3y+5z==0}, {x, y, z}]
solves the system of equations
{
x-2y+z=-4
3x+2y-z=8 in .78 seconds.
-x+3y+5z=0
}

```

In this case, Mathematica displays the time it takes to perform the calculation.

Another way to solve systems of equations is based on the matrix form of the system of equations, $Ax=b$. The matrix of coefficients in the previous example is entered as **matrixa** along with the vector of right-hand side values **vectorb**. After defining the vector of variables, **vectorx**, the system $Ax=b$ is solved explicitly with the command `Solve[matrixa.vectorb==vectorb,vectorx]`. Compare the computation times for each calculation.

```

In[15]:=
matrixa={{1,-2,1},{3,2,-1},{-1,3,5}};
vectorb={-4,8,0};
vectorx={x1,y1,z1}

Out[15]=
{x1, y1, z1}

In[16]:=
Solve[matrixa.vectorx==vectorb,vectorx]

Out[16]=
{{x1 -> 1, y1 -> 2, z1 -> -1}}

Time: 0.82 seconds

```

Define matrixa, vectorb, and vectorx.

solves the matrix equation for vectors in .82 seconds.

□ Example:

Next, the system
$$\begin{cases} 2x - 4y + z = -1 \\ 3x + y - 2z = 3 \\ -5x + y - 2z = 4 \end{cases}$$
 is solved in a similar manner. Notice that exact values are

given with **Solve**. This system takes longer to solve than the first example.

```

In[19]:=
Solve[{2x-4y+z== -1, 3x+y-2z==3,
-5x+y-2z==4}, {x,y,z}]

Out[19]=
{{x -> -(1/8), y -> -(15/56), z -> -(51/28)}}

Time: 1.10 seconds

```

Solve[{2x-4y+z== -1, 3x+y-2z==3, -5x+y-2z==4}, {x,y,z}] solves the system of equations

$$\begin{cases} 2x-4y+z=-1 \\ 3x+y-2z=3 \\ -5x+y-2z=4 \end{cases} \text{ for } x, y, \text{ and } z \text{ in}$$

1.10 seconds.

As before, consider the alternate approach to solving this system using matrices and vectors.

```

In[20]:=
  matrixa={{2,-4,1},{3,1,-2},
  {-5,1,-2}};
  vectorb={-1,3,4}

Out[20]=
  {-1, 3, 4}

```

Be sure to nest brackets appropriately.

*matrixa = {{2, -4, 1}, {3, 1, -2}, {-5, 1, -2}};
vectorb = {-1, 3, 4} defines matrixa to be $\begin{bmatrix} 2 & -4 & 1 \\ 3 & 1 & -2 \\ -5 & 1 & -2 \end{bmatrix}$ and vectorb to be $\begin{bmatrix} -1 \\ 3 \\ 4 \end{bmatrix}$.*

Time: 0.32 seconds

Notice that executing the command takes .32 seconds.

The command

LinearSolve[A, b]

calculates the solution x of the system $Ax=b$.

Comparing the computation times for each, **LinearSolve** performs the task more quickly (.57 secs.) than does **Solve**. If the time needed to enter the matrix A and vector b (.32 secs.) is considered, however, the total time (.89 secs.) for this calculation was slightly larger than the original (.78 secs.).

```

In[21]:=
  LinearSolve[matrixa, vectorb]

Out[21]=
  {(-), -(--), -(--)}
  1    15    51
  8    56    28

```


*solves the (matrix) equations
matrixa x= vectorb
for x.
The same result is obtained by executing the
command
Inverse[matrixa].vectorb*

Time: 0.57 seconds

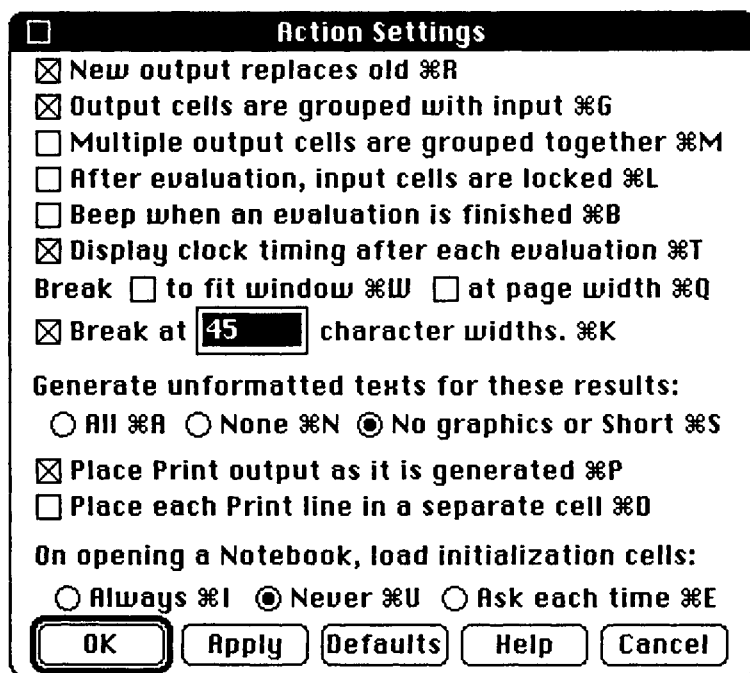
In this case, the calculation takes only .57 seconds.

□ **To Display Time After Each Evaluations:**

In the above examples, the time to perform each calculation is displayed instead of the usual thermometer indicating memory used. To display the time clock after each evaluation, proceed as follows:

- 1) Use the Mouse to move the cursor to **Edit** and select **Settings**; then select **Action**;
- 2) The **Action Settings** window will appear;
- 3) Click the box **Display clock timing after each evaluation**; and
- 4) Click the  button.

Action Settings in Version 1.2; the Action Settings in Version 2.0 are the same.



When using the **Solve** command, the equations may be entered in several different ways as the following example shows. For example, if **equations** is a list of equations and **variables** is a list of variables, then *Mathematica* attempts to solve **equations** in terms of **variables** when the command **Solve[*equations*, *variables*]** is entered; *Mathematica* attempts to solve **equations** in terms of all variables that appear in **equations** when **Solve[*equations*]** is entered:

Example:

Solve the system of equations $\begin{cases} 4x_1 + 5x_2 - 5x_3 - 8x_4 - 2x_5 = 5 \\ 7x_1 + 2x_2 - 10x_3 - x_4 - 6x_5 = -4 \\ 6x_1 + 2x_2 + 10x_3 - 10x_4 + 7x_5 = -7 \\ -8x_1 - x_2 - 4x_3 + 3x_5 = 5 \\ 8x_1 - 7x_2 - 3x_3 + 10x_4 + 5x_5 = 7 \end{cases}$ for $x_1, x_2, x_3, x_4,$ and x_5 .

```
In[22]:=
Solve[{4x[1]+5x[2]-5x[3]-8x[4]-2x[5],
7x[1]+2x[2]-10x[3]-x[4]-6x[5],
6x[1]+2x[2]+10x[3]-10x[4]+7x[5],
-8x[1]-x[2]-4x[3]+3x[5],
8x[1]-7x[2]-3x[3]+10x[4]+5x[5]}==
{5, -4, -7, 5, 7}]

Out[22]:=
      1245          113174
{{x[1] -> ----, x[2] -> ----,
      6626          9939

      7457          38523
x[3] -> -(-----), x[4] -> ----,
      9939          6626

      49327
x[5] -> ----}}
      9939

Time: 1.90 seconds
```

solves the system of equations

$$\begin{cases} 4x_1 + 5x_2 - 5x_3 - 8x_4 - 2x_5 = 5 \\ 7x_1 + 2x_2 - 10x_3 - x_4 - 6x_5 = -4 \\ 6x_1 + 2x_2 + 10x_3 - 10x_4 + 7x_5 = -7 \\ -8x_1 - x_2 - 4x_3 + 3x_5 = 5 \\ 8x_1 - 7x_2 - 3x_3 + 10x_4 + 5x_5 = 7 \end{cases}$$

for $x_1, x_2, x_3, x_4,$ and x_5 .

After defining **matrixa** to be the matrix $\begin{bmatrix} 4 & 5 & -5 & -8 & -2 \\ 7 & 2 & -10 & -1 & -6 \\ 6 & 2 & 10 & -10 & 7 \\ -8 & -1 & -4 & 0 & 3 \\ 8 & -7 & -3 & 10 & 5 \end{bmatrix}$ and **t2** to be vector $\begin{bmatrix} 5 \\ -4 \\ -7 \\ 5 \\ 7 \end{bmatrix}$,

LinearSolve is used to solve the same system much faster:

```

In[21]:=
matrixa={{4,5,-5,-8,-2},{7,2,-10,-1,-6},
         {6,2,10,-10,7},{-8,-1,-4,0,3},
         {8,-7,-3,10,5}}
t2={5,-4,-7,5,7}
xarray=Array[x,{5}];

In[24]:=
LinearSolve[matrixa,t2]

Out[24]=
1245 113174 7457 38523 49327
{----, -----, -(----), -----, -----}
6626 9939 9939 6626 9939

Time: 0.88 seconds

```

■ **Application:** Characteristic and Minimal Polynomials

The **characteristic polynomial** of the $n \times n$ matrix A is the polynomial

$$p_A(x) = \text{Det}[xI_n - A], \text{ where } I_n \text{ is the } n \times n \text{ identity matrix}$$

It is well-known that the eigenvalues of A are the roots of the characteristic polynomial of A .

The **trace** of an $n \times n$ matrix $A = [a_{i,j}]$ is $a_{1,1} + a_{2,2} + \dots + a_{n,n} = \sum_{k=1}^n a_{k,k}$.

If A is a matrix with non-zero determinant and characteristic polynomial

$$p_A(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + c_0, \text{ then } \text{Det}[A] = (-1)^n c_0$$

$$\text{and Trace}[A] = a_{1,1} + \dots + a_{n,n} = \sum_{k=1}^n a_{k,k} = -c_{n-1}.$$

Let $p_A(x)$ be the characteristic polynomial of A and let

$$p_A(x) = (p_1(x))^{n_1} (p_2(x))^{n_2} \dots (p_m(x))^{n_m} \text{ be the factorization of } p_A(x).$$

The **minimal polynomial $q(x)$ of A** is the monic polynomial of least degree satisfying $q(A)=0$. It is well-known

that $p_1(x)p_2(x) \dots p_m(x)$ divides $q(x)$; hence, $p_A(A) = 0$.

□ Example:

Find the trace, characteristic polynomial, and minimal polynomial of the matrix $\begin{bmatrix} 0 & 6 & 3 \\ -2 & -8 & -2 \\ 0 & 0 & -2 \end{bmatrix}$

The process begins by entering the matrix, `matrixa`, and then the associated matrix, `assoca`, where `assoca=x IdentityMatrix[4]-matrixa`.

The characteristic polynomial is then determined using `Det[assoca]`.

MatrixOperations

```

In[80]:=
Clear[x,matrixa]
matrixa={{0,6,3},{-2,-8,-2},{0,0,-2}}
Out[80]=
{{0, 6, 3}, {-2, -8, -2}, {0, 0, -2}}
In[81]:=
assoca=x IdentityMatrix[3]-matrixa
Out[81]=
{{x, -6, -3}, {2, 8 + x, 2}, {0, 0, 2 + x}}
In[82]:=
MatrixForm[assoca]
Out[82]:=MatrixForm=
x      -6      -3
2      8 + x  2
0      0      2 + x
In[83]:=
Det[assoca]
Out[83]=
24 + 28 x + 10 x2 + x3

```

Begin by clearing prior definitions of x and matrixa; then define matrixa to be the matrix

$$\begin{bmatrix} 0 & 6 & 3 \\ -2 & -8 & -2 \\ 0 & 0 & -2 \end{bmatrix}$$

IdentityMatrix[3] is the three-by-three identity matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

assoca is the three-by-three matrix

$$X \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 6 & 3 \\ -2 & -8 & -2 \\ 0 & 0 & -2 \end{bmatrix}$$

computes the determinant of the three-by-three matrix assoca.

The characteristic polynomial is then easily factored using `Factor[Det[assoca]]`.

This yields $(2+x)^2(6+x)$.

Since the minimal polynomial divides the characteristic polynomial, the minimal polynomial must either be

$(2+x)^2(6+x)$ or $(2+x)(6+x)$.

Now, in order to determine which is the minimal polynomial, the definition given earlier must be employed. Since this definition involves substituting **matrixa** into these two polynomials to see which gives the zero matrix (the matrix of all zeros), determining the powers of **matrixa** is necessary. Therefore, the function **matrixpower** discussed earlier is redefined and used to square and cube **matrixa**. For easier use, the 3x3 identity matrix is named **ident**, the square of **matrixa** is called **matrixs**, and the cube of **matrixa** is called **matrixc**.

Note that the same result can be accomplished using the built-in function **MatrixPower**.

The screenshot shows a Mathematica notebook window titled "MatrixOperations". The notebook contains several input and output cells, with callout boxes providing additional information.

Cell 1:
 In[84]:= **Factor[Det[assoca]]**
 Out[84]= $(2 + x)^2 (6 + x)$
 Callout: *Factors Det[assoca]] which is the polynomial $x^3 + 10x^2 + 28x + 24$. The minimal polynomial of **matrixa** divides the characteristic polynomial; moreover, we can conclude that the minimal polynomial is either $(2+x)(6+x)$ or $(2+x)^2(6+x)$.*

Cell 2:
 In[85]:= **matrixpower[a_, 1] := a**
matrixpower[a_, n_] := a.matrixpower[a, n-1]

Cell 3:
 In[86]:= **ident = IdentityMatrix[3]**
 Out[86]= $\{\{1, 0, 0\}, \{0, 1, 0\}, \{0, 0, 1\}\}$
 Callout: *Name the 3-by-3 identity matrix **ident**.*

Cell 4:
 In[87]:= **matrixs = matrixpower[matrixa, 2]**
 Out[87]= $\{\{-12, -48, -18\}, \{16, 52, 14\}, \{0, 0, 4\}\}$
 Callout: *computes $\begin{bmatrix} 0 & 6 & 3 \\ -2 & -8 & -2 \\ 0 & 0 & -2 \end{bmatrix}^2$ and names the result **matrixs**.*

Cell 5:
 In[88]:= **matrixc = matrixpower[matrixa, 3]**
 Out[88]= $\{\{96, 312, 96\}, \{-104, -320, -84\}, \{0, 0, -8\}\}$
 Callout: *computes $\begin{bmatrix} 0 & 6 & 3 \\ -2 & -8 & -2 \\ 0 & 0 & -2 \end{bmatrix}^3$ and names the result **matrixc**.*

Substitution of `matrixa` into the possible minimal polynomial $(2+x)(6+x)$ is done as follows:

```
(2 ident + matrixa).(6 ident + matrixa)
```

Notice that since `matrixa` is a matrix as opposed to a scalar, then each constant in the polynomial must be converted to a matrix by multiplying by the identity matrix. Otherwise, the command would not be defined. Substitution into the other polynomial is done in a similar manner.

Since substitution into the second polynomial yields the zero matrix, the minimal polynomial is

$$(2+x)^2(6+x) \text{ or } x^3 + 10x^2 + 28x + 24 .$$

Since $\begin{bmatrix} 0 & 0 & 6 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}$ is not the zero matrix, $(2+x)(6+x)$ is not the minimal polynomial for `matrixa`. Since this is the zero matrix, $(2+x)^2(6+x)$ is the minimal polynomial for `matrixa`.

The properties of the minimal polynomial can now be verified. Notice that the order of this polynomial is three. Since the general formula for the minimal polynomial of order three is

$$q(x) = x^3 + c_2x^2 + c_1x + c_0 \text{ we have in this case, } c_0 = 24, c_1 = 28, \text{ and } c_2 = 10.$$

Hence, the trace of `A` can be computed with following formula:

$$\text{Trace}[A] = a_{1,1} + a_{2,2} + a_{3,3} = \sum_{k=1}^3 a_{k,k} = -c_{3-1} = -c_2 = -10$$

as well as the determinant: $\text{Det}[A] = (-1)^3 c_0 = -24.$

These results agree with the calculations shown below using *Mathematica* :

□ Example:

Find the trace, characteristic polynomial, and minimal polynomial of the matrix

$$\begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}$$

This problem is solved in a manner similar to that of the previous example. First, the matrix is entered and named **matrixc**. Then, the associated matrix **assocc** is defined. Note that in this case the 4x4 identity must be used. The characteristic polynomial is then found with **Det[assocc]** and called **char** for later use. From **char**, the numerical approximation of the roots of the characteristic polynomial (the eigenvalues) of **matrixc** can be determined with **NRoots[char==0,x]**. In this case, the characteristic polynomial is irreducible.

```

MoreMatrices
In[15]:=
matrixc={{2,-3,1,2},{3,7,-5,2},
{0,1,9,-5},{-2,3,4,3}}
Out[15]=
{{2,-3,1,2},{3,7,-5,2},
{0,1,9,-5},{-2,3,4,3}}
In[16]:=
assocc=x IdentityMatrix[4]-matrixc
Out[16]=
{{-2+x,3,-1,-2},{-3,-7+x,5,-2},
{0,-1,-9+x,5},{2,-3,-4,-3+x}}
In[17]:=
char=Det[assocc]
Out[17]=
1665 - 848 x + 181 x2 - 21 x3 + x4
In[18]:=
NRoots[char==0,x]
Out[18]=
x == 3.59859 - 5.14335 I ||
x == 3.59859 + 5.14335 I ||
x == 4.58305 || x == 9.21977

```

Define **matrixc**
to be the matrix

$$\begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}$$

Then define **assocc** to be the
matrix

$$x \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}$$

The characteristic polynomial of
matrixc is the determinant of
assocc.

Although Mathematica can
compute the exact roots of the
characteristic polynomial, it is
much faster to obtain numerical
approximations of them.

A numerical approximation of the eigenvalues can also be found with **Eigenvalues**[**N**[**matrixc**]]. This is demonstrated below.

The method by which **assoc** is raised to a power is slightly different in this example. Instead of making use of the user-defined function **matrixpower** seen in the example above, we choose to take advantage of the built-in *Mathematica* function **MatrixPower**[**matrix**,**n**]. (Note the capital letters.) This function determines the matrix obtained when **matrix** is raised to the power **n** as did **matrixpower**. The matrices obtained by using **MatrixPower** to raise **matrixc** to the powers 2, 3, and 4 are necessary in determining the minimal polynomial. These are calculated and named **matrix2**, **matrix3**, and **matrix4**, respectively. (Note that only the output of the last command in the second input cell is displayed.) To see that the characteristic polynomial is the minimal polynomial, **matrixc** is substituted into the characteristic polynomial, **char**, to yield the zero matrix.

MoreMatrices

In[19]:=
Eigenvalues[**N**[**matrixc**]] *computes numerical approximations of the eigenvalues of matrixc.*

Out[19]=
 {3.59859 + 5.14335 I, 3.59859 - 5.14335 I,
 4.58305, 9.21977}

In[20]:=
ident4=**IdentityMatrix**[4]
matrix2=**MatrixPower**[**matrixc**,2]
matrix3=**MatrixPower**[**matrixc**,3]
matrix4=**MatrixPower**[**matrixc**,4]
MatrixForm[**matrix4**]

Out[20]//MatrixForm=

64	-646	2947	-2974
-212	3192	-4330	5611
692	-2483	1711	-2688
1341	1331	-1337	324

matrix4 is

$$\begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}^4$$

ident4 is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matrix2 is

$$\begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}^2$$

matrix3 is

$$\begin{bmatrix} 2 & -3 & 1 & 2 \\ 3 & 7 & -5 & 2 \\ 0 & 1 & 9 & -5 \\ -2 & 3 & 4 & 3 \end{bmatrix}^3$$

In[21]:=
1665 ident4-**848 matrixc**+
181 matrix2-
21 matrix3+**matrix4**

Out[21]=
 {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0},
 {0, 0, 0, 0}}

We evaluate the characteristic polynomial when x is the matrix matrixc. Since the characteristic polynomial is irreducible, the characteristic polynomial is the minimal polynomial.

The order of the minimal polynomial is 4. Therefore, since $c_0 = 1665$, $c_1 = -848$, $c_2 = 181$, and $c_3 = 21$, we have $\text{Trace}[A] = -c_{4-1} = -c_3 = -21$.

■ Application: Maxima and Minima Using Linear Programming

We call the linear programming problem of the form:

Minimize $Z = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$, subject to the restrictions
function

$$\text{inequalities} \begin{cases} a_{1,1} x_1 + a_{1,2} x_2 + \dots + a_{1,n} x_n \geq b_1 \\ a_{2,1} x_1 + a_{2,2} x_2 + \dots + a_{2,n} x_n \geq b_2 \\ \vdots \\ a_{m,1} x_1 + a_{m,2} x_2 + \dots + a_{m,n} x_n \geq b_m \end{cases}, \text{ and } x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0$$

the **standard form** of the linear programming problem.

The *Mathematica* command

ConstrainedMin[function, {inequalities}, {variables}] solves the standard form of the linear programming problem.

Similarly, the *Mathematica* command

ConstrainedMax[function, {inequalities}, {variables}] solves the linear programming problem

Maximize $Z = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$, subject to the restrictions
function

$$\text{inequalities} \begin{cases} a_{1,1} x_1 + a_{1,2} x_2 + \dots + a_{1,n} x_n \geq b_1 \\ a_{2,1} x_1 + a_{2,2} x_2 + \dots + a_{2,n} x_n \geq b_2 \\ \vdots \\ a_{m,1} x_1 + a_{m,2} x_2 + \dots + a_{m,n} x_n \geq b_m \end{cases}, \text{ and } x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0.$$

□ Example:

Maximize $z(x_1, x_2, x_3) = 4x_1 - 3x_2 + 2x_3$ subject to the constraints

$$3x_1 - 5x_2 + 2x_3 \leq 60,$$

$$x_1 - x_2 + 2x_3 \leq 10,$$

$$x_1 + x_2 - x_3 \leq 20, \text{ and } x_1, x_2, x_3 \text{ all non-negative}$$

In order to solve a linear programming problem with *Mathematica*, the variables {**x1**, **x2**, **x3**} and objective function **z** {**x1**, **x2**, **x3**} must first be defined. In an effort to limit the amount of typing required to complete the problem, the set of inequalities is assigned the name **ineqs** while the set of variables is called **vars**.

Notice that the symbol "<=", obtained by typing the "<" key and then the "=" key, represents "less than or equal to" and is used in **ineqs**. Hence, the maximization problem is solved with the command

ConstrainedMax[z {**x1**, **x2**, **x3**}, **ineqs**, **vars**].

The solution gives the maximum value of z subject the given constraints as well as the values of x_1 , x_2 , and x_3 which maximize z . These steps are shown below :

```

In[44]:=
Clear[x1, x2, x3, z, ineqs, vars]
vars={x1, x2, x3}
z[x1_, x2_, x3_]=4x1-3x2+2x3

Out[44]=
4 x1 - 3 x2 + 2 x3

In[45]:=
ineqs={3x1-5x2+x3 <= 60,
x1-x2+2x3 <=10, x1+x2-x3<=20};

In[46]:=
ConstrainedMax[z[x1, x2, x3],
ineqs, vars]

Out[46]=
{45, {x1 -> 15, x2 -> 5, x3 -> 0}}

```

First clear all prior definitions of variables to be used in the problem. Then define the function z to be maximized.

To avoid repeatedly typing the same thing, we name the constraints **ineqs**.

Finally, use the Mathematica function **ConstrainedMax** to see that the maximum value is 45 when $x_1=15$, $x_2=5$, and $x_3=0$.

Minimization problems are solved in a similar manner. Consider the following :

□ Example:

Minimize $z(x_1, x_2, x_3) = 4x_1 - 3x_2 + 2x_3$ subject to the constraints

$$3x_1 - 5x_2 + 2x_3 \leq 60,$$

$$x_1 - x_2 + 2x_3 \leq 10,$$

$$x_1 + x_2 - x_3 \leq 20, \text{ and } x_1, x_2, x_3 \text{ all non-negative}$$

After clearing all previously used names of functions and variable values, the variables, objective function, and set of constraints for this problem are defined and entered as they were in the first example. By using

ConstrainedMin[z[x1, x2, x3], ineqs, vars]

the minimum value of the objective function is obtained as well as the variable values which give this minimum.

```

In[48]:=
Clear[x1, x2, x3, z, ineqs, vars]
vars={x1, x2, x3}
z[x1_, x2_, x3_]=4x1-3x2+2x3

Out[48]=
4 x1 - 3 x2 + 2 x3

In[49]:=
ineqs={3x1-5x2+x3 <= 60,
x1-x2+2x3 <=10, x1+x2-x3<=20};

In[50]:=
ConstrainedMin[z[x1, x2, x3],
ineqs, vars]

Out[50]=
{-90, {x1 -> 0, x2 -> 50, x3 -> 30}}

```

We proceed exactly as before except that the final step uses the command **ConstrainedMin**.

Hence, we conclude that the minimum value is -90 which occurs when $x_1=0$, $x_2=50$, and $x_3=30$.

■ The Dual Problem

Given the standard form linear programming problem

$$\text{Minimize } Z = \sum_{j=1}^n c_j x_j \text{ subject to the constraints } \sum_{j=1}^n a_{ij} x_j \geq b_i \text{ for } i=1, 2, \dots, m$$

and $x_j \geq 0$ for $j=1, 2, \dots, n$, the **dual problem** is

$$\text{Maximize } Y = \sum_{i=1}^m b_i y_i \text{ subject to the constraints } \sum_{i=1}^m a_{ij} y_i \leq c_j \text{ for } j=1, 2, \dots, n$$

and $y_i \geq 0$ for $i=1, 2, \dots, m$.

Similarly, for the problem

Maximize $Z = \sum_{j=1}^n c_j x_j$ subject to the constraints $\sum_{j=1}^n a_{i,j} x_j \leq b_i$ for $i=1, 2, \dots, m$
 and $x_j \geq 0$ for $j=1, 2, \dots, n$, the **dual problem** is

Minimize $Y = \sum_{i=1}^m b_i y_i$ subject to the constraints $\sum_{i=1}^m a_{i,j} y_i \geq c_j$ for $j=1, 2, \dots, n$
 and $y_i \geq 0$ for $i=1, 2, \dots, m$.

□ **Example:**

Maximize $Z = 6x_1 + 8x_2$ subject to the constraints $5x_1 + 2x_2 \leq 20$, $x_1 + 2x_2 \leq 10$, $x_1 \geq 0$,
 and $x_2 \geq 0$.

State the dual problem and find its solution.

First, the original (primal) problem is solved. The objective function for this problem is represented by **zx** while that of the dual is given by **zy**. The set of variables $\{x[1], x[2]\}$ of the primal are called **valsx**. Similarly, those of the dual $\{y[1], y[2]\}$ are assigned the name **valsy**. Finally, the set of inequalities for the primal and dual are **ineqsx** and **ineqsy**, respectively. Using the command **ConstrainedMax[zx, ineqsx, {x[1], x[2]}]**, the maximum value of **zx** is found to be 45.

```

LinearProgramming
In[52]:=
Clear[zx, zy, x, y, valsx, valsy, ineqsx, ineqsy]

In[53]:=
zx=6x[1]+8x[2];
ineqsx={5x[1]+2x[2] <=20, x[1]+2x[2]<=10}

Out[53]=
{5 x[1] + 2 x[2] <= 20, x[1] + 2 x[2] <= 10}

In[54]:=
ConstrainedMax[zx, ineqsx, {x[1], x[2]}]

Out[54]=
          5          15
{45. {x[1] -> --, x[2] -> --}}
          2           4
    
```

We begin by clearing all prior definitions of the objects we will use in this example

*We name the functions we wish to maximize **zx** and we name the constraints **ineqs**.*

The maximum value is 45 which occurs when $x[1]$ is $5/2$ and when $x[2]$ is $15/4$.

Since in this problem, we have $c_1 = 6$, $c_2 = 8$, $b_1 = 20$, and $b_2 = 10$, the dual problem is
 Minimize $Z = 20y_1 + 10y_2$ subject to the constraints $5y_1 + y_2 \geq 6$, $2y_1 + 2y_2 \geq 8$, $y_1 \geq 0$,
 and $y_2 \geq 0$.

The dual is solved in a similar fashion by defining the objective function zy and the collection of inequalities $ineqsy$. The minimum value obtained by zy subject to the constraints $ineqsy$ is 45 which agrees with the result of the primal and is found with

`ConstrainedMin[zy, ineqsy, {y[1], y[2]}]`.

```

LinearProgramming

In[55]:=
  zy=20y[1]+10y[2]
  ineqsy={5y[1]+y[2] >=6, 2y[1]+2y[2] >=8}

Out[55]=
  {5 y[1] + y[2] >= 6, 2 y[1] + 2 y[2] >= 8}

In[56]:=
  ConstrainedMin[zy, ineqsy, {y[1], y[2]}]

Out[56]=
  {45, {y[1] -> 1/2, y[2] -> 7/2}}

```

To solve the dual problem, we proceed the same way except we use the command **ConstrainedMin**.

Notice that the minimum value of the dual is the same as the maximum value of the original!

Of course, linear programming models can involve numerous variables. Consider the following : Given the standard form linear programming problem

Minimize $Z = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$, subject to the restrictions

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n \geq b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n \geq b_2$$

⋮

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n \geq b_m, \text{ and } x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0$$

let $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, $b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$, $c = [c_1, \dots, c_n]$, and A denote the $m \times n$ matrix $A = [a_{i,j}]$.

Then the standard form of the linear programming problem is equivalent to find the vector x that maximizes the number $Z = c \cdot x$ subject to the restrictions $A \cdot x \geq b$ and $x \geq 0$. The **dual problem** of Maximize the number $Z = c \cdot x$ subject to the restrictions $A \cdot x \geq b$ and $x \geq 0$ is Minimize the number $Y = y \cdot b$ subject to the restrictions $y \cdot A \leq c$ and $y \geq 0$.

The *Mathematica* command `LinearProgramming[c, A, b]` finds the vector x which minimizes the quantity $Z = c \cdot x$ subject to the restrictions $A \cdot x \geq b$ and $x \geq 0$. This command does not yield the minimum value of Z as did `ConstrainedMin` and `ConstrainedMax`. This value must be determined from the resulting vector.

□ Example:

Maximize $Z = 5x_1 - 7x_2 + 7x_3 + 5x_4 + 6x_5$ subject to the constraints

$$2x_1 + 3x_2 + 3x_3 + 2x_4 + 2x_5 \leq 10,$$

$$6x_1 + 5x_2 + 4x_3 + x_4 + 4x_5 \leq 30,$$

$$-3x_1 - 2x_2 - 3x_3 - x_4 \leq -5,$$

$$-x_1 - x_2 - x_4 \leq -10, \text{ and } x_i \geq 0 \text{ for } i = 1, 2, 3, 4, 5.$$

For this problem, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$, $b = \begin{bmatrix} 10 \\ 30 \\ -5 \\ -10 \\ 0 \end{bmatrix}$, $c = [5, -7, 7, 5, 6]$, and $A = \begin{bmatrix} 2 & 3 & 3 & 2 & 2 \\ 6 & 5 & 4 & 1 & 4 \\ -3 & -2 & -3 & -4 & 0 \\ -1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

Clearly, *Mathematica's* ability to perform matrix algebra will be advantageous in the completion of this type of problem. First, the vectors \mathbf{c} and \mathbf{b} are entered.

Remark: Notice that *Mathematica* does NOT make a distinction between row and column vectors; it "interprets" the vector correctly and consequently performs the calculation correctly

The matrix **A** is entered and named **matrixa**.

A helpful function which can be used in this problem is **zerovec = Table[0, {5}]** which creates a list of five zeros. This can be used instead of typing a vector made up of 5 zeros and is used in defining **matrixa** below. In general, the command **Table[expression, {n}]** produces a list of **n** copies of **expression**.

```

LinearProgramming
In[84]:=
Clear[matrixa, x, y, c, b]
c={5, -7, 7, 5, 6}
b={10, 30, -5, -10, 0}

Out[84]=
{10, 30, -5, -10, 0}

In[85]:=
matrixa={{2, 3, 3, 2, 2}, {6, 5, 4, 1, 4},
{-3, -2, -3, -4, 0}, {-1, -1, 0, -1, 0}, {0, 0, 0, 0, 0}}

Out[85]=
{{2, 3, 3, 2, 2}, {6, 5, 4, 1, 4},
{-3, -2, -3, -4, 0}, {-1, -1, 0, -1, 0},
{0, 0, 0, 0, 0}}

In[86]:=
zerovec=Table[0, {5}]

Out[86]=
{0, 0, 0, 0, 0}

In[87]:=
{{2, 3, 3, 2, 2}, {6, 5, 4, 1, 4},
{-3, -2, -3, -4, 0}, {-1, -1, 0, -1, 0}, zerovec}

Out[87]=
{{2, 3, 3, 2, 2}, {6, 5, 4, 1, 4},
{-3, -2, -3, -4, 0}, {-1, -1, 0, -1, 0},
{0, 0, 0, 0, 0}}

```

*Begin by clearing all prior definitions of objects that will be used in this example. Then define **c**, **b** and **matrixa**.*

*The command **Table[0, {5}]** creates a list of five zeros.*

*Hence, instead of typing **{0, 0, 0, 0, 0}** when defining **matrixa** one could have typed **zerovec**.*

Another useful *Mathematica* command is `Array[x, n]` which creates the list of n elements $\{x[1], x[2], \dots, x[n]\}$. This command is used below to define the list of variables `xvec`. Similarly, the command `Table[x[i], {i, 1, n}]` yields the same list. These variables must be defined before attempting to solve this linear programming problem.

```

In[88]:=
  xvec=Array[x, 5]
Out[88]=
  {x[1], x[2], x[3], x[4], x[5]}
In[89]:=
  Table[x[i], {i, 1, 5}]
Out[89]=
  {x[1], x[2], x[3], x[4], x[5]}
  
```

Notice that the command `Array[x, 5]` yields the same as the command `Table[x[i], {i, 1, 5}]`.

We will see that using indexed variables can save us considerable typing.

After entering the objective function coefficients with the vector `c`, the matrix of coefficients from the inequalities with `matrixa`, and the right-hand side values found in `b`; the problem is solved with `LinearProgramming[c, matrixa, b]`. The solution is called `xvec`. Hence, the maximum value of the objective function is obtained by evaluating the objective function at the variable values which yields a maximum. Since these values are found in `xvec`, the maximum is determined with the product of the vector `c` and the vector `xvec`. (Recall that this product is entered as `c.xvec`.) This value is found to be $35/4$.

```

LinearProgramming
In[90]:=
  xvec=LinearProgramming[c, matrixa, b]
Out[90]=
  5      35
  {0, --, 0, 0, --}
  2      8
In[91]:=
  c.xvec
Out[91]=
  35
  --
  4
  
```

solves the linear programming problem.

Thus, the maximum value of z subject to the given constraints occurs when $x[1]=0$, $x[2]=5/2$, $x[3]=0$, $x[4]=0$, and $x[5]=35/8$.

`c.xvec` computes the maximum value which is $35/4$.

State the dual problem. What is its solution?

Since the dual of the problem is Minimize the number $Y=y.b$ subject to the restrictions $y.A \leq c$ and $y \geq 0$, we use *Mathematica* to calculate $y.b$ and $y.A$:

Remark: Notice that *Mathematica* does NOT make a distinction between row and column vectors; it interprets the vector correctly and consequently performs the calculation properly.

A list of the dual variables $\{y[1], y[2], y[3], y[4], y[5]\}$ is created with `Array[y, 5]`. This list includes 5 elements since there are five constraints in the original problem. The objective function of the dual problem is, therefore, found with `yvec.b`, and the left-hand sides of the set of inequalities are given with `yvec.matrixa`.

The screenshot shows a Mathematica notebook with the following content:

```

In[92]:=
yvec=Array[y, 5]
Out[92]=
{y[1], y[2], y[3], y[4], y[5]}

In[93]:=
yvec.b
Out[93]=
10 y[1] + 30 y[2] - 5 y[3] - 10 y[4]

In[94]:=
yvec.matrixa
Out[94]=
(2 y[1] + 6 y[2] - 3 y[3] - y[4],
 3 y[1] + 5 y[2] - 2 y[3] - y[4],
 3 y[1] + 4 y[2] - 3 y[3],
 2 y[1] + y[2] - 4 y[3] - y[4],
 2 y[1] + 4 y[2])

```

Annotations in the notebook:

- yvec is the list {y[1], y[2], y[3], y[4], y[5]}.** (points to the output of In[92])
- We will interpret this as the function we wish to minimize.* (points to the output of In[93])
- We can interpret these results to help us state the dual problem.* (points to the output of In[93])
- Remember that the solution (minimum value) of the dual is the same as the solution (maximum value) of the original problem.* (points to the output of In[93])
- We will interpret this as our constraints.* (points to the output of In[94])

Hence, we may state the dual problem as follows:

Minimize $Y = 10y_1 + 30y_2 - 5y_3 - 10y_4$ subject to the constraints

$$2y_1 + 6y_2 - 3y_3 - y_4 \leq 5,$$

$$3y_1 + 5y_2 - 2y_3 - y_4 \leq -7,$$

$$3y_1 + 4y_2 - 3y_3 \leq 7,$$

$$2y_1 + y_2 - 4y_3 - y_4 \leq 5,$$

$$2y_1 + 4y_2 \leq 6, \text{ and } y_i \geq 0 \text{ for } i=1, 2, 3, \text{ and } 4.$$

■ **Application:** A Transportation Problem

A certain company has two factories, F 1 and F 2, each producing two products, Product 1 and Product 2, that are to be shipped to three distribution centers, Dist 1, Dist 2, and Dist 3.

The following table illustrates the cost associated with shipping each product from the factory to the distribution center, the minimum number of each product each distribution center needs, and the maximum output of each factory.

	F 1 /P 1	F 1/P 2	F 2/P 1	F 2/P 2	Minimum
Dist 1/Product 1	\$0.75		\$0.80		500
Dist 1/Product 2		\$0.50		\$0.40	400
Dist 2/Product 1	\$1.00		\$0.90		300
Dist 2/Product 2		\$0.75		\$1.20	500
Dist 3/Product 1	\$0.90		\$0.85		700
Dist 3/Product 2		\$0.80		\$0.95	300
Maximum Output	1000	400	800	900	

How much of each product should be shipped from each plant to each distribution center to minimize the total shipping costs?

- Let x_1 denote the number of units of Product 1 shipped from F 1 to Dist 1;
- x_2 denote the number of units of Product 2 shipped from F 1 to Dist 1;
- x_3 denote the number of units of Product 1 shipped from F 1 to Dist 2;
- x_4 denote the number of units of Product 2 shipped from F 1 to Dist 2;
- x_5 denote the number of units of Product 1 shipped from F 1 to Dist 3; and
- x_6 denote the number of units of Product 2 shipped from F 1 to Dist 3.
- Let x_7 denote the number of units of Product 1 shipped from F 2 to Dist 1;
- x_8 denote the number of units of Product 2 shipped from F 2 to Dist 1;
- x_9 denote the number of units of Product 1 shipped from F 2 to Dist 2;
- x_{10} denote the number of units of Product 2 shipped from F 2 to Dist 2;
- x_{11} denote the number of units of Product 1 shipped from F 2 to Dist 3; and
- x_{12} denote the number of units of Product 2 shipped from F 2 to Dist 3.

Then it is necessary to minimize the number:

$$Z = .75x_1 + .5x_2 + x_3 + .75x_4 + .9x_5 + .8x_6 + .8x_7 + .4x_8 + .9x_9 + 1.2x_{10} + .85x_{11} + .95x_{12}$$

subject to the constraints

- $x_1 + x_3 + x_5 \leq 1000$; $x_2 + x_4 + x_6 \leq 400$; $x_7 + x_9 + x_{11} \leq 800$;
- $x_8 + x_{10} + x_{12} \leq 900$; $x_1 + x_7 \geq 500$; $x_3 + x_9 \geq 300$; $x_5 + x_{11} \geq 700$;
- $x_2 + x_8 \geq 400$; $x_4 + x_{10} \geq 500$; $x_6 + x_{12} \geq 300$; and $x_i \geq 0$ for $i = 1, \dots, 12$.

In order to solve this linear programming problem, the objective function which computes the total cost, the 12 variables, and set of inequalities must be entered. The coefficients of the objective function are given in the vector **c**. Using the command **Array[x, 12]** illustrated in the previous example to define the list of 12 variables $\{x[1], x[2], \dots, x[12]\}$, the objective function is given by the product $z = \mathbf{xvec} \cdot \mathbf{c}$ where **xvec** is the name assigned to the list of variables.

```

LinearProgramming
In[26]:=
Clear[xvec, z, constraints, vars, c]
c = { .75, .5, 1, .75, .9, .8, .8, .4, .9, 1.2, .85, .95 }

Out[26]=
{0.75, 0.5, 1, 0.75, 0.9, 0.8, 0.8, 0.4,
 0.9, 1.2, 0.85, 0.95}

In[27]:=
xvec = Array[x, 12]

Out[27]=
{x[1], x[2], x[3], x[4], x[5], x[6], x[7],
 x[8], x[9], x[10], x[11], x[12]}

In[28]:=
z = xvec . c

Out[28]=
0.75 x[1] + 0.5 x[2] + x[3] + 0.75 x[4] +
0.9 x[5] + 0.8 x[6] + 0.8 x[7] +
0.4 x[8] + 0.9 x[9] + 1.2 x[10] +
0.85 x[11] + 0.95 x[12]

```

In order to define **z**, first define the table **c** so that the *i*th element of **c** is the coefficient of X_i .

Array[x, 12] creates the list of twelve elements $x[1], \dots, x[12]$; we will interpret $x[i]$ as X_i .

Notice that **xvec . c** produces the desired quantity, **z**, we wish to minimize. Hence, in defining **z** we have avoided considerable typing by using the command **Array**.

The set of constraints are then entered and named `constraints` for easier use. Therefore, the minimum cost and the value of each variable which yields this minimum cost are found with the command `ConstrainedMin[z, constraints, xvec]`.

```

In[29]:=
constraints={x[1]+x[3]+x[5] <=1000,
x[2]+x[4]+x[6] <=400, x[7]+x[9]+x[11] <=800,
x[8]+x[10]+x[12] <=900, x[1]+x[7] >=500,
x[3]+x[9] >=300, x[5]+x[11] >=700,
x[2]+x[8] >=400, x[4]+x[10] >=500,
x[6]+x[12] >=300};

```

Then type the constraints and name the resulting list `constraints`.

```

In[30]:=
ConstrainedMin[z, constraints, xvec]

```

Remember that a semi-colon placed at the end of a command suppresses the output.

```

Out[30]=
{2115, {x[1] -> 500, x[2] -> 0, x[3] -> 0,
x[4] -> 400, x[5] -> 200, x[6] -> 0,
x[7] -> 0, x[8] -> 400, x[9] -> 300,
x[10] -> 100, x[11] -> 500, x[12] -> 300}}

```

Finally, use Mathematica to find the values of `xvec` to minimize `z` subject to the constraints `constraints`.

The elements of the list which results from the command `ConstrainedMin[z, constraints, xvec]` can be extracted if this list is assigned a name. Therefore, the name `values` is given to this list. Notice that `values` is a list made up of two elements, the minimum value of the cost function, 2115, and the list of the variable values `{x[1]->500, x[2]->0, ...}`. Hence, the minimum cost is obtained with the command `values[[1]]` and the list of variable values which yield the minimum cost is extracted with `values[[2]]`.

LinearProgramming	
<code>In[11]:=</code>	<code>values=ConstrainedMin[z, constraints, xvec]</code>
<code>Out[11]=</code>	<code>{2115, {x[1] -> 500, x[2] -> 0, x[3] -> 0,</code> <code>x[4] -> 400, x[5] -> 200, x[6] -> 0,</code> <code>x[7] -> 0, x[8] -> 400, x[9] -> 300,</code> <code>x[10] -> 100, x[11] -> 500, x[12] -> 300}}</code>
<code>In[12]:=</code>	<code>values[[1]]</code>
<code>Out[12]=</code>	<code>2115</code>
<code>In[13]:=</code>	<code>values[[2]]</code>
<code>Out[13]=</code>	<code>{x[1] -> 500, x[2] -> 0, x[3] -> 0,</code> <code>x[4] -> 400, x[5] -> 200, x[6] -> 0,</code> <code>x[7] -> 0, x[8] -> 400, x[9] -> 300,</code> <code>x[10] -> 100, x[11] -> 500, x[12] -> 300}</code>

If we name the solutions, we can use the values later. Notice that `values` is a list with two elements.

`values[[1]]` is the first element of the list `values`.

`values[[2]]` is the second element of the list `values`; it is a list of twelve lists.

Using these extraction techniques, the number of units produced by each factory can be computed. Since

x_1 denotes the number of units of Product 1 shipped from F1 to Dist 1;
 x_3 denotes the number of units of Product 1 shipped from F1 to Dist 2;
 and x_5 denotes the number of units of Product 1 shipped from F1 to Dist 3,

then the total number of units of Product 1 produced by Factory 1 is given by $x[1]+x[3]+x[5]$. The command

```
x[1]+x[3]+x[5] /. values[[2]]
```

evaluates this sum at the values of $x[1]$, $x[3]$, and $x[5]$ given in the list `values[[2]]`. Similarly, the number of units of each product that each factory produces can be calculated. These results are shown below :

<pre>In[14]:= x[1]+x[3]+x[5] /. values[[2]] Out[14]= 700</pre>	<pre>replaces x[1] by 500, x[3] by by 0 and x[5] by 200.</pre>	<p>Hence, factory 1 produces 700 units of product 1.</p>
<pre>In[15]:= x[2]+x[4]+x[6] /. values[[2]] Out[15]= 400</pre>	<pre>replaces x[2] by 0, x[4] by 400, and x[6] by 0.</pre>	<p>Factory 1 produces 400 units of product 2.</p>
<pre>In[16]:= x[7]+x[9]+x[11] /. values[[2]] Out[16]= 800</pre>	<pre>replaces x[7] by 0, x[9] by 300, and x[11] by 500.</pre>	<p>Factory 2 produces 800 units of product 1.</p>
<pre>In[17]:= x[8]+x[10]+x[12] /. values[[2]] Out[17]= 800</pre>	<pre>replaces x[8] by 400, x[10] by 100, and x[12] by 300.</pre>	<p>Factory 2 produces 800 units of product 2.</p>
<pre>In[18]:= x[1]+x[7] /. values[[2]] Out[18]= 500</pre>	<pre>replaces x[1] by 500 and x[7] by 0.</pre>	

Also, the number of units of Products 1 and 2 received by each distribution center can be computed. The command `x[3]+x[9]/.values[[2]]` gives the total amount of Product 1 received at Dist 1 since `x[3]`= amount of Product 1 received by Dist 2 from F1 and `x[9]`= amount of Product 1 received by Dist 2 from F2 . Notice that this amount is the minimum number of units (300) of Product 1 requested by Dist 1. The amount of Products 1 and 2 received at each distribution center are calculated in a similar manner and illustrated below :

LinearProgramming

`In[19]:= x[3]+x[9]/.values[[2]]` replaces `x[3]` by 0 and `x[9]` by 300.

`Out[19]=` 300

`In[20]:= x[5]+x[11]/.values[[2]]` replaces `x[5]` by 200 and `x[11]` by 500.

`Out[20]=` 700

`In[21]:= x[2]+x[8]/.values[[2]]` replaces `x[2]` by 0 and `x[8]` by 400.

`Out[21]=` 400

`In[22]:= x[4]+x[10]/.values[[2]]` replaces `x[4]` by 400 and `x[10]` by 100.

`Out[22]=` 500

`In[23]:= x[6]+x[12]/.values[[2]]` replaces `x[6]` by 0 and `x[12]` by 300.

`Out[23]=` 300

Each distribution center receives exactly the minimum number of each product it requests.

■ 5.3 Vector Calculus

■ Review of Definitions and Notation

The terminology and notation used in *Mathematica by Example* is standard. Nevertheless, we review basic definitions briefly.

A **scalar field** is a function with domain a set of ordered-triples and range a subset of the real numbers:

$f:U \rightarrow V$ is a scalar field means $U \subseteq \mathfrak{R}^3$ and $V \subseteq \mathfrak{R}$.

The **gradient of the scalar field f** is defined to be the vector

$$\text{grad } f = \frac{\partial f}{\partial x} \mathbf{i} + \frac{\partial f}{\partial y} \mathbf{j} + \frac{\partial f}{\partial z} \mathbf{k} = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle = \langle f_x, f_y, f_z \rangle, \text{ where } \mathbf{i} = \langle 1, 0, 0 \rangle, \mathbf{j} = \langle 0, 1, 0 \rangle, \text{ and } \mathbf{k} = \langle 0, 0, 1 \rangle.$$

A **vector field f** is a vector-valued function:

$f:V \rightarrow U$, $U \subseteq \mathfrak{R}^3$ and $V \subseteq \mathfrak{R}$, is a vector field means that f can be written in the form $f(x, y, z) = f_1(x, y, z)\mathbf{i} + f_2(x, y, z)\mathbf{j} + f_3(x, y, z)\mathbf{k} = \langle f_1(x, y, z), f_2(x, y, z), f_3(x, y, z) \rangle$ for each (x, y, z) in the domain of f .

A **conservative vector field f** is a vector field that is the gradient of a scalar field:

f is a conservative vector field means there is a scalar field g satisfying

$f = \nabla^2 g$. In this case, g is usually called a **potential function** for f .

The **divergence of the vector field f** is defined to be the scalar

$$\begin{aligned} \text{div } f &= \text{div } f(x, y, z) = \text{div} \langle f_1(x, y, z), f_2(x, y, z), f_3(x, y, z) \rangle \\ &= \frac{\partial f_1(x, y, z)}{\partial x} + \frac{\partial f_2(x, y, z)}{\partial y} + \frac{\partial f_3(x, y, z)}{\partial z} \\ &= \nabla \cdot f. \end{aligned}$$

The **laplacian of the scalar field f** is defined to be $\text{div}(\text{grad } f)$

$$\text{laplacian}(f) = \nabla^2 f = \Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = f_{xx} + f_{yy} + f_{zz}.$$

For three-dimensional vector analysis, the package **VectorAnalysis.m** contains the commands **Grad**, **Div**, **Curl**, and **Laplacian**.

Be sure to load the package **VectorAnalysis.m** prior to using these functions.

VectorAnalysis.m is contained in the folder **Calculus**.

Because *Mathematica* recognizes Cartesian (x, y, z) , Cylindrical (r, ϕ, z) , and Spherical (r, θ, ϕ) coordinates, and because the operations discussed in this section differ in the various coordinate systems, the desired coordinate system must be indicated. This is accomplished with

SetCoordinates[System] where

System is usually one of **Cartesian**, **Cylindrical**, or **Spherical**.

However, the available coordinate systems are:

Cartesian, Cylindrical, Spherical, Parabolic, ParabolicCylinder, ProlateEllipsoidal, EllipticCylinder, OblateEllipsoidal, Toroidal, Elliptic, and Bipolar.

The examples illustrated below are done in cartesian coordinates.

After the function $f[x, y, z]$ has been defined, the gradient is found with :

Grad[f[x, y, z]].

Since this is a function of (x,y,z) , it is denoted **gradientf[x, y, z]** for later use.

The laplacian of f is determined using

Laplacian[f[x, y, z]],

and the divergence is found with

Div[gradient[f[x, y, z]]].

□ Example:

Let $f(x,y,z) = \text{Cos}(xyz)$. Compute ∇f , $\nabla^2 f$, and $\text{Div}(\nabla f)$.

Note: When defining f , be sure to include the space between the variables.

Notice that `Laplacian[f[x,y,z]]` yields the same result as `Div[gradient[x,y,z]]`.

The screenshot shows a Mathematica notebook window titled "ThreeDVectors". The code and output are as follows:

```

In[41]:=
<<VectorAnalysis.m
SetCoordinates[Cartesian];

In[42]:=
Clear[f]
f[x_,y_,z_]=Cos[x y z];

In[43]:=
gradientf[x_,y_,z_]=Grad[f[x,y,z]]

Out[43]=
(-(y z Sin[x y z]), -(x z Sin[x y z]),
 -(x y Sin[x y z]))

In[44]:=
Laplacian[f[x,y,z]]

Out[44]=
  2 2      2 2
-(x y Cos[x y z]) - x z Cos[x y z] -
  2 2
y z Cos[x y z]

In[45]:=
Div[gradientf[x,y,z]]

Out[45]=
  2 2      2 2
-(x y Cos[x y z]) - x z Cos[x y z] -
  2 2
y z Cos[x y z]

```

Annotations in the image explain the functions used:

- The built-in functions Grad, Div, Curl, and Laplacian are contained in the package VectorAnalysis.m be sure to load it prior to using the functions for three-dimensional vector analysis.**
- SetCoordinates[Cartesian] specifies that our calculations will be using Cartesian coordinates.**
- computes grad $f(x,y,z) = \nabla f(x,y,z)$.**
- computes Laplacian $f(x,y,z) = \nabla^2 f(x,y,z) = \text{Div}(\text{Grad } f)$.**
- computes the divergence of $\text{gradientf}[x,y,z]$: $\text{Div}(\text{Grad } f) = \nabla^2 f$.**

If S is the graph of $f(x,y)$ and $g(x,y,z)=z - f(x,y)$, then the gradient

$\nabla g(x,y,z)$ is a normal vector to the graph of $g(x,y,z) = 0$.

At the point (x,y,z) a **unit normal vector** n can be obtained via:

$$n = \frac{\nabla g(x,y,z)}{\|\nabla g(x,y,z)\|} = \frac{-f_x(x,y)\mathbf{i} - f_y(x,y)\mathbf{j} + \mathbf{k}}{\sqrt{[f_x(x,y)]^2 + [f_y(x,y)]^2 + 1}} = \frac{1}{\sqrt{[f_x(x,y)]^2 + [f_y(x,y)]^2 + 1}} \langle -f_x(x,y), -f_y(x,y), 1 \rangle.$$

The **curl of the vector field f** is defined to be the vector field

$$\begin{aligned} \text{curl } f &= \text{curl } f(x,y,z) = \text{curl}(f_1(x,y,z), f_2(x,y,z), f_3(x,y,z)) \\ &= \left(\frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z} \right) \mathbf{i} + \left(\frac{\partial f_1}{\partial z} - \frac{\partial f_3}{\partial x} \right) \mathbf{j} + \left(\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y} \right) \mathbf{k} \\ &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ f_1 & f_2 & f_3 \end{vmatrix}. \end{aligned}$$

□ **Example:**

Let $f(x,y,z) = xy\mathbf{i} + xz^2y\mathbf{j} - e^{2z}\mathbf{k} = \{xy, xz^2y, -e^{2z}\}$. Compute

$$\text{Curl } f = \left\{ \left(\frac{\partial}{\partial y}(-e^{2z}) - \frac{\partial}{\partial z}(xz^2y) \right), \left(\frac{\partial}{\partial z}(xy) - \frac{\partial}{\partial x}(-e^{2z}) \right), \left(\frac{\partial}{\partial x}(xz^2y) - \frac{\partial}{\partial y}(xy) \right) \right\};$$

$$\text{Div } f = \frac{\partial}{\partial x}(xy) + \frac{\partial}{\partial y}(xz^2y) + \frac{\partial}{\partial z}(-e^{2z}); \quad \text{Laplacian}(\text{Div } f) = \nabla^2(\text{Div } f);$$

$$\text{Grad}(\text{Laplacian}(\text{Div } f)) = \text{Grad}(\nabla^2(\text{Div } f)); \quad \text{and}$$

$$\text{Laplacian}(\text{Grad}(\text{Laplacian}(\text{Div } f))) = \nabla^2(\text{Grad}(\text{Laplacian}(\text{Div } f))).$$

The first step towards solving this problem is to enter the unit vectors in Cartesian coordinates $\mathbf{i} = \{1, 0, 0\}$, $\mathbf{j} = \{0, 1, 0\}$, and $\mathbf{k} = \{0, 0, 1\}$. The vector-valued function $\mathbf{f}[\mathbf{x}, \mathbf{y}, \mathbf{z}]$ can then be defined using these three unit vectors as follows:

$\mathbf{f}[\mathbf{x}, \mathbf{y}, \mathbf{z}] = \mathbf{x} \mathbf{y} \mathbf{i} + \mathbf{x} \mathbf{z}^2 \mathbf{y} \mathbf{j} - \text{Exp}[2\mathbf{z}] \mathbf{k}$ (remembering to place appropriate spaces between variables for multiplication).

Notice that the coordinate system has not been set in this problem. However, the correct system can be indicated in each command. For example, the curl of f in Cartesian coordinates is determined with `Curl[f[x,y,z],Cartesian]`. The curl could similarly be obtained in the other systems by replacing `Cartesian` with `Cylindrical`, `Spherical`, or one of the other available coordinate systems, in the command above.

```

In[26]:=
<<VectorAnalysis.m

In[27]:=
Clear[i,j,k]
i={1,0,0}
j={0,1,0}
k={0,0,1}

Out[27]=
{0, 0, 1}

In[28]:=
Clear[f]
f[x_,y_,z_]=x y i+x z^2 y j-Exp[2z]k

Out[28]=
      2      2 z
{x y, x y z, -E }

In[29]:=
curlf[x_,y_,z_]=
  Curl[f[x,y,z],Cartesian]

Out[29]=
      2
{-2 x y z, 0, -x + y z }

```

Be sure the package `VectorAnalysis.m` has been loaded prior to using the built-in command `Curl`, `Grad`, `Div`, and `Laplacian`.

After defining $i=\{1,0,0\}$, $j=\{0,1,0\}$, and $k=\{0,0,1\}$, defining f by the command $f[x_,y_,z_]=x y i+x z^2 y j-\text{Exp}[2z]k$ produces the same result as defining f by the command $f[x_,y_,z_]=\{x y, x z^2 y, -\text{Exp}[2z]\}$

Don't forget to include the space between the variables to denote multiplication.

`Curl[f[x,y,z],Cartesian]` computes the curl of the (vector-valued) function f .

As was the case with computing the curl of f , the divergence of f can be calculated in Cartesian coordinates with `Div[f[x,y,z],Cartesian]`. Again, since the divergence is a function of (x,y,z) , it is named `divf[x,y,z]` for later use. Hence, the Laplacian of the divergence of f is computed with `Laplacian[divf[x,y,z],Cartesian]`. This function is called `ladiv[x,y,z]` so that

$$\text{Grad}(\text{Laplacian}(\text{Div } f)) = \text{Grad}(\nabla^2(\text{Div } f))$$

can be found with `Grad[ladivf[x,y,z],Cartesian]`. The resulting function is then named

$$\text{grad}[x,y,z] \text{ so that } \text{Laplacian}(\text{Grad}(\text{Laplacian}(\text{Div } f))) = \nabla^2(\text{Grad}(\text{Laplacian}(\text{Div } f)))$$

can be computed with `Laplacian[grad[x,y,z],Cartesian]`.

The screenshot shows a Mathematica notebook window titled "ThreeDVectors" with the following content:

```

In[31]:=
  divf[x_,y_,z_]=
    Div[f[x,y,z],Cartesian]

Out[31]=
  2 z      2
-2 E      + y + x z

In[33]:=
  ladivf[x_,y_,z_]=
    Laplacian[divf[x,y,z],Cartesian]

Out[33]=
  2 z
-8 E      + 2 x

In[35]:=
  grad[x_,y_,z_]=
    Grad[ladivf[x,y,z],Cartesian]

Out[35]=
      2 z
{2, 0, -16 E }

In[37]:=
  Laplacian[grad[x,y,z],Cartesian]

Out[37]=
      2 z
{0, 0, -64 E }

```

A callout box points to the `Cartesian` argument in the `Div` command, stating: "Cartesian indicates that we are working in cylindrical coordinates."

Annotations on the right side of the notebook describe the operations:

- `Div[f[x,y,z],Cartesian]` computes the divergence of the (vector-valued) function f . The resulting real-valued function of $x, y,$ and z is defined to be `divf[x,y,z]`.
- `Laplacian[divf[x,y,z],Cartesian]` computes the Laplacian of the real-valued function `divf`. The resulting function is defined to be `ladivf[x,y,z]`.
- `Grad[ladivf[x,y,z],Cartesian]` computes the gradient of the real-valued function `ladivf`. The resulting (vector-valued) function is defined to be `grad[x,y,z]`.
- `Laplacian[grad[x,y,z],Cartesian]` computes the Laplacian of the (vector-valued) function `grad`.

□ Example:

Let $w(x,y) = \cos(4x^2 + 9y^2)$. Let $n_{(x,y)}$ denote a unit vector normal to the graph of w at the point $(x,y,w(x,y))$. Find a formula for n .

In order to visualize the unit normal vector at points $(x,y,w(x,y))$ to the surface $w(x,y)$, this function is plotted using several of the options available with `Plot3D`. These options are discussed below :

☐
ThreeDVectors
☐

```

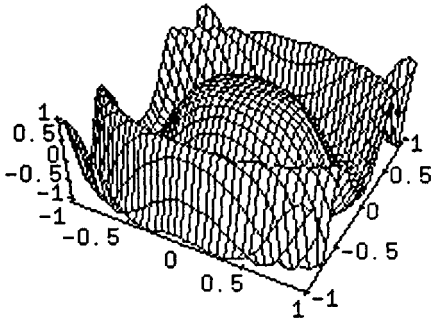
In[71]:=
Clear[w]
w[x_,y_]=Cos[4x^2+9y^2];

In[75]:=
Plot3D[w[x,y],{x,-1,1},{y,-1,1},
Boxed->False, Axes->Automatic,
PlotPoints->35, Shading->False]

```

After clearing all prior definitions of w , define $w(x,y) = \cos(4x^2 + 9y^2)$.

Use the `Plot3D` command to graph w .



Remember that the form of the `Plot3D` command is `Plot3D[f[x,y],{x,xmin,xmax},{y,ymin,ymax},options]`.

The option `Boxed->False` prevents a box from being drawn around the graph; the option `Axes->Automatic` specifies that Mathematica is to choose "reasonable" axes; the option `PlotPoints->35` specifies that 35 values are to be sampled from the x-axis and 35 values are to be sampled from the y-axis; and the option `Shading->False` specifies that the resulting graph is not to be shaded.

```

Out[75]=
-SurfaceGraphics-

```

The equation $z = w(x,y)$ is written as $z - w(x,y) = 0$. The left-hand side of this equation is a function of x , y , and z and is defined as `wz[x_,y_,z_]=z-w[x,y]`.

Since the partial derivative of wz with respect to z is -1 , the gradient of wz is a function of x and y only. Hence, the gradient of wz is named $gw[x_, y_]$ and is computed with `Grad[wz[x, y, z], Cartesian]`. The length of the gradient of wz which is necessary in determining the unit normal vector is the square root of the dot product of the gradient of wz with itself. This product is computed with `gw[x, y].gw[x, y]`.

ThreeDVectors

In[75]:= `wz[x_, y_, z_] = z - w[x, y]`

Out[75]=
$$z - \cos[4x^2 + 9y^2]$$

In[78]:= `gw[x_, y_] = Grad[wz[x, y, z], Cartesian]`

Out[78]=
$$\{8x \sin[4x^2 + 9y^2], 18y \sin[4x^2 + 9y^2], 1\}$$

In[79]:= `gw[x, y].gw[x, y]`

Out[79]=
$$1 + 64x^2 \sin^2[4x^2 + 9y^2] + 324y^2 \sin^2[4x^2 + 9y^2]$$

Define $wz[x, y, z]$ to be the real-valued function of $x, y,$ and $z, wz[x, y, z] = z - w[x, y]$.

computes the gradient of the function $wz[x, y, z]$: $-w_x(x, y)i - w_y(x, y)j + k = \{-w_x(x, y), -w_y(x, y), 1\}$ Notice that the resulting vector-valued function is a function of x and $y,$ but NOT $z.$ Hence, we name the result $gw[x, y]$.

computes the dot product $gw[x, y].gw[x, y].$ $(w_x(x, y))^2 + (w_y(x, y))^2 + 1.$

Therefore, the unit normal vector is the gradient of g , $\mathbf{gw}[x, y]$, divided by the the square root of $\mathbf{gw}[x, y] \cdot \mathbf{gw}[x, y]$ as shown below. This is also a function of the variables x and y since the unit normal vector differs from point to point on the surface. Hence, this vector is assigned the name `normalw[x_, y_]` so that the unit vector at any point $(x,y,w(x,y))$ can be easily determined by evaluating `normalw[x_, y_]` at any point (x,y) .

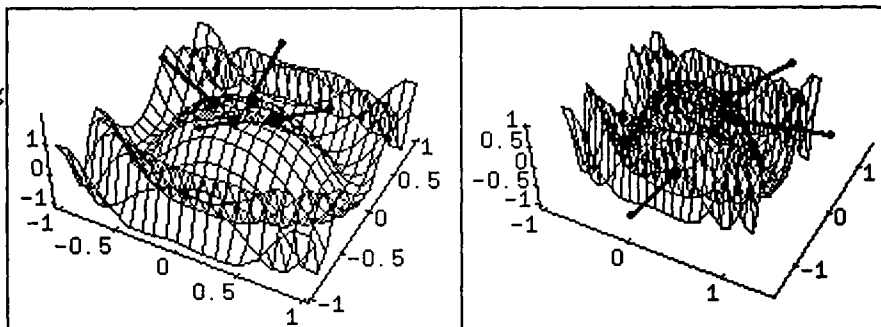
```

ThreeDVectors
In[50]:=
normalw[x_, y_]=gw[x, y]/
  Sqrt[gw[x, y].gw[x, y]]
Out[50]=
      2      2
  ((8 x Sin[4 x + 9 y ] ) /
    Sqrt[1 + 64 x Sin[4 x + 9 y ] +
      324 y Sin[4 x + 9 y ] ] .
    (18 y Sin[4 x + 9 y ] ) /
    Sqrt[1 + 64 x Sin[4 x + 9 y ] +
      324 y Sin[4 x + 9 y ] ] .
    1 / Sqrt[1 + 64 x Sin[4 x + 9 y ] +
      324 y Sin[4 x + 9 y ] ] )
  
```

yields a unit normal vector for w :

$$n = \frac{-w_x(x,y)i - w_y(x,y)j + k}{\sqrt{(w_x(x,y))^2 + (w_y(x,y))^2 + 1}}$$

We can use *Mathematica* to visualize $w(x,y)$ along with several normal unit vectors.



■ **Application: Green's Theorem**

Green's Theorem: Let C be a piecewise smooth simple closed curve and let R be the region consisting of C and its interior. If f and g are functions that are continuous and have continuous first partial derivatives throughout an

open region D containing R, then $\oint_C (m(x,y)dx + n(x,y)dy) = \iint_R \left(\frac{\partial n}{\partial x} - \frac{\partial m}{\partial y} \right) dA$.

□ **Example:**

Use Green's Theorem to evaluate $\int_C (x + e^{\sqrt{y}})dx + (2y + \cos(x))dy$ where

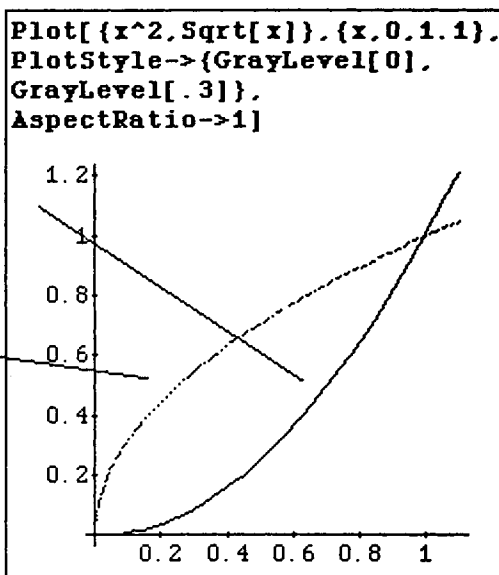
C is the boundary of the region enclosed by the parabolas $y = x^2$ and $x = y^2$.

To calculate the limits of integration, we use *Mathematica* to graph the functions x^2 and \sqrt{x} .

This is the graph of the function x^2 .

This is the graph of the function \sqrt{x} .

Notice that the two functions intersect at the points (0,0) and (1,1).



In this example,

$m(x,y) = x + e^{\sqrt{y}}$ and $n(x,y) = 2y + \cos(x)$. Therefore, applying Green's Theorem,

$$\begin{aligned} \int_C (x + e^{\sqrt{y}})dx + (2y + \cos(x))dy &= \int_C m(x,y)dx + n(x,y)dy \\ &= \iint_R \left(\frac{\partial n}{\partial x} - \frac{\partial m}{\partial y} \right) dA = \int_0^1 \int_{x^2}^{\sqrt{x}} \left(\frac{\partial n}{\partial x} - \frac{\partial m}{\partial y} \right) dy dx. \end{aligned}$$

Therefore we will use *Mathematica* to define $m(x,y)$, $n(x,y)$, and to

compute $\frac{\partial n}{\partial x}$, $\frac{\partial m}{\partial y}$, and $\int_0^1 \int_{x^2}^{\sqrt{x}} \left(\frac{\partial n}{\partial x} - \frac{\partial m}{\partial y} \right) dy dx$.

First, the functions $m(x,y)$ and $n(x,y)$ are defined. Recall that in computing partial derivatives, the variable of differentiation must be given. Therefore, the partial of $n[x,y]$ with respect to x is given by $D[n[x,y],x]$. This derivative is then named nx for later use. Similarly, the partial derivative of $m[x,y]$ with respect to y is found with $D[m[x,y],y]$ and named my .

GreensTheorem	
<code>In[9]:=</code>	<code><<IntegralTables.m</code>
<code>Out[9]=</code>	Integrator`
<code>In[9]:=</code>	<code>Clear[m,n]</code>
	<code>m[x_,y_]=x+Exp[Sqrt[y]]</code>
	<code>n[x_,y_]=2y+Cos[x];</code>
<code>In[10]:=</code>	<code>nx=D[n[x,y],x]</code>
<code>Out[10]=</code>	-Sin[x]
<code>In[11]:=</code>	<code>my=D[m[x,y],y]</code>
<code>Out[11]=</code>	Sqrt[y] E ----- 2 Sqrt[y]

Be sure to load the package IntegralTables.m before attempting to compute any definite integrals with Mathematica.

After clearing all prior definitions of m and n, define m and n.

`D[n[x,y],x]`
computes $\frac{\partial n}{\partial x}$.

`D[m[x,y],y]`
computes $\frac{\partial m}{\partial y}$.

Mathematica computes the exact value of the double integral. To obtain a more meaningful value, we approximate the value of the double integral using the **N** command. In general, the command **N[*]** produces a numerical approximation of the previous output.

<pre>In[14]:= Integrate[nx-my, {x, 0, 1}, {y, x^2, Sqrt[x]}] Out[14]= -27 + 9 E + 2 Cos[1] - Sqrt[Pi] Erf[Sqrt[-I]] ----- 4 Sqrt[-I] Sqrt[Pi] Erf[Sqrt[I]] ----- + 2 Sin[1] 4 Sqrt[I]</pre>	<p><i>computes the double integral</i></p> $\int_0^1 \int_{x^2}^{\sqrt{x}} \left(-\sin(x) - \frac{e\sqrt{y}}{2\sqrt{y}} \right) dy dx.$
<pre>In[15]:= N[*] Out[15]= -0.676441</pre>	<p>N[*] <i>produces a numerical approximation of the previous output; hence, -0.676441 is a numerical approximation of the double integral.</i></p>

■ **Application:** The Divergence Theorem

The Divergence Theorem: Let Q be any domain with the property that each straight line through any interior point of the domain cuts the boundary in exactly two points, and such that the boundary S is a piecewise-smooth, closed, oriented surface with unit outer normal n. If f is a vector field that has continuous partial derivatives on Q, then

$$\iint_S \mathbf{f} \cdot \mathbf{n} dS = \iiint_Q \text{div } \mathbf{f} \, dV = \iiint_Q \nabla \cdot \mathbf{f} \, dV.$$

$\iint_S \mathbf{f} \cdot \mathbf{n} dS$ is called the **outward flux** of the vector field f across the surface S.

If S is a portion of the level curve $g(x,y,z)=c$ for some g, then a unit normal vector n may be taken to be either

$$\mathbf{n} = \frac{\nabla g}{\|\nabla g\|} \quad \text{or} \quad \mathbf{n} = -\frac{\nabla g}{\|\nabla g\|}.$$

■ Recall the following formulas for the evaluation of surface integrals:

Let S be the graph of $z=f(x,y)$ ($y=h(x,z)$ or $x=k(y,z)$) and let

R_{xy} (R_{xz} or R_{yz}) be the projection of S on the xy (xz or yz)–plane. Then,

$$\iint_S g(x, y, z) dS = \begin{cases} \iint_{R_{xy}} g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1} dA \\ \iint_{R_{xz}} g(x, h(x, z), z) \sqrt{(h_x(x, z))^2 + (h_z(x, z))^2 + 1} dA \\ \iint_{R_{yz}} g(k(y, z), y, z) \sqrt{(k_y(y, z))^2 + (k_z(y, z))^2 + 1} dA \end{cases}$$

□ Example:

Use the Divergence Theorem to compute the outward flux of the field

$$\mathbf{vf}(x, y, z) = \{xy + x^2yz, yz + xy^2z, xz + xyz^2\} = (xy + x^2yz)\mathbf{i} + (yz + xy^2z)\mathbf{j} + (xz + xyz^2)\mathbf{k}$$

through the surface of the cube cut from the first octant by the planes $x=2$, $y=2$, and $z=2$.

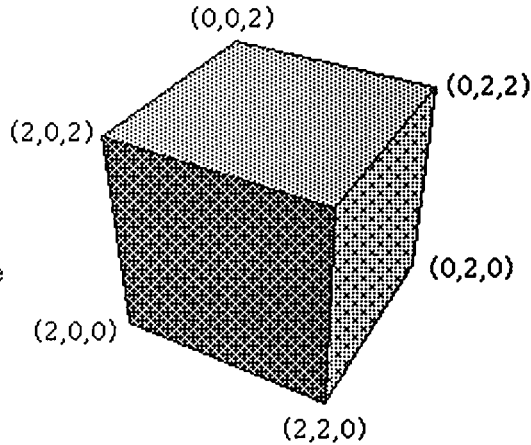
By the Divergence Theorem,

$$\iint_{\text{Cube Surface}} \mathbf{vf} \cdot \mathbf{n} dA = \iiint_{\text{Cube Interior}} \nabla \cdot \mathbf{vf} dV.$$

Notice that without the Divergence Theorem,

calculating $\iint_{\text{Cube Surface}} \mathbf{vf} \cdot \mathbf{n} dA$ would require six

separate integrals. However, with the Divergence Theorem, calculating the flux can be accomplished by integrating the divergence.



After loading the `IntegralTables.m` and `VectorAnalysis.m` packages, the vector field \mathbf{vf} is defined as a list of three elements, the x, y, and z components. Since the volume integral is that of a cube, the logical choice for a coordinate system is cartesian coordinates. Hence the divergence of \mathbf{vf} is calculated with `Div[vf[x,y,z],Cartesian]`. Since the divergence is a function of (x,y,z), it is defined as the function `divvf` for later use in the volume integral. Therefore, the outward flux of the field \mathbf{vf} through the surface of the cube is found to be 72 with

```
Integrate[divvf[x,y,z],{x,0,2},{y,0,2},{z,0,2}].
```

DivergenceTheorem

*Be sure to load the packages
IntegralTables.m and VectorAnalysis.m
before attempting to compute definite integrals
of divergence of three-dimensional vector fields.*

```
In[1]:=
<<IntegralTables.m
<<VectorAnalysis.m
```

*First define the function
vf to be the three-dimensional
vector field.*

```
In[2]:=
vf[x_,y_,z_]={x y +x^2 z y.y z+ x y^2 z,x z+x y z^2 }
```

```
Out[2]=
      2           2           2
{x y + x y z, y z + x y z, x z + x y z }
```

*Use the command
to compute the divergence
of vf ; name the resulting
function (of x, y, and z) divvf.*

```
In[3]:=
divvf[x_,y_,z_]=Div[vf[x,y,z],Cartesian]
```

```
Out[3]=
x + y + z + 6 x y z
```

*computes the triple
integral*

```
In[4]:=
Integrate[divvf[x,y,z],{x,0,2},{y,0,2},{z,0,2}]
```

```
Out[4]=
72
```

$$\int_0^2 \int_0^2 \int_0^2 \text{divvf}[x,y,z] dz dy dx.$$

■ **Application:** Stoke's Theorem

Stokes's Theorem: Let S be an oriented surface with finite surface area, unit normal \mathbf{n} , and boundary C. Let F be a continuous vector-field defined on S such that the component functions of F have continuous partial derivatives

at each non-boundary points of S. Then,
$$\int_C \mathbf{F} \cdot d\mathbf{r} = \iint_S (\text{Curl } \mathbf{F}) \cdot \mathbf{n} dS.$$

In other words, the surface integral of the normal component of the curl of F taken over S equals the line integral of

the tangential component of the field taken over C:
$$\oint_C \mathbf{F} \cdot T ds = \iint_S \text{curl } \mathbf{F} \cdot \mathbf{n} dS.$$

In particular, if $\mathbf{F} = M\mathbf{i} + N\mathbf{j} + P\mathbf{k} = \{M, N, P\}$, then

$$\int_C (M(x,y,z)dx + N(x,y,z)dy + P(x,y,z)dz) = \iint_S (\text{Curl } \mathbf{F}) \cdot \mathbf{n} dS.$$

□ Example:

Verify Stoke's Theorem for the vector field

$$\mathbf{vf}(x,y,z) = (y^2 - z)\mathbf{i} + (z^2 + x)\mathbf{j} + (x^2 - y)\mathbf{k} = \{y^2 - z, z^2 + x, x^2 - y\} \text{ and } S \text{ the paraboloid}$$

$$z = f(x,y) = 4 - (x^2 + y^2), \quad z \geq 0.$$

Since we must show

$$\int_C \mathbf{vf} \cdot d\mathbf{r} = \iiint_S (\text{Curl } \mathbf{vf}) \cdot \mathbf{n} \, dS, \text{ we must compute } \text{Curl } \mathbf{vf}, \mathbf{n}, \iiint_S (\text{Curl } \mathbf{vf}) \cdot \mathbf{n} \, dS, \mathbf{r}, d\mathbf{r}, \text{ and } \int_C \mathbf{vf} \cdot d\mathbf{r}.$$

First define the vector-field and the surface as \mathbf{vf} and \mathbf{f} , respectively. Compute the curl of \mathbf{vf} and name it `curlvf[x,y,z]` then compute a normal vector and name it `normal[x,y,z]`:

StokesTheorem

```
In[1]:=
<<IntegralTables.m
<<VectorAnalysis.m
In[2]:=
Clear[vf,f,h,normal,un,g,curlvf,n]
vf[x_,y_,z_]={y^2-z,z^2+x,x^2-y}
f[x_,y_]=4-(x^2+y^2);
In[3]:=
curlvf[x_,y_,z_]=Curl[vf[x,y,z],Cartesian]
Out[3]=
{-1 - 2 z, -1 - 2 x, 1 - 2 y}
In[4]:=
h[x_,y_,z_]=z-f[x,y]
Out[4]=
      2      2
-4 + x  + y  + z
In[5]:=
normal[x_,y_,z_]=Grad[h[x,y,z],Cartesian]
Out[5]=
{2 x, 2 y, 1}
```

Be sure the packages `IntegralTables.m` and `VectorAnalysis.m` have been loaded prior to using the commands `Integrate`, `Grad`, and `Curl`.

Begin by clearing all prior definitions of objects to be named in this example. Then define \mathbf{vf} and \mathbf{f} .

Don't forget to specify that the curl and gradient are to be computed using Cartesian coordinates.

`curlvf[x,y,z]` is the curl of the vector function.

Define $h[x,y,z]$: we will use h to compute a unit normal vector.

Define `normal[x,y,z]` to be the gradient of h .

A unit normal vector \mathbf{n} is given by $\mathbf{n} = \frac{\nabla h}{\|\nabla h\|} = \frac{\text{normal}[x,y,z]}{\|\text{normal}[x,y,z]\|}$.

Since `normal[x,y,z]` is a list (of three elements), `normal[x,y,z][[i]]` yields the i th element of the list `normal[x,y,z]`. Therefore,

`||normal[x,y,z]||` is given by the command

`Sqrt[Sum[(normal[x,y,z][[i]])^2, {i,1,3}]]`. An alternative approach is to recall that

for a vector \mathbf{v} , $\|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$. Consequently, the command `Sqrt[normal[x,y,z].normal[x,y,z]]`

yields the same result.

In order to easily use the surface integral evaluation formula, define $g[x, y, z]$ to be the dot product of $\text{curl}vf[x, y, z]$ and $\text{un}[x, y, z]$.

StokesTheorem

```

In[6]:=
un[x_, y_, z_] = normal[x, y, z] /
  Sqrt[Sum[(normal[x, y, z][[i]])^2, {i, 1, 3}]]

```

Out[6]=

$$\left\{ \frac{2x}{\sqrt{1+4x^2+4y^2}}, \frac{2y}{\sqrt{1+4x^2+4y^2}}, \frac{1}{\sqrt{1+4x^2+4y^2}} \right\}$$

Sqrt[Sum[(normal[x, y, z][[i]])^2, {i, 1, 3}]] computes $\sqrt{\sum_{i=1}^3 (\text{normal}[x, y, z][[i]])^2} = \sqrt{4x^2 + 4y^2 + 1}$. Therefore, $\text{un}[x, y, z]$ is a unit vector normal to the graph of $g(x, y, z) = 0$ at the point (x, y, z) and hence a unit vector normal to the graph of $f(x, y)$.

```

In[7]:=
g[x_, y_, z_] = Together[curlvf[x, y, z] . un[x, y, z]]

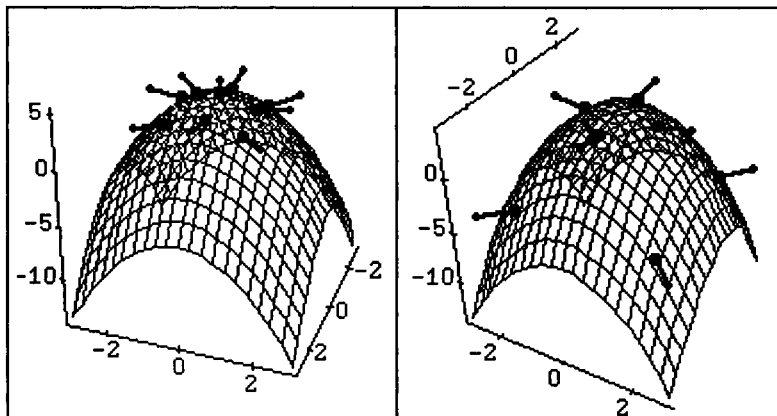
```

Out[7]=

$$\frac{1 - 2x - 4y - 4xy - 4xz}{\sqrt{1 + 4x^2 + 4y^2}}$$

*$g[x, y, z]$ is the dot product of $\text{curl}vf$ and un . The command **Together** expresses the result as a single fraction.*

Here we visualize $f(x, y)$ with several normal unit vectors.



By the surface integral evaluation formula,

$$\iint_S \overbrace{(\text{Curl } \mathbf{v}) \cdot \mathbf{n}}^{\text{curl } \mathbf{v}} dS = \iint_S g(x, y, z) dS = \iint_R g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1} dA, \text{ where}$$

R is the projection of $f(x, y)$ on the xy -plane. Hence, in this example, R is the region bounded by the graph of the circle $x^2 + y^2 = 4$. Thus,

$$\iint_R g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1} dA = \int_{-2}^2 \int_{-\sqrt{4-x^2}}^{\sqrt{4-x^2}} g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1} dy dx.$$

Notice that the integral $\int_{-2}^2 \int_{-\sqrt{4-x^2}}^{\sqrt{4-x^2}} g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1} dy dx$ can be

easily evaluated using polar coordinates. To do so, in

$g(x, y, f(x, y)) \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2 + 1}$, replace each x by $r \cos(t)$ and each y by $r \sin(t)$.

StokesTheorem

```

In[8]:=
function=g[x,y,f[x,y]] Sqrt[(D[f[x,y],x])^2
+(D[f[x,y],y])^2+1]

Out[8]=

1 - 2 x - 4 y - 4 x y - 4 x (4 - x - y )

In[9]:=
Integrate[function, {x, -2, 2}, {y, -Sqrt[4-x^2],
Sqrt[4-x^2]}]

Out[9]=
4 Pi

In[10]:=
function=g[x,y,f[x,y]] Sqrt[(D[f[x,y],x])^2
+(D[f[x,y],y])^2+1] /. x->r Cos[t] /.
y->r Sin[t]

Out[10]=
1 - 2 r Cos[t] - 4 r Sin[t] -
2
4 r Cos[t] Sin[t] -
2 2 2 2
4 r Cos[t] (4 - r Cos[t] - r Sin[t] )

```

To evaluate the surface integral, we use the surface integral evaluation formulas.

Hence, the value of the surface integral is 4π .

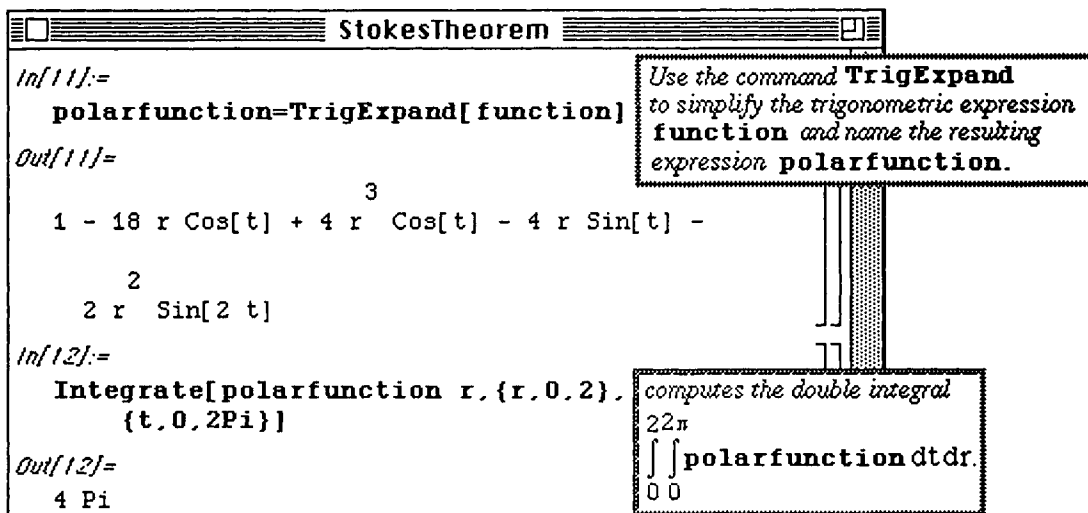
Notice that when we redefine function the prior definition is erased.

To use polar coordinates, replace each x in function by $r \cos(t)$ and each y in function by $r \sin(t)$.

The command `TrigExpand[expression]` applies basic trigonometric identities to attempt to simplify **expression**. Finally, to evaluate the integral in polar coordinates, the limits of integration must be changed and $dx dy$ must be replaced by $r dt dr$. Hence, the same value is obtained by the integral:

$$\int_0^{2\pi} \int_0^2 \underbrace{(1 - 18r \cos(t) + 4r^3 \cos(t) - 4r \sin(t) - 2r^2 \sin(2t))}_{\text{simplified version of function}} r dt dr.$$

- Version 2.0 does not include the command `TrigExpand`. The same results as `TrigExpand[expression]` are obtained with `Expand[expression, Trig->True]`.



```

In[11]:=
  polarfunction=TrigExpand[function]
Out[11]=
      3
1 - 18 r Cos[t] + 4 r Cos[t] - 4 r Sin[t] -
      2
      2 r Sin[2 t]
In[12]:=
  Integrate[polarfunction r, {r, 0, 2}, {t, 0, 2Pi}]
Out[12]=
  4 Pi

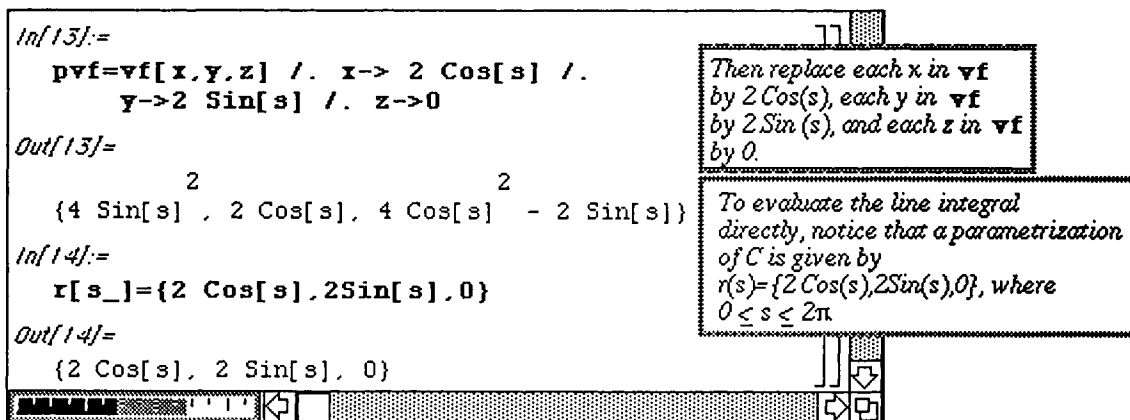
```

Use the command `TrigExpand` to simplify the trigonometric expression **function** and name the resulting expression **polarfunction**.

computes the double integral $\int_0^{2\pi} \int_0^2 \text{polarfunction } dt dr$.

We are able to evaluate the line integral directly by noticing that the boundary of

$z = f(x, y) = 4 - (x^2 + y^2)$, $z \geq 0$ is the circle $x^2 + y^2 = 4$ which has parametrization $x = 2\cos(s)$, $y = 2\sin(s)$, and $z = 0$, for $0 \leq s \leq 2\pi$.



```

In[13]:=
  pvf=vf[x,y,z] /. x->2 Cos[s] /.
  y->2 Sin[s] /. z->0
Out[13]=
      2
{4 Sin[s] , 2 Cos[s], 4 Cos[s] - 2 Sin[s]}
In[14]:=
  r[s_]={2 Cos[s], 2 Sin[s], 0}
Out[14]=
{2 Cos[s], 2 Sin[s], 0}

```

Then replace each x in \mathbf{vf} by $2\cos(s)$, each y in \mathbf{vf} by $2\sin(s)$, and each z in \mathbf{vf} by 0 .

To evaluate the line integral directly, notice that a parametrization of C is given by $\mathbf{r}(s) = \{2\cos(s), 2\sin(s), 0\}$, where $0 \leq s \leq 2\pi$.

StokesTheorem

```

In[15]:=
  r'[s]
Out[15]=
  {-2 Sin[s], 2 Cos[s], 0}

In[16]:=
  pvf.r'[s]
Out[16]=
  4 Cos[s]^2 - 8 Sin[s]^3

In[17]:=
  Integrate[pvf.r'[s], {s, 0, 2Pi}]
Out[17]=
  4 Pi

```

r'[s]
computes the derivative of r with respect to s.

pvf.r'[s]
computes the dot product of pvf and r'[s].

Integrate[pvf.r'[s], {s, 0, 2Pi}]
computes the value of the line integral.

■ 5.4 Saving Results for Future *Mathematica* Sessions

Beginning users of *Mathematica* quickly notice that in order to use results from a previous *Mathematica* session, they must first be re-calculated. The purpose of this example is to illustrate how results can be saved for future use.

■ **Application:** Constructing a Table of Zeros of Bessel Functions

In this example, we will create a table of the first six zeros of the Bessel functions of the first kind,

$J_0(x)$, $J_1(x)$, $J_2(x)$, $J_3(x)$, $J_4(x)$, and $J_5(x)$ then save the resulting table of numbers

in a file. Consequently, for future calculations involving zeros of Bessel functions, we need only use the table of numbers we have already created instead of re-computing the zeros. This will thus save not only substantial time but also substantial memory.

The built-in *Mathematica* command `BesselJ [alpha ,x]` represents $J_{\text{alpha}}(x)$, and, hence

will be used in the construction of this table of zeros. This command as well as Bessel functions are discussed in more detail in Chapter 6.

We begin by looking at a graph of `BesselJ [0 ,x]` on the interval [0,25] and observing the first six zeros. We create a list of six numbers corresponding to initial guesses of the first six zeros and then use `FindRoot` to approximate the first six zeros.

The list `guess[0]`, a list of initial guesses of the first six zeros of `BesselJ[0, x]` is used with `FindRoot` to approximate these zeros. Since this list is used in conjunction with `Table`, an approximation of the first zero is obtained with `guess[0][[1]]`, an approximation of the second zero is obtained with `guess[0][[2]]`, and so forth. This produces a list of approximations of the first six zeros of `BesselJ[0, x]` called `bz[0]`.

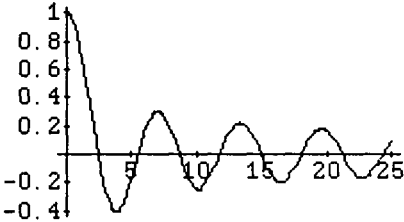
□
□ □ □
InputOutput

```

In[1]:=
Plot[BesselJ[0, x], {x, 0, 25}]

```

graphs $J_0(x)$ on the interval $[0, 25]$.



```

Out[1]=
-Graphics-

```

```

In[2]:=
guess[0] = {2.14, 5.59, 8.72, 11.8, 15, 18.2};

```

```

In[3]:=
Clear[bz]
bz[0] = Table[FindRoot[BesselJ[0, x] == 0,
  {x, guess[0][[i]]}], {i, 1, 6}]

```

```

Out[3]=
{{x -> 2.40483}, {x -> 5.52008},
 {x -> 8.65373}, {x -> 11.7915},
 {x -> 14.9309}, {x -> 18.0711}}

```

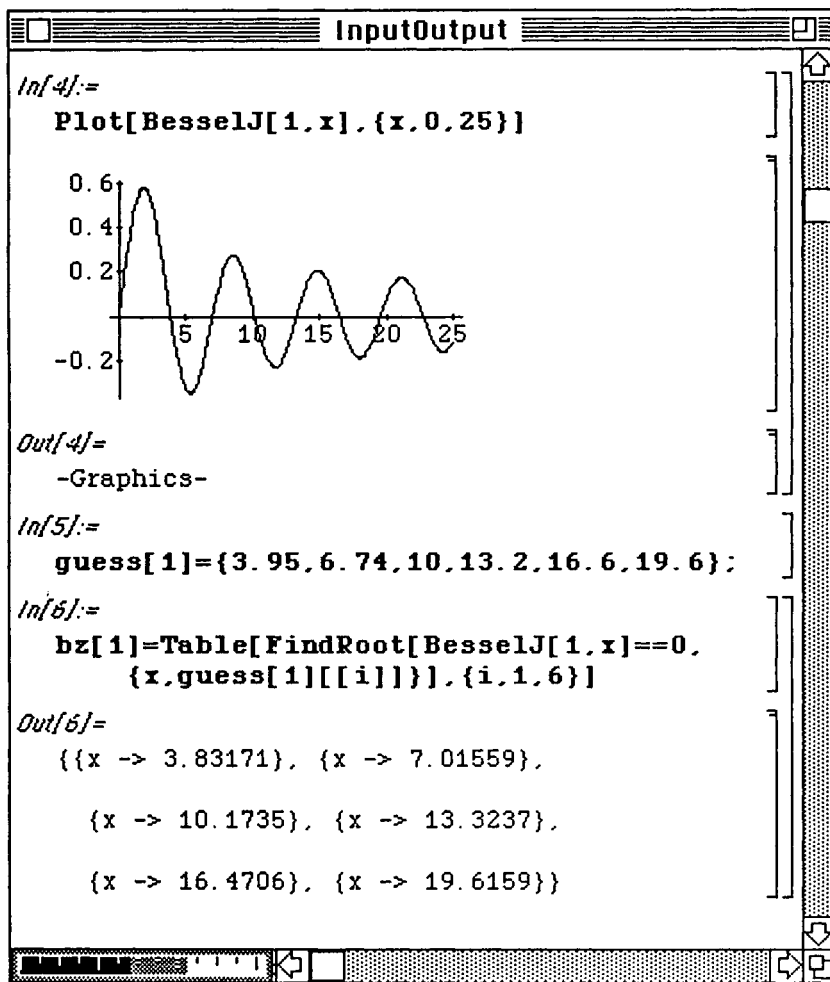
□ □ □
□ □ □

We use the graph of $J_0(x)$ to obtain rough approximations of its first six zeros. We will use these approximations in the command **FindRoot** to locate more accurate approximations of the first six zeros of $J_0(x)$.

guess[0] is a list of six numbers corresponding to those we will use as initial approximations of the zeros of $J_0(x)$ in the command **FindRoot**.

computes a table of six numbers corresponding to approximations of the first six zeros of $J_0(x)$.

We then proceed to `BesselJ[1, x]`:



In the same manner as above, we use the graph of $J_1(x)$ to obtain our initial guesses of its first six zeros.

guess[1]
is a list of six numbers corresponding to our initial guesses of the first six zeros of $J_1(x)$.

bz[1]
is a list of approximations of the first six zeros of $J_1(x)$.

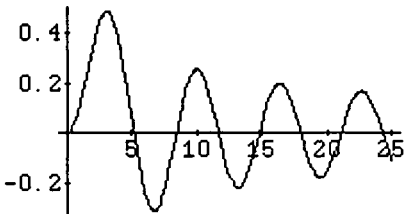
We repeat the procedure for `BesselJ[2, x]`, `BesselJ[3, x]`, `BesselJ[4, x]` and `BesselJ[5, x]`:

InputOutput

```

In[7]:=
Plot[BesselJ[2, x], {x, 0, 25}]

```



*graphs $J_2(x)$ on the interval $[0, 25]$. As before, we use the resulting graph to obtain initial approximations of the zeros for later use in with the command **FindRoot**.*

```

Out[7]=
-Graphics-

```

```

In[8]:=
guess[2] = {5.26, 8.55, 11.7, 15.18, 1.21, 1};

```

```

In[9]:=
bz[2] = Table[FindRoot[BesselJ[2, x] == 0,
  {x, guess[2][[i]]}], {i, 1, 6}]

```

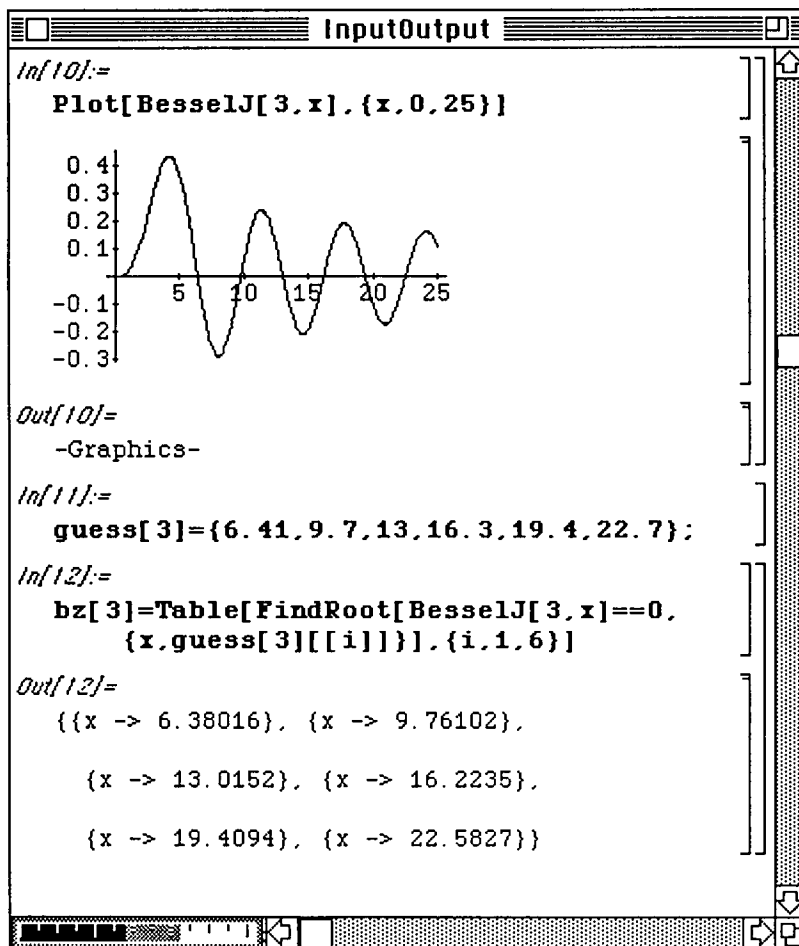
bz[2]
is a list of six numbers corresponding to the first six zeros of $J_2(x)$.

```

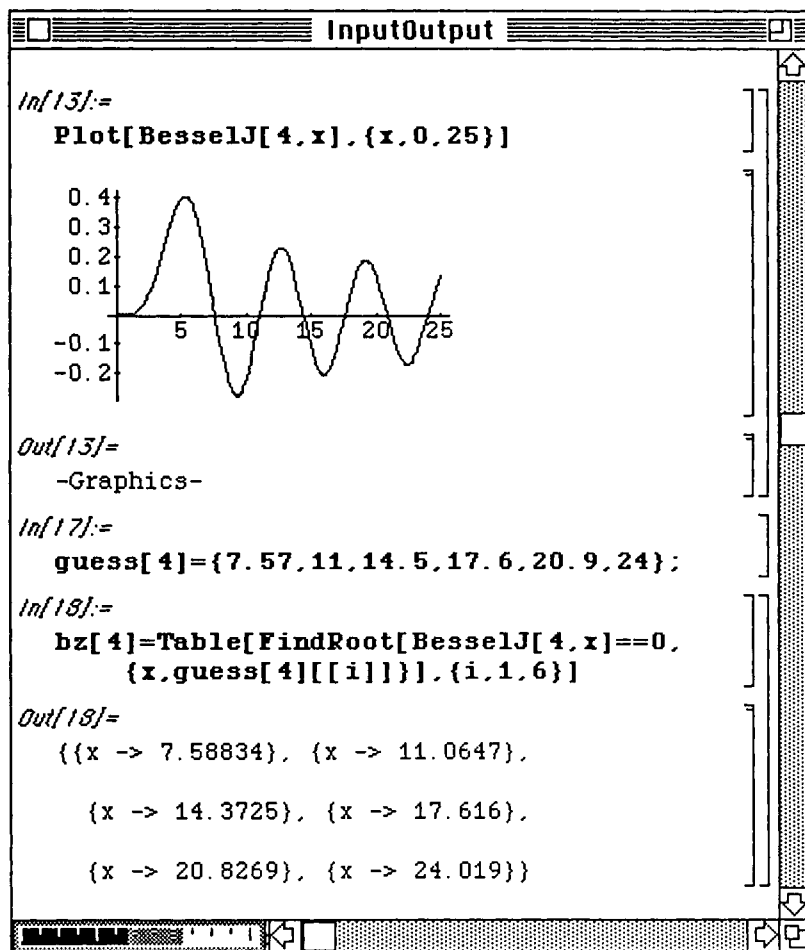
Out[9]=
{{x -> 5.13562}, {x -> 8.41724},
 {x -> 11.6198}, {x -> 14.796},
 {x -> 17.9598}, {x -> 21.117}}

```

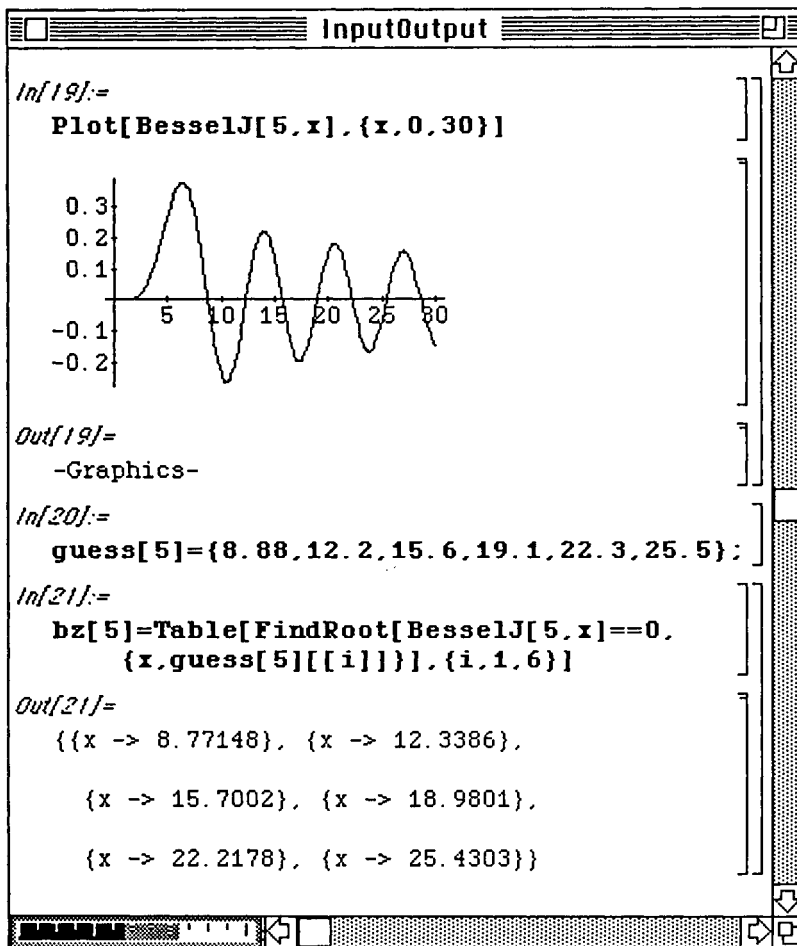
The first six zeros of `BesselJ[3, x]` are found below:



Similarly, the zeros of `BesselJ[4, x]` are found.



Finally, a list of the first six zeros of `BesselJ[5, x]` is determined.



An alternative approach would have been to compile a list of initial guesses for the first six zeros by looking at the graphs of `BesselJ[0, x]`, `BesselJ[1, x]`, `BesselJ[2, x]`, `BesselJ[3, x]`, `BesselJ[4, x]` and `BesselJ[5, x]` and then to use `FindRoot`. In order to apply this alternate approach, a table of initial guesses must be compiled. This is done in `starts` below using the previously defined lists `guess[n]` for $n = 0, 1, 2, 3, 4, 5$. Then, a list of the first six zeros of the Bessel functions of the first kind

$$J_0(x), J_1(x), J_2(x), J_3(x), J_4(x), \text{ and } J_5(x)$$

are computed in `besselzeros`. Notice that `besselzeros` is a list of lists, so the n th zero of the Bessel function `BesselJ[m, x]` is extracted with `besselzeros[[m+1, n, 1, 2]]`.

```

In[19]:=
  starts=Table[guess[j].{j,0.5}
Out[19]=
  {{(2.14, 5.59, 8.72, 11.8, 15.
    18.2)}, {3.95, 6.74, 10, 13.2,
    16.6, 19.6)},
  {5.26, 8.55, 11.7, 15, 18.1,
    21.1}, {6.41, 9.7, 13, 16.3,
    19.4, 22.7)},
  {7.57, 11, 14.5, 17.6, 20.9,
    24}, {8.88, 12.2, 15.6, 19.1,
    22.3, 25.5}}

In[20]:=
  besselzeros=Table[FindRoot[BesselJ[j, x]==0,
    {x, starts[[j+1, i]]}], {j,0,5}, {i,1,6}]
  Short[besselzeros,5]
Out[20]//Short=
  {{{x -> 2.40483}, {x -> 5.52008},
    {x -> 8.65373}, {x -> 11.7915},
    {x -> 14.9309}, {x -> 18.0711}}\
  <<5>>}
  
```

is a table consisting of {guess[0], guess[1], ..., guess[5]}. Notice that starts is a list of lists. In fact, starts[[m,n]] corresponds to the initial guess of the nth zero of the Bessel function $J_m(x)$.

computes a table of approximations of the i th zero of the Bessel function $J_j(x)$, using the command FindRoot for $j=0,1,2,3,4$, and 5; $i=1,2,3,4,5$, and 6. The resulting list is named besselzeros and an abbreviated form (consisting of no more than five lines) is displayed.

The first list in `besselzeros` is the list of zeros of `BesselJ[0, x]`. Hence, `besselzeros[[1]]` gives this list. Since the indices are shifted, `besselzeros[[j+1]] = bz[j]` where `bz[j]` is the list of the first six zeros of `BesselJ[j, x]` found earlier. We verify that the results obtained by the alternative approach are the same as those found by the previous approach.

The notation commonly used to denote the n th zero of the Bessel function $J_m(x)$ is α_{mn} .

Hence, the function `alpha[m,n]` is defined below in terms of the values in `bz[n]` for later use.

The screenshot shows a Mathematica InputOutput window with the following content:

```

In[21]:=
  BesselZeros[[1]]==bz[0]
Out[21]=
  True
In[22]:=
  BesselZeros[[3]]==bz[2]
Out[22]=
  True
In[23]:=
  BesselZeros[[5]]==bz[4]
Out[23]=
  True
In[24]:=
  BesselZeros[[5,3,1,2]]
Out[24]=
  14.3725
In[25]:=
  alpha[m_,n_] := BesselZeros[[m]][[n,1,2]]

```

Three callout boxes provide additional information:

- Box 1 (top):** *To verify the same result is produced, we verify that `BesselZeros[[k]]` and `bz[k-1]` are the same for $k=1, 2, \dots, 6$.*
- Box 2 (middle):** *The same result would be obtained by entering `BesselZeros[[4]][[3,1,2]]`. This can be interpreted as the third zero of the Bessel function $J_4(x)$.*
- Box 3 (bottom):** *defines `alpha[m,n]` to be the n th number of the list `BesselZeros[[m]]`. Based on our preceding work, m must be between 0 and 5; n must be between 1 and 6.*

The table `zeros` is a table of the first six zeros of the first six Bessel functions.

The command `Table[alpha[m,n], {m,0,5}, {n,1,6}]>>BesselZeros` first computes the table of numbers `Table[alpha[m,n], {m,0,5}, {n,1,6}]` and then writes the results to the file `BesselZeros`.

It is important to notice that if the file `BesselZeros` does not exist, it is created; if it does exist, it is written over. To **append** the results of a command to an existing file, the form of the command is `command>>>output file`.

The function `alphaalt[m,n]` can then easily find the n th zero of `BesselJ[m,x]` from the list `besselzeros`. These zeros do not have to be recomputed with this approach and are located in the table `zerosalt`. The fact that identical values are contained in `zeros` and `zerosalt` is verified.

```

In[26]:=
  zeros=Table[alpha[m,n],{m,0,5},{n,1,6}]
  TableForm[zeros]
Out[26]:=TableForm=
  2.40483  5.52008  8.65373  11.7915
          14.9309  18.0711
  3.83171  7.01559  10.1735  13.3237
          16.4706  19.6159
  5.13562  8.41724  11.6198  14.796
          17.9598  21.117
  6.38016  9.76102  13.0152  16.2235
          19.4094  22.5827
  7.58834  11.0647  14.3725  17.616
          20.8269  24.019
  8.77148  12.3386  15.7002  18.9801
          22.2178  25.4303
In[27]:=
  Table[alpha[m,n],{m,0,5},{n,1,6}] >> besselzeros
In[28]:=
  alphaalt[m_,n_] := besselzeros[[m+1,n,1,2]]
In[29]:=
  zerosalt=Table[alphaalt[m,n],{m,0,5},{n,1,6}];
In[30]:=
  zerosalt==zeros
Out[30]=
  True

```

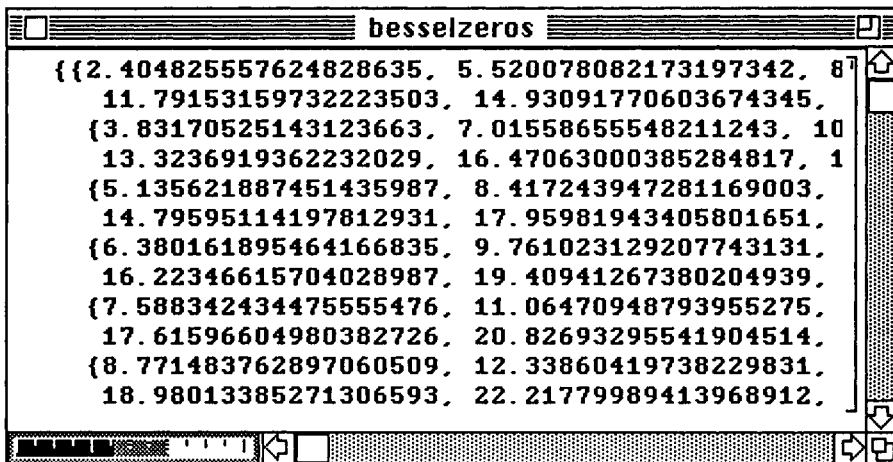
creates a table zeros and expresses the result in Table Form.

creates the same table as above and writes the output to the file besselzeros.

The same results would have been obtained using the table besselzeros.

The file `besselzeros` is a nested list of numbers; it does not rely on previous calculations.

*The file `besselzeros`
is in the folder `Mathematicaf`.*



In this case, we modify `besselzeros` by converting the initialization cell to an ordinary input cell (by selecting **Input Cell** from **Cell Style** under the **Style** heading on the *Mathematica Menu*), naming the table `zbn` and defining `bzero[m, n]`. For future use, we need only open the file `besselzeros`, enter its input cells, and the function `bzero[m, n]` will give the n th zero of the m th Bessel function, $J_m(x)$.

The calculation of the zeros of the Bessel functions are important in many problems in applied mathematics, so the procedures described here can be quite useful. Notice how these values are easily obtained using `bzero[m, n]`.

In order to use the values located in `zbj`, the input cell containing `zbj` and the input cell containing the definition of `bzero` must first be entered.

besselzeros

We modify the file `besselzeros` by naming the table `zbj`.

`In[32]:=`
`zbj={{2.404825557624828635, 5.11.79153159732223503, 14.9: {3.83170525143123663, 7.015! 13.3236919362232029, 16.47({5.135621887451435987, 8.41' 14.79595114197812931, 17.9! {6.380161895464166835, 9.76: 16.22346615704028987, 19.4({7.588342434475555476, 11.0(17.61596604980382726, 20.8; {8.771483762897060509, 12.3: 18.98013385271306593, 22.2: 25.43034115340598979}}};`

The function `bzero[m, n]` gives an approximation of the n th zero of the Bessel function $J_m(x)$.

`In[33]:=`
`bzero[m_, n_] := zbj[[m+1, n]]`

`In[34]:=`
`bzero[0, 1]`

is an approximation of the first zero of $J_0(x)$.

`Out[34]=`
`2.404825557624828635`

`In[35]:=`
`bzero[1, 2]`

is an approximation of the second zero of $J_1(x)$.

`Out[35]=`
`7.01558655548211243`

In general, `bzero[m, n]` makes sense as long as m is between 0 and 5; n is between 1 and 6.

■ An Alternative Method

In the previous example, we saw how to create a table of numbers and save them in a separate file for future use. The command `Table[{Cos[j], Sin[i]}, {i, 1, 3}, {j, 1, 3}]>>outputfile` creates a table of order pairs `{Cos[j], Sin[i]}` for $i=1,2,3$ and $j=1,2,3$, creates (or erases) `outputfile` and places the table in `outputfile`. Notice that in the first example below that the results are given in an output cell which cannot be accessed. Hence, if the output is to be saved for later use another approach may be more desirable. The second example illustrates such an approach. `Print` is used within the `Table` command so that the results are given in a print cell which can be accessed with the cursor. This gives the user a second method by which to save a file for future use.

```

In[35]:=
  Table[{Cos[j], Sin[i]}, {i, 1, 3}, {j, 1, 3}]
Out[35]=
  {{{Cos[1], Sin[1]}, {Cos[2], Sin[1]},
    {Cos[3], Sin[1]}}},
  {{{Cos[1], Sin[2]}, {Cos[2], Sin[2]},
    {Cos[3], Sin[2]}}},
  {{{Cos[1], Sin[3]}, {Cos[2], Sin[3]},
    {Cos[3], Sin[3]}}}

In[36]:=
  Table[Print[{Cos[j], Sin[j]}], {i, 1, 3},
        {j, 1, 3}]
  {Cos[1], Sin[1]}
  {Cos[2], Sin[2]}
  {Cos[3], Sin[3]}
  {Cos[1], Sin[1]}
  {Cos[2], Sin[2]}
  {Cos[3], Sin[3]}
  {Cos[1], Sin[1]}
  {Cos[2], Sin[2]}
  {Cos[3], Sin[3]}
Out[36]=
  {{Null, Null, Null}, {Null, Null, Null},
   {Null, Null, Null}}

```

In general, output cells cannot be modified. Notice that the cursor is a "bullseye" when it is within an output cell, indicating that one cannot type within the output cell.

On the other hand, the same results can be obtained using the `Print` command. Notice that a `Print` cell can be modified by typing within it and/or converting it to an input cell.

Chapter 6

Applications Related to Ordinary and Partial Differential Equations

- *Mathematica* can perform calculations necessary when computing solutions of various differential equations and, in some cases, can be used to find the exact solution of certain differential equations using the built-in command **DSolve**. In addition, Version 2.0 contains the built-in command **NDSolve** which can be used to obtain numerical solutions of other differential equations. The purpose of Chapter 6 is to illustrate various computations *Mathematica* can perform when solving differential equations.
- Commands introduced and discussed in this chapter from Version 1.2 include:

Differential Equations:

DSolve[*differentialequation*, *function*, *variable*]
DSolve[{*differentialequations*}, {*functions*}, *variable*]
DSolve[{*de*, *initialcond*}, ...]
DSolve[{*des*, *initialconds*}, ...]

Special Functions:

BesselJ[*alpha*, *x*]
BesselY[*alpha*, *x*]

Programming

Block[{*localvariables*}, *procedure*]

Algebraic Operations:

Variables[*expression*]
Exponent[*polynomial*, *variable*]
Coefficient[*poly*, *var*, *i*]

Other Operations:

Flatten[*list*]
Print[*expression*]
Dt[*function*]
==

Trigonometric Operations:

TrigExpand[*expression*]
ComplexToTrig[*expression*]

- Commands introduced and discussed in this chapter from Version 2.0 include:

NDSolve
InterpolatingFunction
Evaluate

Commands from previous chapters are frequently used.

- Applications discussed in this chapter include the Falling Bodies Problem, Spring Problems, Classification of Equilibrium Points, and the Wave Equation.

6.1 Linear Equations

The general solution of the linear equation $\frac{dy}{dx} + P(x)y = f(x)$, where P and f are continuous on the interval I , is $y = e^{-\int P(x)dx} \int e^{\int P(x)dx} f(x)dx + c_1 e^{-\int P(x)dx}$.

Mathematica can solve equations of this type with **DSolve**. However, since solutions of first-order linear equations obviously depend on the computation of an integral, if you are using Version 1.2, **IntegralTables.m** must be loaded before trying to solve any differential equations.

o If you are using Version 2.0, **IntegralTables.m** is automatically loaded at startup.

After this is done, the linear equation given above is solved with the command **DSolve[y'[x]+P[x]y[x]==f[x],y[x],x]** where the functions $P(x)$ and $f(x)$ are usually directly entered in the **DSolve** command. Notice that the command consists of three parts: the differential equation; the dependent variable (or solution), **y[x]**; and the independent variable, **x**. Also notice that the dependent variable must be entered as **y[x]** each time it appears in the differential equation. Otherwise, **DSolve** will yield a meaningless result. Several examples are given below to illustrate the use of **DSolve**.

□ **Example:**

Solve the first order linear differential equation $(1+x^2)\frac{dy}{dx} + xy = -(x^3+x)$.

The solution, called **sol**, is easily found below with

DSolve[(1+x^2)y'[x]+x y[x]==-(x^3+x),y[x],x].

Notice that the output for this command is a list of one element (obtained with **sol[[1]]**),

$$\{ y[x] \rightarrow \frac{1-(1+x^2)^{3/2}}{3E^{\text{Log}[1+x^2]/2}} + \frac{C[1]}{E^{\text{Log}[1+x^2]/2}} \}.$$

This is also a list of one element (obtained with **sol[[1,1]]**),

$$y[x] \rightarrow \frac{1-(1+x^2)^{3/2}}{3E^{\text{Log}[1+x^2]/2}} + \frac{C[1]}{E^{\text{Log}[1+x^2]/2}}$$

which is composed of two parts, **y[x]** and the expression following the arrow. Of course, the second part of this element is the portion of interest since it gives the formula of the solution. Therefore, in order to extract this formula, the command **sol[[1,1,2]]** is used. (**sol[[1,1,1]]** yields **y[x]**.)

Note: In most instances, the ability to extract the formula for the solution will be of great importance. In order to analyze solutions to most differential equations, obtaining the formula of the solution is necessary. One alternative is to define the solution as a function once it has been found with **DSolve**. However, many solutions are rather complicated, so typing is cumbersome and mistakes are likely. Hence, the logical choice for obtaining the formula is by extracting it from the output list. At first, the technique of extracting solutions may seem difficult to understand, but it should become clearer after several examples.

```

LinearEquations
In[1]:=
  <<IntegralTables.m
Out[1]=
  Integrator`
In[67]:=
  sol=DSolve[(1+x^2)Y'[x]+x Y[x]==-(x^3+x),
    Y[x],x]
Out[67]=
  {{Y[x] ->
    1 - (1 + x )2 3/2
    ----- +
    2
    Log[1 + x ]/2
    3 E
    -----}}
    C[1]
    2
    Log[1 + x ]/2
    E
In[68]:=
  sol[[1,1,2]]
Out[68]=
    1 - (1 + x )2 3/2
    ----- + -----
    2
    Log[1 + x ]/2      Log[1 + x ]/2
    3 E                E
  
```

Be sure that the package **IntegralTables.m** has been loaded prior to using **DSolve** to solve first order linear differential equations.

solves the equation $(1+x^2)y'+xy = -(x^3+x)$ for $y(x)$. The result of executing the command is a list which we name **sol**.

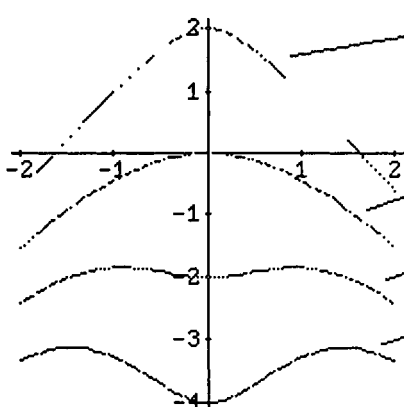
sol[[1,1,2]] extracts the solution from the list **sol**.

Solutions for various values of the constant $C[1]$ can be graphed. First, the graphs for $C[1] = -4, -2, 0,$ and 2 are requested in a single input cell using different **GrayLevel** assignments. Each plot is assigned a name so that the **Show** command can be used in the same cell to plot these four graphs simultaneously. Since all of the commands are in the same cell, only the output from the last command is shown.

LinearEquations
⌵

```

In[69]:=
pn4=Plot[sol[[1,1,2]] /. C[1]->-4, {x, -2, 2},
PlotStyle->GrayLevel[.1],
DisplayFunction->Identity]
pn2=Plot[sol[[1,1,2]] /. C[1]->-2, {x, -2, 2},
PlotStyle->GrayLevel[.2],
DisplayFunction->Identity]
p0=Plot[sol[[1,1,2]] /. C[1]->0, {x, -2, 2},
PlotStyle->GrayLevel[.3],
DisplayFunction->Identity]
p2=Plot[sol[[1,1,2]] /. C[1]->2, {x, -2, 2},
PlotStyle->GrayLevel[.4],
DisplayFunction->Identity]
Show[pn4, pn2, p0, p2, AspectRatio->1,
DisplayFunction->DisplayFunction]
    
```



We may graph the solution for various values of C[1].

Corresponds to p2.

Corresponds to p0.

Corresponds to pn2.

Corresponds to pn4.

```

Out[69]=
-Graphics-
    
```

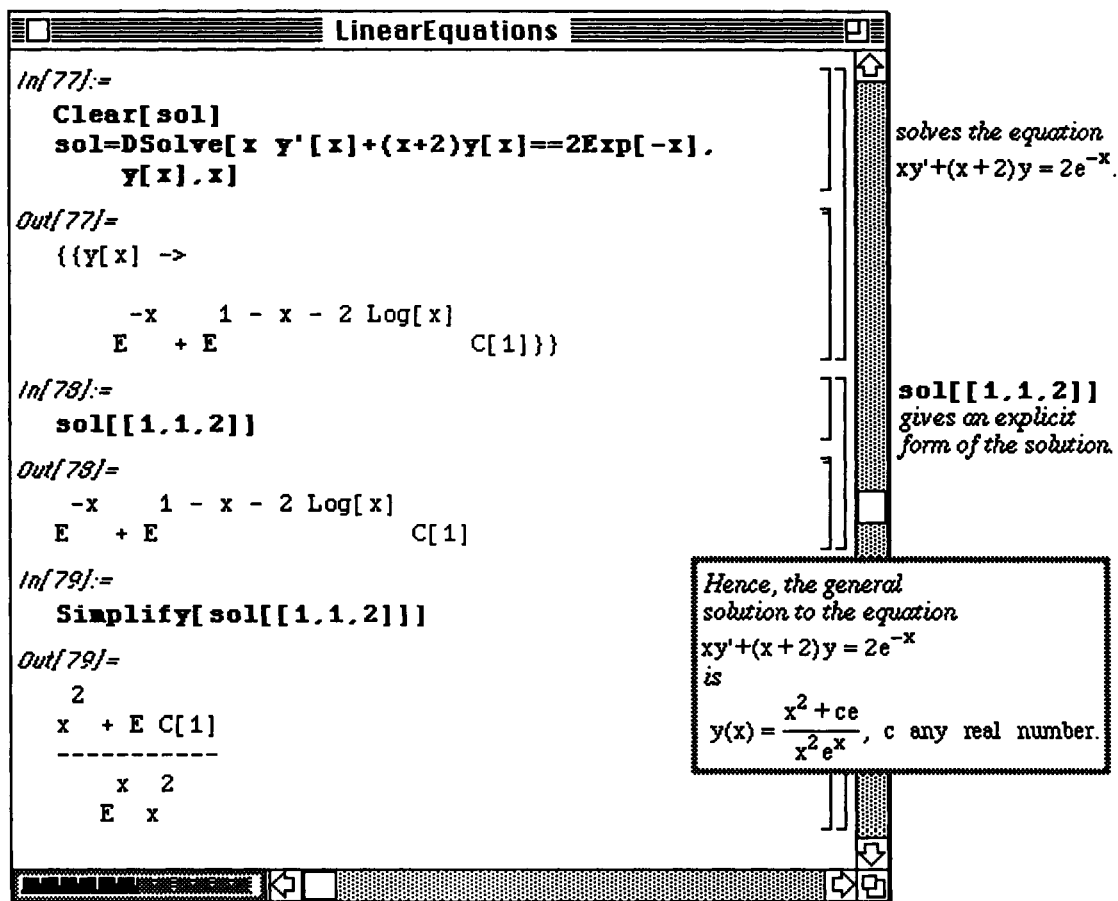
⏪
⏩

Mathematica's DSolve command can also be used to solve initial value problems. The command is altered slightly to include the initial condition.

□ Example:

Solve: $x \frac{dy}{dx} + (x+2)y = 2e^{-x}$, $y(1) = 0$.

The differential equation can be solved with `DSolve` as in the previous example. However, this gives the general solution to the problem which represents a family of solutions. The solution to the initial value problem the one solution which passes through the point (1,0).



```

In[77]:=
Clear[sol]
sol=DSolve[x Y'[x]+(x+2)Y[x]==2Exp[-x],
Y[x], x]
Out[77]=
{{Y[x] ->
  -x      1 - x - 2 Log[x]
  E      + E          C[1]}}
In[78]:=
sol[[1,1,2]]
Out[78]=
  -x      1 - x - 2 Log[x]
  E      + E          C[1]
In[79]:=
Simplify[sol[[1,1,2]]]
Out[79]=
  2
  x  + E C[1]
  -----
  x  2
  E  x

```

solves the equation $xy'+(x+2)y = 2e^{-x}$.

`sol[[1,1,2]]` gives an explicit form of the solution.

Hence, the general solution to the equation $xy'+(x+2)y = 2e^{-x}$ is $y(x) = \frac{x^2 + ce}{x^2 e^x}$, c any real number.

To find the solution to the initial value problem with `DSolve`, the initial condition must be entered in the `DSolve` command. This is accomplished in the following way:

```
DSolve[{x Y'[x]+(x+2)Y[x]==2Exp[-x], Y[1]==0}, Y[x], x].
```

Notice that the initial condition is placed in "curly" brackets, $\{ \}$, along with the differential equation. Otherwise, the command is unchanged. Also note that a double equals sign is used in the initial condition. The solution is found below and named `sol`. The formula for $y[x]$ is then extracted and simplified in a single command. The expression which results is called `simsol` for use in the `Plot` command which follows.

LinearEquations

```

In[2]:=
Clear[sol]
sol=DSolve[{x y'[x]+(x+2)y[x]==2Exp[-x],
           y[1]==0},y[x],x]

Out[2]=
      -x      -x - 2 Log[x]
{{y[x] -> E      E      }}

In[3]:=
simsol=Simplify[sol[[1,1,2]]]

Out[3]=
      2
-1 + x
-----
      x 2
      E x

In[4]:=
Plot[simsol,{x,-5,2},PlotRange->{-20,20}]

Out[4]=
-Graphics-

```

solves the initial value problem
 $xy'+(x+2)y=2e^{-x}$, $y(1)=0$.

Simplify[sol[[1,1,2]]]
extracts and simplifies the solution of
 $xy'+(x+2)y=2e^{-x}$, $y(1)=0$.
We conclude that the solution of
 $xy'+(x+2)y=2e^{-x}$, $y(1)=0$.
is $y(x)=\frac{x^2-1}{x^2e^x}$.

For convenience, we name the solution **simsol**.

Finally, we graph the solution on the interval [-5,2]. Notice that the solution is undefined when $x=0$. The option **PlotRange->{-20,20}** *specifies that the y-values displayed are between -20 and 20.*

■ **Application:** The Falling Body Problems

A useful application of first-order differential equations is solving problems encountered in mechanics. One such problem is as follows:

A body falls through the air towards the earth. In such a circumstance, the body is subjected to a certain amount of air resistance (which in some cases is proportional to the body's velocity). The objective is to determine the velocity and the distance fallen at time t seconds.

Mathematica can be quite useful in solving problems of this type. To illustrate how these falling body problems are solved, consider the following problem.

An object weighing 32 pounds is released from rest 50 feet above the surface of a calm lake. The air resistance (in pounds) is given by $2v$, where v is the velocity (in feet/sec). After the object passes beneath the surface, the water resistance (in pounds) is given by $6v$. Further, the object is then buoyed up by a buoyancy force of 8 pounds. Find the velocity of the object 2 seconds after it passes beneath the surface of the lake.

This problem is made up of two parts. First, the forces acting on the object before it reaches the surface of the lake must be considered. Then, the set of forces which act upon the object beneath the lake's surface must be determined in order to solve the problem. Using Newton's second law, the initial value problem which determines the object's

velocity above the surface is: $\frac{dv}{dt} = 32 - 2v$, $v(0) = 0$.

DSolve can be used to solve this initial value problem. This is done below. The velocity is then extracted from the resulting expression with **deq1**[[1,1,2]] and named **vell**.

Be sure that the package `IntegralTables.m` has been loaded before using the command `DSolve`, unless using Version 2.0.

```

In[2]:=
<<IntegralTables.m
deq1=DSolve[{v'[t]==32-2 v[t],
            v[0]==0}, v[t], t]

Out[2]=
      2 t
      -16 + 16 E
      { {v[t] -> -----} }
            2 t
            E

In[3]:=
vell=deq1[[1,1,2]]

Out[3]=
      2 t
      -16 + 16 E
      -----
            2 t
            E
  
```

Solves the initial value problem $v' = 32 - 2v$, $v(0) = 0$.

Since the solution to the initial value problem $v' = 32 - 2v$, $v(0) = 0$ is expressed as a list, the solution is extracted with the command `deq1[[1,1,2]]` and named `vell`.

Before determining the velocity beneath the lake's surface, the object's velocity at the point of impact must be found. Therefore, the time at which the object hits the surface of the lake must be calculated by integrating the velocity $v(t)$ to obtain the object's position function $x(t)$ using the initial position $x(0) = 0$. (Recall that $x'(t) = v(t)$) This calculation is carried out below. Note that the position function is extracted and assigned the name **position1**.

```

In[4]:=
  deq2=DSolve[{x'[t]==vel1, x[0]==0}, x[t], t]
Out[4]=
  {{x[t] ->
    -8 + ---- - 8 Log[E-2 t ]}}
           2 t
           E

In[5]:=
  position1=deq2[[1,1,2]]
Out[5]=
  -8 + ---- - 8 Log[E-2 t ]
       2 t
       E

```

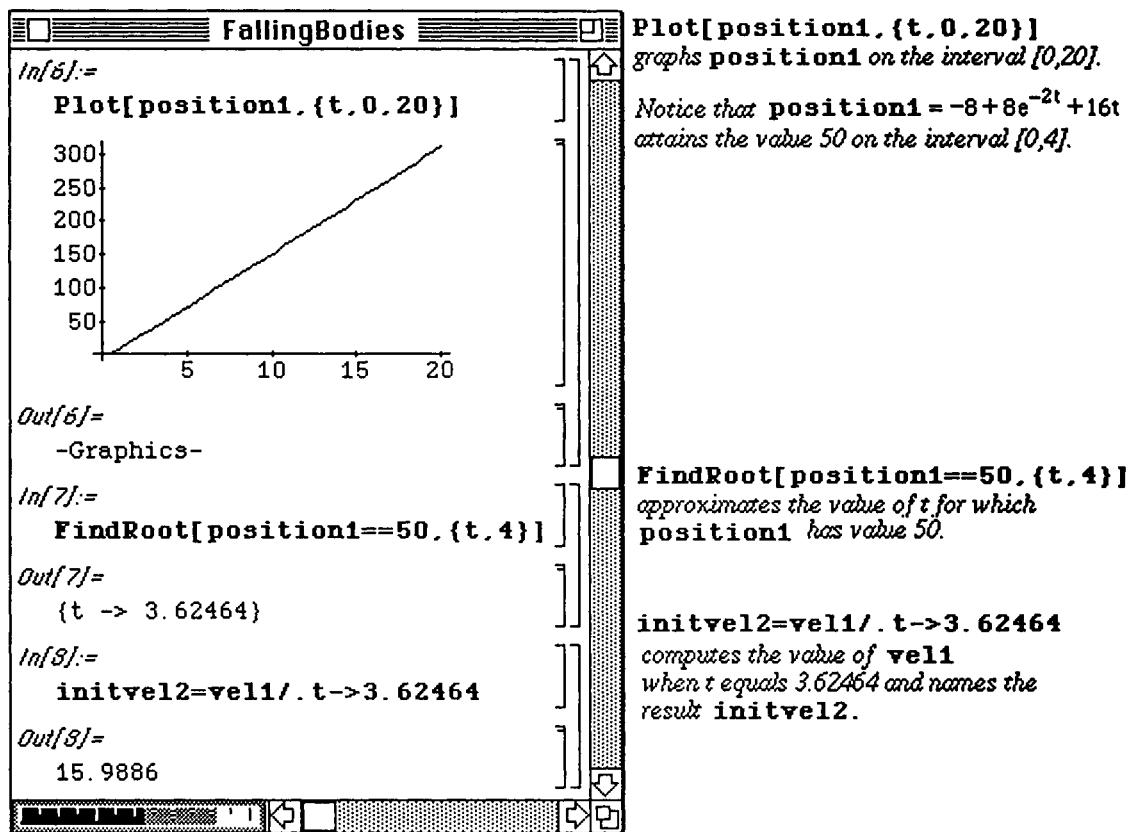
solves the initial value problem

$$x' = \frac{16e^{2t} - 16}{e^{2t}}, \quad x(0) = 0.$$

vel1

*Since **deq2** is a list, execute the command **deq2[[1,1,2]]** to obtain an explicit form of the solution. The result is named **position1**.*

Then, the value of t when $x(t) = 50$ is computed (i.e., the time when the object hits the lake's surface). This is accomplished by making use of **FindRoot**. Since **FindRoot** depends on an initial guess of the solution to the equation, **position1** is graphed in order to obtain an approximate value of t when $x(t) = 50$. The position appears to equal 50 near $t = 4$. Hence, **FindRoot** is used to determine the root of the equation $x(t) = 50$ with the initial-guess $t=4$. Therefore, the velocity at the point of impact is found by substituting the value obtained with **FindRoot**, $t = 3.62464$, into the velocity function of the object above the surface, **vel1**. This is accomplished with **vel1/.t->3.62464**, and the resulting expression is named **initvel2**.



Now that the velocity of the object at its point of impact is known, the initial value problem to determine the velocity

beneath the lake's surface is: $\frac{dv}{dt} = 24 - 6v$, $v(0) = 15.9886$.

can be solved. This problem is solved below with `DSolve`. (Note that `initvel2` is used in the initial condition instead of entering the numerical value.) Once solved, the exact value of the velocity at $t=2$ seconds is calculated by extracting the velocity formula from `deq3` and evaluating it at $t=2$ with `deq3[[1,1,2]]/.t->2`. The numerical approximation of the velocity (4.00007 seconds) is then computed.

```

In[9]:=
deq3=DSolve[{v'[t]==24-6 v[t],
             v[0]==initvel2},v[t],t]

Out[9]=
{{v[t] ->  $\frac{15.9886}{e^{6t}} + \frac{-4 + 4e^{6t}}{6t}$ }}

In[10]:=
deq3[[1,1,2]]/.t->2

Out[10]=
 $\frac{15.9886}{e^{12}} + \frac{-4 + 4e^{12}}{12}$ 

In[11]:=
N[*]

Out[11]=
4.00007

```

solves the initial value problem
 $v' = 24 - 6v$, $v(0) = \text{initvel2} = 15.9886$.

`deq3[[1,1,2]]/.t->2`
 computes the exact value of
 $\text{deq3}[[1,1,2]] = 11.9886e^{-6t} + 4$
 when t equals 2.

`N[*]`
 computes a numerical approximation
 of the previous output.

■ 6.2 Exact Differential Equations

Certain types of nonlinear ordinary differential equations such as exact differential equations can be solved with the aid of *Mathematica*. The differential equation $M(x,y) dx + N(x,y) = 0$ is called an **exact differential equation** in

a domain D if there exists a function F such that $\frac{\partial F(x,y)}{\partial x} = M(x,y)$ and $\frac{\partial F(x,y)}{\partial y} = N(x,y)$

for all (x,y) in D . In order to determine if an equation is exact, the following well-known theorem can be used: Let M and N have continuous first partial derivatives at all points (x,y) in a domain D .

If $\frac{\partial M(x,y)}{\partial y} = \frac{\partial N(x,y)}{\partial x}$ for all (x,y) in D , then the differential equation $M(x,y)dx + N(x,y)dy = 0$ is exact

Hence, if the differential equation is exact the total differential of F , $dF(x,y) = M(x,y) dx + N(x,y)dy = 0$.

Therefore, the solution of the exact equation is $F(x,y) = \text{Constant}$. The method by which $F(x,y)$ is determined is illustrated with the following example.

□ Example:

Solve the initial value problem $\frac{1+8xy^{2/3}}{x^{2/3}y^{1/3}} dx + \frac{2x^{4/3}y^{2/3} - x^{1/3}}{y^{4/3}} dy = 0, y(1) = 8.$

First, we must verify that this differential equation is exact. This is done by entering the functions $M(x,y)$ and $N(x,y)$. (To avoid confusion with any built-in *Mathematica* function or constant, small letters are used in these definitions.) Next, the partial derivative of $M(x,y)$ with respect to y , $D[m[x,y],y]$, must be calculated so that it can be compared with the partial derivative of $N(x,y)$ with respect to x , $D[n[x,y],x]$. These derivatives are conveniently named my and nx .

ExactDiffEqn

In[7]:=

$$m[x_,y_]=(1+8x y^{2/3})/(x^{2/3}y^{1/3})$$

$$n[x_,y_]=(2x^{4/3}y^{2/3}-x^{1/3})/(y^{4/3});$$

In[8]:=

$$my=D[m[x,y],y]$$

Out[8]=

$$-\frac{(1+8xy^{2/3})}{3x^{2/3}y^{4/3}} + \frac{16x^{1/3}}{3y^{2/3}}$$

In[9]:=

$$nx=D[n[x,y],x]$$

Out[9]=

$$-\frac{1}{3x^{2/3}} + \frac{8xy^{2/3}}{3y^{4/3}}$$

Notice that the same results would have been obtained, but not displayed, if the command

$$my=D[m[x,y],y];$$

$$nx=D[n[x,y],x];$$

had been entered.

We first define m and n . Remember that a semi-colon placed at the end of a *Mathematica* command suppresses the output.

$my=D[m[x,y],y]$
computes $\frac{\partial m}{\partial y}$
and names the result my .

$nx=D[n[x,y],x]$
calculates $\frac{\partial n}{\partial x}$
and names the result nx .

At this point `my` and `nx` do not appear to be equal. However, once simplified, equality is verified when the result of the test equality `my1 == nx1` (note the double equals sign) is `True`. (The simplified derivatives are assigned the names `my1` and `nx1`).

```

ExactDiffEqn
In[10]:=
  my1=Simplify[my]
Out[10]=
      2/3
-1 + 8 x y
-----
      2/3 4/3
3 x    y

In[11]:=
  nx1=Simplify[nx]
Out[11]=
      2/3
-1 + 8 x y
-----
      2/3 4/3
3 x    y

In[12]:=
  my1==nx1
Out[12]=
  True

```

`my1=Simplify[my]`
simplifies `my`.

`nx1=Simplify[nx]`
simplifies `nx`.

`my1==nx1`
tests to see if `my1` and
`nx1` are the same. If they were not the
same, `False` would be returned instead of
`True`.

Now that the equation is known to be exact, the process of finding the function $F(x,y)$ can begin. Since by definition,

$$\frac{\partial F(x,y)}{\partial x} = M(x,y), \text{ we have } F(x,y) = \int M(x,y) \partial x + g(y).$$

where $g(y)$ is an arbitrary function of y . In order to determine $g(y)$, differentiate the above equation with respect to y

and, make use of the fact that $\frac{\partial F(x,y)}{\partial y} = N(x,y)$. Then, $\frac{\partial F(x,y)}{\partial y} = \frac{\partial}{\partial y} \int M(x,y) \partial x + g'(y) = N(x,y)$.

Therefore, $g'(y) = N(x,y) - \frac{\partial}{\partial y} \int M(x,y) \partial x$ so $g(y) = \int \left(N(x,y) - \frac{\partial}{\partial y} \int M(x,y) \partial x \right) dy$.

These steps are carried out with *Mathematica* in the following manner.

First, integrate $M(x,y)$ with respect to x . The resulting expression is called \mathbf{f} . Of course, *Mathematica* does not indicate the presence of the arbitrary function of y , $g(y)$. However, the possibility that a nonconstant function $g(y)$ exists must be investigated. This is accomplished by differentiating \mathbf{f} with respect to y , naming it \mathbf{fy} , and comparing \mathbf{fy} to $N(x,y)$. Since the difference of $N(x,y)$ and \mathbf{fy} is $g'(y)$, their difference is computed below and named \mathbf{gprime} . When simplified, \mathbf{gprime} is found to be zero. Therefore, $g(y)$ is a constant function, so the solution of this exact differential equation is $\mathbf{f} = C$. The constant C is found by evaluating \mathbf{f} at the point (1,8). This is done with the command: `$\mathbf{f} /. \mathbf{x} \rightarrow 1 /. \mathbf{y} \rightarrow 8$`

This shows that $C = \frac{27}{2}$. Therefore, the solution to the initial value problem is: $\frac{3x^{1/3}}{y^{1/3}} + 6x^{4/3}y^{1/3} = \frac{27}{2}$.

ExactDiffEqn	
<code>In[2]:=</code>	Don't forget to load the package <code>IntegralTables.m</code> before attempting to compute indefinite integrals.
<code><<IntegralTables.m</code> <code>f=Integrate[m[x,y],x]</code>	
<code>Out[2]=</code>	<code>f=Integrate[m[x,y],x]</code> computes $\int m(x,y)dx$ and names the result <code>f</code> .
$\frac{3x^{1/3}}{y^{1/3}} + 6x^{4/3}y^{1/3}$	
<code>In[3]:=</code> <code>fy=D[f,y]</code>	<code>fy=D[f,y]</code> computes $\frac{\partial f}{\partial y}$ and names the result <code>fy</code> .
<code>Out[3]=</code>	
$-\frac{x^{1/3}}{4/3y^{4/3}} + \frac{2x^{4/3}}{2/3y^{2/3}}$	calculating
<code>In[4]:=</code> <code>gprime=n[x,y]-fy</code>	<code>gprime=n[x,y]-fy</code> and
<code>Out[4]=</code>	<code>Simplify[gprime]</code> shows $n(x,y) = \frac{\partial f}{\partial y}$.
$\frac{x^{1/3}}{4/3y^{4/3}} - x^{1/3} + 2x^{4/3}y^{2/3} - 2x^{4/3}$	
<code>In[5]:=</code> <code>Simplify[gprime]</code>	
<code>Out[5]=</code> 0	
<code>In[6]:=</code> <code>f /. x -> 1 /. y -> 8</code>	calculates the value of <code>f</code> when $x=1$ and $y=8$.
<code>Out[6]=</code> 27 -- 2	Since $n(x,y) = \frac{\partial f}{\partial y}$, we conclude that <code>f</code> is the desired function.

Mathematica can now be used to check that the function f is correct.

Dt[f] computes the total differential of f . This is computed below and called **totalf**. The symbols **Dt[x]** and **Dt[y]** in the output represent dx and dy , respectively. Hence, the command

Together[Coefficient[totalf,Dt[x]]] gives the coefficient of dx and writes it as a single fraction. A similar command is used to obtain the coefficient of dy . Since the differential equation is exact, the coefficients of dx and dy should be the functions

$$M(x,y) = \frac{1+8xy^{2/3}}{x^{2/3}y^{1/3}} \quad \text{and} \quad N(x,y) = \frac{2x^{4/3}y^{2/3}-x^{1/3}}{y^{4/3}}$$

from the differential equation. The results below verify the solution.

The screenshot shows the Mathematica interface for the **ExactDiffEq** window. It displays three input-output pairs:

Input 15: `totalf=Dt[f]`
Output 15:

$$\frac{Dt[x]}{x^{2/3}y^{1/3}} + 8xy^{1/3}Dt[x] - \frac{1}{x^{1/3}y^{4/3}}Dt[y] + \frac{2x^{4/3}}{y^{2/3}}Dt[y]$$

Input 16: `Together[Coefficient[totalf,Dt[x]]]`
Output 16:

$$\frac{1+8xy^{2/3}}{x^{2/3}y^{1/3}}$$

Input 17: `Together[Coefficient[totalf,Dt[y]]]`
Output 17:

$$\frac{-x^{1/3}+2x^{4/3}y^{2/3}}{y^{4/3}}$$

Annotations on the right side of the window explain the commands:

- totalf=Dt[f]** computes the total differential of total names the result totalf.
- `Together[Coefficient[totalf,Dt[x]]]` computes the coefficient of Dt[x] in totalf and expresses the result as a single fraction.
- `Together[Coefficient[totalf,Dt[y]]]` computes the coefficient of Dt[y] in totalf and expresses the result as a single fraction.

□ Example:

To illustrate that the arbitrary function $g(y)$ may not be constant, consider the following problem:

$$\frac{3-y}{x^2} dx + \frac{y^2-2x}{xy^2} dy = 0, \quad y(-1) = 2$$

Begin by clearing all previously used expression names. Then, the same steps are followed as were used in the previous example. The functions $M(x,y)$ and $N(x,y)$ are entered and the equation found to be exact by verifying that

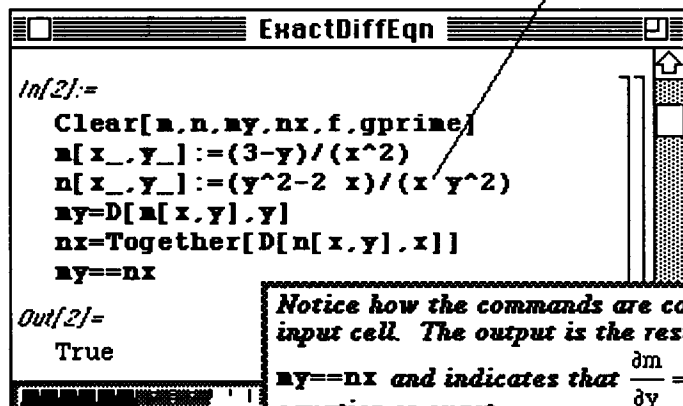
$$\frac{\partial M(x,y)}{\partial y} = \frac{\partial N(x,y)}{\partial x}.$$

Note that the partial derivative of $N(x,y)$ with respect to x must be simplified to see that it is identical to the partial of $M(x,y)$ with respect to y .

Don't forget to include the space between x and y to denote multiplication.

To verify that the differential equation

$$\frac{3-y}{x^2} dx + \frac{y^2-2x}{xy^2} dy = 0$$
is exact, we first clear all prior definitions of $m, n, my, nx, f,$ and $gprime$. We then define m and n , compute $\frac{\partial m}{\partial y}$ and $\frac{\partial n}{\partial x}$, and finally check to see if they are equal.



```

In[2]:=
Clear[m, n, my, nx, f, gprime]
m[x_, y_] := (3-y)/(x^2)
n[x_, y_] := (y^2-2 x)/(x y^2)
my=D[m[x, y], y]
nx=Together[D[n[x, y], x]]
my==nx

Out[2]=
True

```

Notice how the commands are combined into a single input cell. The output is the result of the last command $my==nx$ and indicates that $\frac{\partial m}{\partial y} = \frac{\partial n}{\partial x}$. Hence the differential equation is exact.

Next, the function $F(x,y)$ is determined as in the previous example. Note that in this problem, the simplified form of **gprime** is nonconstant.

The screenshot shows a Mathematica notebook window titled "ExactDiffEqn" with three input-output pairs:

Input 15: `f=Integrate[m[x,y],x]`
Output 15:
$$3 - y - \left(\frac{\quad}{x}\right)$$

Input 16: `fy=D[f,y]`
Output 16:
$$1 - x$$

Input 17: `gprime=n[x,y]-fy`
Output 17:
$$-\left(\frac{\quad}{x}\right) + \frac{-2x + y^2}{xy}$$

On the right side of the notebook, there are three explanatory text blocks:

- f=Integrate[m[x,y],x]**
calculates $\int m(x,y)dx$
and names the result **f**.
- fy=D[f,y]**
calculates $\frac{\partial f}{\partial y} = \frac{\partial}{\partial y} \left(\int m(x,y)dx \right)$
and names the result **fy**.
- gprime=n[x,y]-fy**
calculates $n[x,y]-fy$
and names the result **gprime**.

Therefore, **gprime** must be integrated with respect to y to find the formula for $g(y)$, and the solution of the exact equation, **solution**, is the sum of the functions **f** and $g(y)$. Finally, to solve the initial value problem, **solution** is evaluated at the point $(-1,2)$ with the command **solution/.x->-1/.y->2** to

obtain a value of 2. Hence the solution is $-\frac{3-y}{x} + \frac{2}{y} = 2$

ExactDiffEqn

In[18]:= **simplegprime=Simplify[gprime]**

Out[18]=
-2
--
2
y

In[19]:= **g=Integrate[simplegprime,y]**

Out[19]=
2
-
y

In[20]:= **solution=f+g**

Out[20]=
 $-\left(\frac{3-y}{x}\right) + \frac{2}{y}$

In[21]:= **solution/.x->-1/.y->2**

Out[21]=
2

simplifies gprime.

*calculates $\int \frac{-2}{y^2} dy$
and names the result
g.*

*We claim that the general solution
to the differential equation is
 $f+g=C$, where C is a constant.*

*In particular, the solution that
satisfies the initial condition
 $x(-1)=2$ is $f+g=2$.*

*calculates the value of
solution when $x=-1$ and
 $y=2$.*

As was illustrated in the previous example, the solution can be verified by calculating the total differential, **totalsol**, of **solution** with **Dt[solution]**. Again, collecting the coefficients of **Dt[x]** and **Dt[y]** in **totalsol** and comparing these results to the functions **M(x,y)** and **N(x,y)** from the differential equation shows that the solution is correct.

The screenshot shows a Mathematica notebook window titled "ExactDiffEqn". It contains the following input and output cells:

In[22]:=
totalsol=Dt[solution]

Out[22]=

$$\frac{(3 - y) \text{Dt}[x]}{x^2} + \frac{\text{Dt}[y]}{x} + \frac{2 \text{Dt}[y]}{y^2}$$

In[23]:=
Together[Coefficient[totalsol ,Dt[x]]]

Out[23]=

$$\frac{3 - y}{x^2}$$

In[24]:=
Together[Coefficient[totalsol ,Dt[y]]]

Out[24]=

$$\frac{-2 x + y^2}{x y^2}$$

Annotations on the right side of the notebook describe the operations:

- totalsol=Dt[solution]** computes the total derivative of solution.
- Together[Coefficient[totalsol ,Dt[x]]]** computes the coefficient of **Dt[x]** in **totalsol** and expresses the result as a single fraction.
- Together[Coefficient[totalsol ,Dt[y]]]** computes the coefficient of **Dt[y]** in **totalsol** and expresses the result as a single fraction.

6.3 Undetermined Coefficients

Example:

Use the method of undetermined coefficients to solve the differential equation $y'' - 2y' + y = \sin(x)$.

Recall that the method of undetermined coefficients is used to solve nonhomogeneous linear ordinary differential equations. The general solution to the nonhomogeneous equation

$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \dots + a_1 y' + a_0 y = g(x)$ is $y(x) = y_h(x) + y_p(x)$, where $y_h(x)$ is the solution to the corresponding homogeneous equation $a_n y^{(n)} + a_{n-1} y^{(n-1)} + \dots + a_1 y' + a_0 y = 0$, and $y_p(x)$ is a particular solution to the nonhomogeneous equation.

The following solution is constructed with Version 1.2. Notice that Version 1.2 is unable to solve the differential equation $y'' - 2y' + y = \sin(x)$ with the built-in command **DSolve**.

- Because **DSolve** does not solve most nonhomogeneous equations (as illustrated below) the problem must be divided into two parts. First, the homogeneous equation must be solved, and then a particular solution to the nonhomogeneous equation must be found.

Since **DSolve** can be used to solve homogeneous linear ordinary differential equations with constant

coefficients of degree four or less, the homogeneous solution, $y_h(x)$, is found with

DSolve[$y''[x] - 2y'[x] + y[x] == 0, y[x], x$] and called **sol1** for later use. Also, the nonhomogeneous function, $g(x) = \sin x$, is assigned the name **exp**.

```

In[20]:=
DSolve[y''[x]-2y'[x]+y[x]==Sin[x],
y[x],x]

DSolve::NotYet:
Built-in procedures cannot solve
this differential equation.

Out[20]:=
DSolve[y[x] - 2 y'[x] + y''[x] ==
Sin[x], y[x], x]

In[21]:=
sol1=DSolve[y''[x]-2y'[x]+y[x]==0,
y[x],x]

Out[21]:=
{{y[x] -> E^x C[1] + E^-x C[2]}}

In[22]:=
exp=Sin[x]

Out[22]:=
Sin[x]

```

The Mathematica command **DSolve** is unable to solve the (non-linear) differential equation $y'' - 2y' + y = \sin(x)$.

However, to use the method of undetermined coefficients, we must solve the (linear) differential equation $y'' - 2y' + y = 0$. The Mathematica command **DSolve** is used to solve this equation and its solution is $y(x) = c_1 e^x + c_2 x e^x$.

To avoid some typing, define **exp** to be $\sin(x)$.

The annihilator form of the method of undetermined coefficients will be used to find a particular solution of the

nonhomogeneous problem. Recall that the n -th order derivative of a function y is $D^n y = \frac{d^n y}{dx^n}$.

Thus, the linear n -th order ordinary differential equation with constant coefficients

$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \dots + a_1 y' + a_0 y = g(x)$ can be expressed in operator notation

$a_n D^n y + a_{n-1} D^{n-1} y + \dots + a_1 D y + a_0 y = g(x)$ or $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0) y = g(x)$.

The expression $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0)$ is called an **n -th order differential operator**.

The differential operator $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0)$ is said to **annihilate** a function f (which possesses at least n derivatives) if $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0) f(x) = 0$.

In order to solve the nonhomogeneous equation, recall the following :

(i) The differential operator D^n annihilates each of the functions $1, x, x^2, \dots, x^{n-1}$.

(ii) The differential operator $(D - \alpha)^n$ annihilates each of the functions $e^{\alpha x}, x e^{\alpha x}, x^2 e^{\alpha x}, \dots, x^{n-1} e^{\alpha x}$.

(iii) The differential operator $[D^2 - 2\alpha D + (\alpha^2 + \beta^2)]^n$ annihilates each of the functions $e^{\alpha x} \cos \beta x, x e^{\alpha x} \cos \beta x, x^2 e^{\alpha x} \cos \beta x, \dots, x^{n-1} e^{\alpha x} \cos \beta x, e^{\alpha x} \sin \beta x, x e^{\alpha x} \sin \beta x, x^2 e^{\alpha x} \sin \beta x, \dots, x^{n-1} e^{\alpha x} \sin \beta x$.

Let $P(D)$ denote the differential operator $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0)$.

Then the nonhomogeneous linear n -th order ordinary differential equation with constant coefficients equation can be expressed as $P(D) y = g(x)$. When $g(x)$ consists of either

- (a) a constant k ,
- (b) a polynomial in x ,
- (c) an exponential function $e^{\alpha x}$,
- (d) $\sin \beta x, \cos \beta x$

or finite sums and products of these functions, another differential operator which annihilates $g(x)$ can be determined.

Suppose that the differential operator $P_1(D)$ annihilates $g(x)$. Then applying $P_1(D)$ to the nonhomogeneous equation yields $P_1(D) P(D) y = P_1(D) g(x) = 0$. The form of the particular solution is found by solving the homogeneous equation $P_1(D) P(D) y = 0$.

The function **ann** [\mathbf{q}, \mathbf{f}] which verifies that the annihilating operator, \mathbf{q} (a polynomial in \mathbf{d}), annihilates the function \mathbf{f} is defined below with a **Block**. This is accomplished by transforming the polynomial \mathbf{q} into its equivalent differential form and applying it to the function \mathbf{f} .

Block [{**var1**, **var2**, ... }, **procedure**] allows the variables to be treated locally in the procedure without affecting their value outside. Hence, each time the procedure is executed, the original value of each of the variables in the list {**var1**, **var2**, ... } is saved and restored at the end of the procedure. The commands used in the definition of **ann** [\mathbf{q}, \mathbf{f}] are explained below. **Expand** [\mathbf{q}] expands all products and powers in \mathbf{q} . This is useful when the annihilator is a product of differential operators.

This expanded form of \mathbf{q} is called **p** locally.

- o In Version 2.0, the command **Block** has been replaced by the command **Module**.

Variables [**p**] gives a list of all of the variables in **p**. This command is used to obtain the variable, **d**, used in the annihilating operator.

Exponent [**p**, **var**[[1]]] yields the maximum power for which **var**[[1]] appears in **p**. In this case, **var**[[1]] represents the variable **d**. Hence, this command determines the highest power of **d** in **p** and names it **exp**. (Since **exp** is contained in the **Block**, it should not be confused with the name of the nonhomogeneous function, $g(x) = \sin x$, which was named **exp** outside of the **Block**.) Knowing the highest power of **d** enables the differential form of the operator to be determined by, first, finding the coefficients.

c[**i**]:=Coefficient [**p**, **var**[[1]], **i**] gives the coefficient of **var**[[1]]^{**i**} in **p**. In other words, it finds the coefficient of each term in **p**. (**c**[1] is the coefficient of **d**, **c**[2] is the coefficient of **d**², etc.) Since **c**[0] is the the constant in **p**, it can be obtained by evaluating **p** when the variable **d**=0.

The final command, **Sum**[**c**[**i**] **D**[**f**, {**x**, **i**}], {**i**, 0, **exp**}], applies the annihilator to the function by substituting **f** into the differential form of the operator.

In this example, the nonhomogeneous term in the equation is the function $g(x) = \sin x$ which falls under category (iii) given above. Therefore, the annihilator is d^2+1 (since $\alpha = 0$, $\beta = 1$, and $n = 1$). This is verified with `ann[d^2+1, exp]` where $g(x) = \sin x$ was given the name `exp`.

The variables `var`, `exp`, `c`, `val`, and `p` are local to the function `ann[q, f]`.

```
In[23]:=
ann[q_, f_] :=
  Block[{var, exp, c, val, p},

    p=Expand[q];

    var=Variables[p];

    exp=Exponent[p, var[[1]]];

    c[0]=p /. var[[1]] -> 0;

    c[i_]:=Coefficient[p, var[[1]], i];

    val=Sum[c[i] D[f, {x, i}], {i, 0, exp}]

  ]

In[24]:=
ann[d^2+1, exp]
Out[24]:=
0
```

Notice that the entire definition of `ann[q, f]` is contained within `Block[...]`.

*computes $(D^2 + 1)\sin(x) = 0$.
The same result would be obtained by executing the command
`D[Sin[x], {x, 2}] + Sin[x]`.*

After the annihilator has been verified, the procedure to determine the particular solution to the nonhomogeneous differential equation may begin. The first step is to represent both the annihilator and the left-hand side of the differential equation in differential operator form. The annihilator is, therefore, represented as y^2+1 , and the differential equation as y^2-2y+1 . (The variable y is being used instead of d .) Now, applying the annihilator to the differential equation is equivalent to taking the product of these two differential operators. This is done with `Expand[(y^2+1)(y^2-2y+1)]`. Hence, a fourth-order differential operator is obtained. This operator must then be used to return to the form of a differential equation. (i.e., the fourth-order differential operator obtained must be applied to y .) The function `polytodiff` defined in the `Block` below accomplishes this task by making the

making the conversion from $(a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0)y = 0$ to the differential equation $a_n y^{(n)} + a_{n-1} y^{(n-1)} + \dots + a_1 y' + a_0 y = 0$.

It uses the same commands which appeared in `ann[q, f]` above to yield a fourth-order differential equation. This equation is solved with `DSolve` with the result named `sol2`.

Corresponds to the annihilator of $\sin(x)$:

Undetermined *Corresponds to the (linear) differential equation $y'' - 2y' + y = 0$.*

```
In[25]:=
Expand[(y^2+1)(y^2-2y+1)]
Out[25]=
1 - 2 y + 2 y^2 - 2 y^3 + y^4
```

Hence, the auxiliary equation to be solved is $y^{(4)} - 2y^{(3)} + 2y^{(2)} - 2y' + y = 0$.

```
In[26]:=
polytodiff[poly_]:=
Block[{p,var,exp,c},
p=Expand[poly];
var=Variables[p];
exp=Exponent[p,var[[1]]];
c[0]=p /. var[[1]] -> 0;
c[i_]:=Coefficient[p,var[[1]],i];
Sum[c[i] D[y[x],{x,i}],{i,0,exp}]==0
]
Out[26]=
polytodiff[(y^2+1)(y^2-2y+1)]
y[x] - 2 y'[x] + 2 y''[x] -
(3) (4)
2 y [x] + y [x] == 0
```

The function `polytodiff[poly]` converts polynomials to differential equations.

converts the polynomial $(y^2+1)(y^2-2y+1)$ to the differential equation $y^{(4)} - 2y^{(3)} + 2y^{(2)} - 2y' + y = 0$.

```
In[27]:=
polytodiff[(y^2+1)(y^2-2y+1)]
Out[27]=
y[x] - 2 y'[x] + 2 y''[x] -
(3) (4)
2 y [x] + y [x] == 0
```

solves the differential equation $y^{(4)} - 2y^{(3)} + 2y^{(2)} - 2y' + y = 0$.

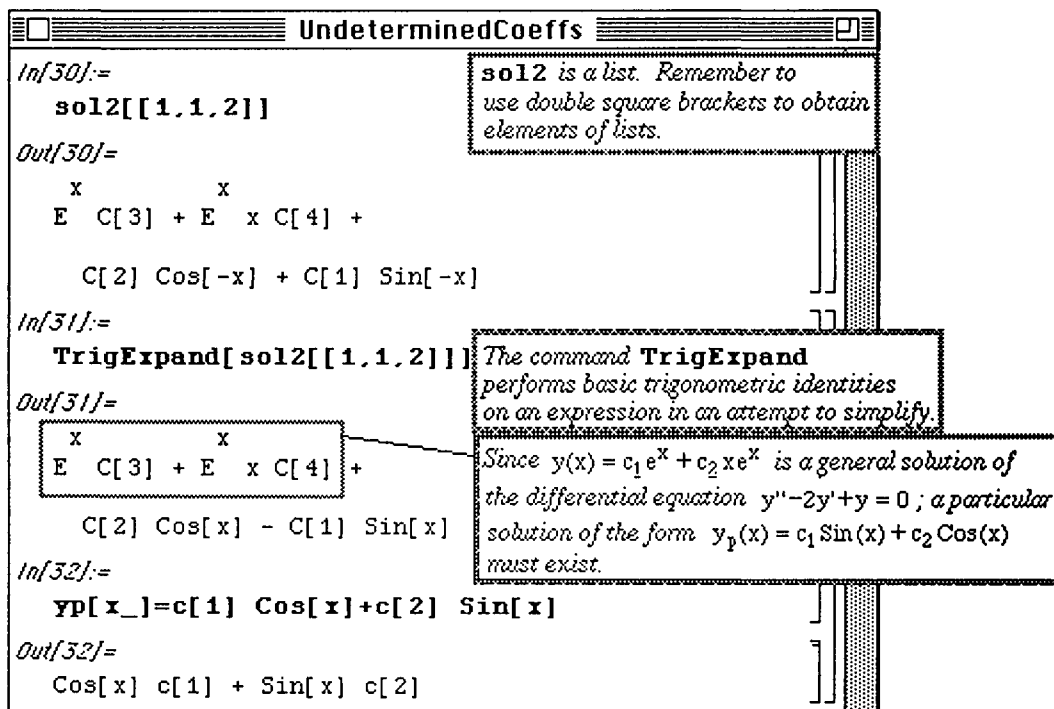
```
In[28]:=
sol2=DSolve[polytodiff[(y^2+1)(y^2-2y+1)],
y[x],x]
Out[28]=
{{y[x] ->
E^x C[3] + E^-x C[4] +
C[2] Cos[-x] + C[1] Sin[-x]}}
```

Since `sol2` is a list, the formula for $y[x]$ must be extracted. This is done with `sol2[[1,1,2]]`. Obviously, this formula can be simplified with trigonometric identities. The command, `TrigExpand[sol2[[1,1,2]]]`, does this. Recalling that the general solution to the homogeneous equation obtained earlier in the example, `sol1`, was

given by $y[x] = C[1]E^x + C[2]x E^x$, the form of the particular solution must be the remaining terms in the simplified version of `sol2[[1,1,2]]` given below. Therefore, the particular solution is defined as $y_p(x) = c_1 \text{Cos}x + c_2 \text{Sin}x$.

- o In Version 2.0, `TrigExpand` is obsolete. The command `Expand[expression, Trig->True]` in Version 2.0 performs the same calculation as `TrigExpand[expression]` in Version 1.2.

Note that the constants are arbitrary, so the negative sign associated with `C[1]` in `sol2[[1,1,2]]` as well as the order of these constants can be ignored. The objective is to solve for the arbitrary constants found in the particular solution. Hence, the name of these constants and the signs associated with them is not a concern. In order to find the values of these two constants, the particular function is defined below as the function `yp`.



```

In[30]:=
  sol2[[1,1,2]]
Out[30]=
  Ex C[3] + Ex x C[4] +
  C[2] Cos[-x] + C[1] Sin[-x]
In[31]:=
  TrigExpand[sol2[[1,1,2]]]
Out[31]=
  Ex C[3] + Ex x C[4] +
  C[2] Cos[x] - C[1] Sin[x]
In[32]:=
  yp[x_]:=c[1] Cos[x]+c[2] Sin[x]
Out[32]=
  Cos[x] c[1] + Sin[x] c[2]

```

sol2 is a list. Remember to use double square brackets to obtain elements of lists.

The command TrigExpand performs basic trigonometric identities on an expression in an attempt to simplify.

Since $y(x) = c_1 e^x + c_2 x e^x$ is a general solution of the differential equation $y'' - 2y' + y = 0$; a particular solution of the form $y_p(x) = c_1 \text{Sin}(x) + c_2 \text{Cos}(x)$ must exist.

The particular solution of a nonhomogeneous equation satisfies the nonhomogeneous equation. Hence, `yp` must satisfy $y'' - 2y' + y = \text{Sin}(x)$. `yp` is substituted into this equation below, and the expression which

results is named `eqn` so that it can be easily simplified with `Simplify[eqn]`. This simplified equation reveals that $c[1] = 1/2$ and $c[2] = 0$ by equating the coefficients of $\text{Cos}[x]$ and $\text{Sin}[x]$ on each side of the equation. Evaluating the function `yp` for these values of $c[1]$ and $c[2]$ yields the particular solution.

```
In[33]:=
  eqp=yp''[x]-2 yp'[x]+yp[x]==Sin[x]
```

```
Out[33]=
  -2 (-(Sin[x] c[1]) +
      Cos[x] c[2]) == Sin[x]
```

```
In[34]:=
  Simplify[eqp]
```

```
Out[34]=
  2 Sin[x] c[1] - 2 Cos[x] c[2] ==
  Sin[x]
```

```
In[35]:=
  Clear[yp]
  yp[x_]=c[1] Cos[x]+c[2] Sin[x] /. c[1]->1/2
      /. c[2] -> 0
```

```
Out[35]=
  Cos[x]
  -----
  2
```

Since $y_p(x) = c_1 \text{Sin}(x) + c_2 \text{Cos}(x)$ must satisfy the differential equation $y'' - 2y' + y = \text{Sin}(x)$, we substitute y by `yp` in the equation; for convenience, we name the result `eqp`.

`Simplify[eqp]` simplifies `eqp`. Equating coefficients, we see that $c[1]$ must be $1/2$ and $c[2]$ must be 0 .

After clearing `yp` we redefine `yp` to be $c[1] \text{Cos}[x] + c[2] \text{Sin}[x]$ with $c[1]$ replaced by $1/2$ and $c[2]$ replaced by 0 .

The general solution to the nonhomogeneous equation is the sum of the general solution to the homogeneous equation and the particular solution to the nonhomogeneous equation. This solution is defined below with y . To check that the solution is correct, it is substituted into the left-hand side of the differential equation and simplified to verify the results.

The screenshot shows a Mathematica notebook window titled "UndeterminedCoeffs". It contains the following input and output cells:

```

In[37]:=
  Y[x_]=sol1[[1,1,2]]+yP[x]
Out[37]=
  x      x      Cos[x]
  E C[1] + E x C[2] + -----
  2
In[40]:=
  verify=y''[x]-2y'[x]+y[x]
Out[40]=
  x      x
  2 E C[1] + 2 E C[2] +
  x
  2 E x C[2] -
  x      x
  2 (E C[1] + E C[2] +
  x      Sin[x]
  E x C[2] - -----)
  2
In[41]:=
  Simplify[verify]
Out[41]=
  Sin[x]
  
```

Annotations in the notebook:

- A callout box next to the output of In[37] states: "Hence, every solution of $y''-2y'+y = \sin(x)$ is of the form $y(x) = c_1 e^x + c_2 x e^x + \frac{1}{2} \cos(x)$."
- Text next to the input of In[40] says: "To verify, we compute and simplify."

- o The command **DSolve** has been dramatically improved in Version 2.0. In fact, Version 2.0 computes the exact solution of $y''-2y'+y=\sin(x)$. In general, when attempting to solve a differential equation, try **DSolve** first; if **DSolve** does not produce a solution, try other methods.

The screenshot shows a Mathematica notebook window titled "Version2.0DifferentialEquations". It contains the following input and output cells:

```

In[4]:=
  DSolve[y''[x]-2y'[x]+y[x]==Sin[x], y[x], x]
Out[4]=
  {{y[x] -> E^x C[1] + E^x x C[2] + Cos[x]/2}}
  
```

A callout box next to the output of In[4] states: "Version 2.0 can calculate the exact solution of the differential equation $y''-2y'+y=\sin(x)$."

■ 6.4 Linear n-th Order Differential Equations with Constant Coefficients

A differential equation of the form $a_n \frac{d^n y}{dx^n} + a_{n-1} \frac{d^{n-1} y}{dx^{n-1}} + \cdots + a_1 \frac{dy}{dx} + a_0 y = 0$, where each a_i is a real number, is called

a **linear n-th order homogeneous differential equation**.

The **characteristic equation** of the linear n-th order homogeneous differential equation

$$a_n \frac{d^n y}{dx^n} + a_{n-1} \frac{d^{n-1} y}{dx^{n-1}} + \cdots + a_1 \frac{dy}{dx} + a_0 y = 0 \text{ is } a_n m^n + a_{n-1} m^{n-1} + \cdots + a_1 m + a_0 = 0.$$

Recall that this characteristic equation is found by assuming that the solutions of the differential equation are of the

form $y(x) = e^{mx}$, and the solution is found by substituting this solution into the differential equation and solving for m . Hence, the roots of the characteristic equation are values of m which yield solutions to the differential equation. *Mathematica* is somewhat limited in its ability to solve linear n-th order homogeneous differential equations with constant coefficients directly with **DSolve** since **DSolve** only works for equations of this type of degree four or less. However, these equations can still be solved with *Mathematica* by considering the characteristic equation.

□ Example:

Mathematica's inability to solve the fifth-order equation

$$\frac{d^5y}{dx^5} - 3\frac{d^4y}{dx^4} - 5\frac{d^3y}{dx^3} + 15\frac{d^2y}{dx^2} + 4\frac{dy}{dx} - 12y = 0$$

with `DSolve` is demonstrated below by first defining the left-hand side of the equation with `lhseqn`, and then using `DSolve[lhseqn==0, y[x], x]`. Recall that the n -th derivative of $y[x]$ is given with `D[y[x], {x, n}]`. Also, since y is a function of x , $y[x]$ must be used in defining the left-hand side of the differential equation.

When `DSolve` fails to solve the equation, the characteristic equation is defined as `chareqn` so that an alternate method of solution can be employed.

```

In[18]:=
Clear[y, chareqn]
lhseqn=D[y[x], {x, 5}]-3D[y[x], {x, 4}]-5D[y[x], {x, 3}]+
15D[y[x], {x, 2}]+4D[y[x], x]-12y[x];

In[19]:=
DSolve[eqn==0, y[x], x]

DSolve::HighDegree:
Differential equation of order
higher than four encountered.

Out[19]=
DSolve[(-12 y[x] + 4 y'[x] +
(3)
15 y''[x] - 5 y'''[x] -
(4) (5)
3 y''''[x] + y'''''[x] == 0) ==
0, y[x], x]

In[20]:=
chareqn=m^5-3m^4-5m^3+15m^2+4m-12==0;

```

As usual we begin by clearing all variables to be used in the problem. Then we define `lhseqn` to be the left hand side of the differential equation

$$\frac{d^5y}{dx^5} - 3\frac{d^4y}{dx^4} - 5\frac{d^3y}{dx^3} + 15\frac{d^2y}{dx^2} + 4\frac{dy}{dx} - 12y = 0$$

Unfortunately, the command is unable to solve

$$\frac{d^5y}{dx^5} - 3\frac{d^4y}{dx^4} - 5\frac{d^3y}{dx^3} + 15\frac{d^2y}{dx^2} + 4\frac{dy}{dx} - 12y = 0$$

However, in this case the characteristic equation is

$$m^5 - 3m^4 - 5m^3 + 15m^2 + 4m - 12 = 0.$$

Since the characteristic equation is of degree five, the roots can be found exactly with `Solve[chareqn, m]`.

Since five distinct roots result, the functions e^{-2x} , e^{-x} , e^x , e^{2x} , e^{3x} are five linearly independent

solutions to the fifth-order differential equation. Therefore, the general solution to the differential equation is the linear combination of these five functions. The general solution is defined below as $y[x]$. It is then substituted into the left-hand side of the differential equation. Since this expression is lengthy, a shorted version is requested with `Short[lhseqn, 3]`. This expression is simplified to show that y satisfies the homogeneous differential equation.

```
In[21]:=
Solve[chareqn, m]
```

```
Out[21]:=
{{m -> -1}, {m -> -2}, {m -> 3},
 {m -> 2}, {m -> 1}}
```

```
In[22]:=
y[x_]=c[1] Exp[-2x]+c[2]Exp[-x]+c[3]Exp[x]+
c[4]Exp[2x]+c[5]Exp[3x];
```

```
In[23]:=
Short[lhseqn, 3]
```

```
Out[23]:=Short=
-32 c[1]
----- + <<8>> -
  2 x
E

  16 c[1]
3 (----- + <<3>> +
  2 x
E

  3 x
81 E c[5])
```

```
In[24]:=
Simplify[%]
```

```
Out[24]=
0
```

In this case, Mathematica is able to exactly solve the characteristic equation. In any case, Mathematica is ALWAYS able to approximate the roots of the characteristic equation with the command `NRoots`.

Therefore the general solution of the differential equation is

$$y(x) = c_1 e^{-2x} + c_2 e^{-x} + c_3 e^x + c_4 e^{2x} + c_5 e^{3x}$$

After defining $y[x]$ to be the solution, we evaluate and simplify `lhseqn` to verify it is the general solution of the differential equation.

Remember that a semi-colon placed at the end of a command suppresses the resulting output.

If a set of initial conditions accompany the differential equation, then the constants in the general solution must be found so that the solution satisfies the differential equation and the initial conditions. Consider the following set of initial conditions:

$$y(0) = 0, y'(0) = 1, y''(0) = 0, y'''(0) = 3, y^{(4)}(0) = 0.$$

These initial conditions are entered as the list `initialconds` below. Next, a table consisting of the constants `c[1]`, `c[2]`, `c[3]`, `c[4]`, and `c[5]` is created with `Table[c[i], {i, 1, 5}]` and named `table`. Then the command `Solve[initialconds, table]` calculates the constants. The result is named `values` so that the solution of the initial value problem can be easily obtained by evaluating `y` for the values of the constants extracted from the list `values`. This is done with `y[x] /. values[[1]]`.

```

initialconds
initialconds={y[0]==0,y'[0]==1,y''[0]==0,
              y''''[0]==3,y''''''[0]==0};
table=Table[c[i],{i,1,5}];
values=Solve[initialconds,table]

      1      1
{{c[1] -> -(-), c[2] -> -(-),
      6      6

      1      1
c[3] -> -, c[4] -> -, c[5] -> 0}

}

y[x] /. values[[1]]

      x      2 x
      -1      1      E      E
----- + -----
      2 x      x      6      6
6 E      6 E

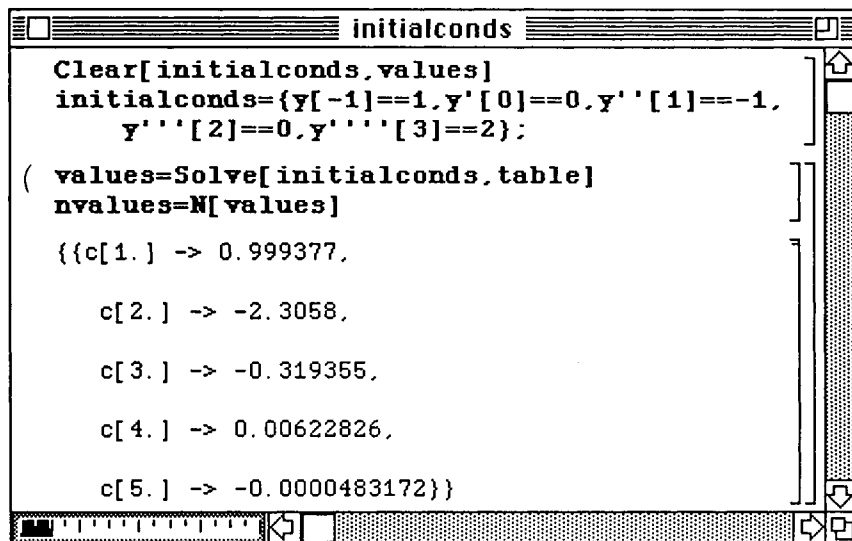
```

Since `y` is defined above, `initialconds` is a list of 5 equations with variables `c[1]`, `c[2]`, `c[3]`, `c[4]`, and `c[5]`. Consequently, `Solve[initialconds,table]` solves the system of five equations `initialconds` for `c[1]`, `c[2]`, `c[3]`, `c[4]`, and `c[5]`. In this case, the same result would be obtained with `Solve[initialconds]`.

Consider another set of conditions :

$$y(0) = 0, y'(0) = 1, y''(0) = 0, y'''(0) = 3, y^{(4)}(0) = 0.$$

After clearing all previously used definitions, the problem is solved in a similar manner as the previous example. The new set of conditions are defined in a list, and the exact values of the constants are found in **values** as they were above. However, the exact values are not as meaningful as the numerical approximation to each constant. Therefore, a numerical approximation, named **nvalues**, is obtained with **N[values]**. The solution **y[x]** can then be evaluated at **nvalues[[1]]** to yield the solution to this problem.



```

Clear[initialconds, values]
initialconds={y[-1]==1, y'[0]==0, y''[1]==-1,
  y'''[2]==0, y^{(4)}[3]==2};
( values=Solve[initialconds, table]
  nvalues=N[values]
  {{c[1.] -> 0.999377,
    c[2.] -> -2.3058,
    c[3.] -> -0.319355,
    c[4.] -> 0.00622826,
    c[5.] -> -0.0000483172}}

```

To obtain the solution, the function $y[x]$ is evaluated at the elements of `nvalues` with `y[x] /. nvalues[[1]]`.

NthOrderLinear

values

```
{c[1] ->
```

$$\frac{\begin{aligned} & ((2 E^6 (((((-2 E^5) + 7 E^6) - \\ & \quad (2 E^7 - 18 E^8 + 26 E^9) \backslash \\ & \quad - 4 E^{10} - 16 E^{11} - \\ & \quad (18 E^{12} + 54 E^{13}) - \\ & \quad (39 E^{14} - 30 E^{15} + \\ & \quad (108 E^{16}) - 126 E^{17} + \\ & \quad (108 E^{18})) / \\ & \quad ((((((((((-32 E^5) + \end{aligned}}{E^6} ((-2 E^5 + 7 E^6) - (2 E^7 - 18 E^8 + 26 E^9) \backslash - 4 E^{10} - 16 E^{11} - (18 E^{12} + 54 E^{13}) - (39 E^{14} - 30 E^{15} + (108 E^{16}) - 126 E^{17} + (108 E^{18})) / ((((((((((-32 E^5) +$$

In this case, Mathematica calculates the exact values of $c[1]$, $c[2]$, $c[3]$, $c[4]$, and $c[5]$. However, numerical approximations of the exact values are probably more useful.

6.5 The Cauchy-Euler Equation

A differential equation of the form $a_n x^n \frac{d^n y}{dx^n} + a_{n-1} x^{n-1} \frac{d^{n-1} y}{dx^{n-1}} + \dots + a_1 x \frac{dy}{dx} + a_0 y = g(x)$

is called a **Cauchy-Euler equation**.

Equations of this type are solved by assuming solutions of the form $y = x^m$, substituting this solution into the differential equation, and solving for m .

Example:

Solve the Cauchy-Euler equation $x^2 y'' - 4xy' + 6y = 0$.

Problems of this type can be solved nicely with *Mathematica* as shown below. For convenience, the left-hand side of the equation is defined and named `exp`. After defining the solution `y[x]` as x^m , re-entering `exp` causes this form of the solution (x^m) to be substituted into the left-hand side of the equation. This results in an expression involving terms which can be simplified with `Simplify[exp]`. This simplified expression is called `exp2`.

CauchyEulerEquation

```

In[1]:=
Clear[y, exp, exp2, exp3]
exp=x^2 y''[x]-4x y'[x]+6y[x]
Out[1]=
      2
6 y[x] - 4 x y'[x] + x y''[x]
In[2]:=
y[x_]=x^m
Out[2]=
m
x
In[3]:=
exp
Out[3]=
      m      m      m
6 x  - 4 m x  + (-1 + m) m x
In[4]:=
exp2=Simplify[exp]
Out[4]=
      m
(-3 + m) (-2 + m) x

```

In order to solve the Cauchy-Euler differential equation
 $x^2 y'' - 4xy' + 6y = 0$,
 we avoid typing by first defining
`exp=x^2 y''[x]-4x y'[x]+6y[x]`

*To solve the equation, assume the solution y has the form $y = x^m$;
 define*
`y[x]=x^m.`

Notice that y, y', and y'' are replaced by the correct values for
 $y = x^m$

`Simplify[exp]`
simplifies exp ; at this point we can see that the values of m that satisfy `exp=0` are $m=3$ and $m=2$.

Division of `exp2` by x^m yields the expression `exp3` which depends only on m . Therefore, equating `exp3` to 0 and solving for m , yields the solution of the differential equation. Since these roots are $m = 3$ and $m = 2$, the

functions x^2 and x^3 are both solutions. Because these functions are linearly independent, the general solution to the differential equation is $y = c_1x^3 + c_2x^2$.

This general solution is defined as $y[x]$ below. As was seen earlier, entering `exp` results in the substitution of $y[x]$ into the left-hand side of the equation. Simplification reveals that $y[x]$ is, in fact, the solution since a value of zero results.

```

CauchyEulerEquation

In[5]:=
  exp3=Simplify[exp2/x^m]
Out[5]=
  (-3 + m) (-2 + m)

In[6]:=
  Solve[exp3==0]
Out[6]=
  {{m -> 3}, {m -> 2}}

In[7]:=
  Clear[y]
  y[x_]=c[1] x^3+c[2] x^2
Out[7]=
  3      2
  x  c[1] + x  c[2]

In[8]:=
  exp
Out[8]=
  2
  x (6 x c[1] + 2 c[2]) -
  2
  4 x (3 x  c[1] + 2 x c[2]) +
  3      2
  6 (x  c[1] + x  c[2])

In[9]:=
  Simplify[exp]
Out[9]=
  0

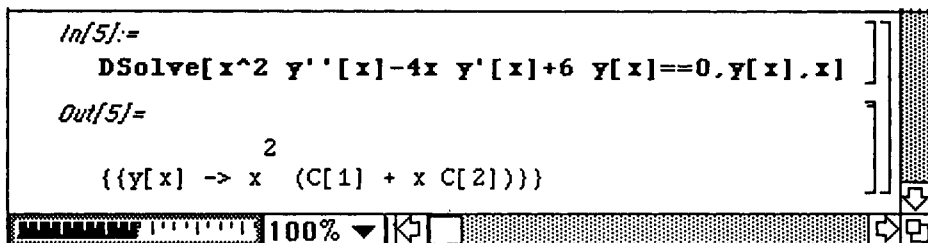
```

Nevertheless, after dividing by x^m we are able to solve the equation $(m-3)(m-2)=0$ via the command `Solve[exp3==0]`.

We conclude that the general solution of the differential equation $x^2y''-4xy'+6y=0$ is $y = c_1x^2 + c_2x^3$. To check, we redefine y and compute and simplify `exp`.

- o Even though Version 1.2 is unable to solve the Cauchy-Euler equation with the command **DSolve**, Version 2.0 can solve Cauchy-Euler equations:

```
In[5]:=
  DSolve[x^2 y''[x]-4x y'[x]+6 y[x]==0,y[x],x]
Out[5]=
  {{y[x] -> x^2 (C[1] + x C[2])}}
```



■ 6.6 Variation of Parameters

Let y_1 and y_2 be a fundamental set of solutions on the interval I of the differential equation $y'' + P(x)y' + Q(x)y = 0$, where P and Q are continuous on the interval I .

The **Wronskian** W of y_1 and y_2 is $W = \det \begin{bmatrix} y_1 & y_2 \\ y_1' & y_2' \end{bmatrix}$.

Let $u_1 = -\int \frac{y_2 f(x)}{W} dx$, $u_2 = \int \frac{y_1 f(x)}{W} dx$, and $y_p = u_1 y_1 + u_2 y_2$. Then the general solution of the differential equation $y'' + P(x)y' + Q(x)y = f(x)$ is $y = y_c + y_p$ where $y_c = c_1 y_1 + c_2 y_2$ is the general solution of the differential equation $y'' + P(x)y' + Q(x)y = 0$.

Because the method of variation of parameters depends on integration, *Mathematica* can be of great service in solving second-order linear nonhomogeneous ordinary differential equations by this method. Consider the following problem:

□ Example:

Solve $y'' - \frac{4}{x}y' + \frac{6}{x^2}y = \frac{1}{x^4}$ by the method of variation of parameters

DSolve does not solve this equation, as shown below, since it is nonhomogeneous and involves variable coefficients. Once again, the problem must be solved in two parts. The homogeneous equation must be solved and a particular solution must be determined through variation of parameters.

The homogeneous equation is $y'' - \frac{4}{x}y' + \frac{6}{x^2}y = 0$. Multiplication by x^2 yields $x^2 y'' - 4x y' + 6y = 0$.

This equation is the same as that solved the previous section on Cauchy-Euler equations. Hence, the solution to the homogeneous equation (the complimentary solution) has been found to be

$y_c(x) = c_1x^2 + c_2x^3$. This function is defined below as yc .

VariationofParameters

```
In[91]:=
Clear[x,y,yc,wronskian,w,y1,y2,f,yp,u1,u2]
In[92]:=
DSolve[y'[x]-4/x y'[x]+6/x^2 y[x]==
1/x^4,y[x],x]
DSolve::NotYet:
Built-in procedures cannot solve
this differential equation.
Out[92]=
DSolve[----- +
          2      4 y'[x]
          x      x
          -4
          y'[x] == x . y[x], x]
In[93]:=
yc[x_]=c[1]x^2+c[2]x^3
Out[93]=
      2      3
x c[1] + x c[2]
```

Notice that Mathematica is unable to solve the differential equation

$$x^2 y'' - 4xy' + 6y = \frac{1}{x^2}$$
with the DSolve command.

By the previous example,

$$y_c(x) = c_1x^2 + c_2x^3$$
is a solution to

$$x^2 y'' - 4xy' + 6y = 0$$
and hence to

$$y'' - \frac{4}{x}y' + \frac{6}{x^2}y = 0.$$

The next step in the solution process is to find a particular solution to the nonhomogeneous problem. In order to do this by the method of variation of parameters, the Wronskian of the two solutions to the homogeneous equation must be determined. The Wronskian for two arbitrary functions h and k is defined below as a function **wronskian** of two variables, h and k . Since the solutions to the homogeneous equation are defined as y_1 and y_2 , the Wronskian is computed given by **wronskian**[y_1, y_2]. The Wronskian is a function of the independent variable x and is defined as $w[x]$.

Before attempting to evaluate the integrals involved in the formulas given earlier, Version 1.2 users must load the package `IntegralTables.m` as illustrated below.

```

In[94]:=
  f[x_]=1/x^4
  y1[x_]=x^2
  y2[x_]=x^3
Out[94]=
  3
  x
In[95]:=
  wronskian[h_,k_]=Det[{{h[x],k[x]},{h'[x],k'[x]}}]
Out[95]=
  -(k[x] h'[x]) + h[x] k'[x]
In[96]:=
  w[x_]=wronskian[y1,y2]
Out[96]=
  4
  x
In[97]:=
  <<IntegralTables.m
Out[97]=
  Integrator`

```

For this problem, $f(x) = \frac{1}{x^4}$, $y_1(x) = x^2$ and $y_2(x) = x^3$.

computes

$$\text{Wronskian}(y_1(x), y_2(x)) = \text{Det} \begin{bmatrix} y_1(x) & y_2(x) \\ y_1'(x) & y_2'(x) \end{bmatrix} = x^4.$$

Be sure to load the package `IntegralTables.m` before attempting to compute definite integrals.

In order to find a particular solution of the form $y_p = u_1 y_1 + u_2 y_2$, the functions u_1 and u_2 must be calculated with the following integrals $u_1 = -\int \frac{y_2 f(x)}{W} dx$ and $u_2 = \int \frac{y_1 f(x)}{W} dx$.

The functions u_1 and u_2 are determined below with `Integrate`. The general solution to the nonhomogeneous problem is then defined as the sum of the complimentary solution found earlier,

$y_c[x]$, and the particular solution, $y_p[x] = u_1[x] y_1[x] + u_2[x] y_2[x]$. The general solution is defined as $y[x]$.

VariationofParameters

In[98]:= $u_1[x] = \text{Integrate}[-y_2[x] f[x]]/w[x], x]$

Out[98]=

$$\frac{1}{4x^4}$$

In[99]:= $u_2[x] = \text{Integrate}[y_1[x] f[x]]/w[x], x]$

Out[99]=

$$-\frac{1}{5x^5}$$

In[100]:= $y_p[x] = u_1[x] y_1[x] + u_2[x] y_2[x]$

Out[100]=

$$\frac{1}{20x^2}$$

In[101]:= $y[x] = y_c[x] + y_p[x]$

Out[101]=

$$\frac{1}{20x^2} + x^2 c[1] + x^3 c[2]$$

computes

$$u_1(x) = -\int \frac{y_2(x)f(x)}{W} dx = \frac{1}{4x^4}$$

computes

$$u_2(x) = \int \frac{y_1(x)f(x)}{W} dx = \frac{-1}{5x^5}$$

computes

$$y_p(x) = u_1(x)y_1(x) + y_2u_2(x) = \frac{1}{20x^2}$$

computes

$$y(x) = y_c(x) + y_p(x) = c_1x^2 + c_2x^3 + \frac{1}{20x^2}$$

The general solution is verified by substitution into the left-hand side of the differential equation. If correct, the resulting expression is equivalent to x^{-4} . After simplification, the desired result is obtained:

VariationofParameters

In[104]:= $Y''[x] - 4/x Y'[x] + 6/x^2 Y[x]$

Out[104]=

$$\frac{3}{10 x^4} + 2 c[1] + 6 x c[2] -$$

$$\frac{4 \left(\frac{-1}{10 x^3} + 2 x c[1] + 3 x^2 c[2] \right)}{20 x^2} + \frac{x}{x^2}$$

In[105]:= **Simplify[%]**

Out[105]= $\frac{-4}{x}$

To verify that $Y[x]$ is the solution, we compute and simplify $Y''[x] - 4/x Y'[x] + 6/x^2 Y[x]$

% refers to the previous output. Consequently, **Simplify[%]** simplifies the previous, or most recent, output.

6.7 Capabilities of DSolve

Although the command `DSolve` is quite useful in Version 1.2, `DSolve` is unable to solve many standard differential equations:

```

Version1.2NotSolvable

In[3]:=
DSolve[
  x^2 y''[x]+x y'[x]+(x^2-n^2) y[x]==0,y[x],x
]

DSolve::NotYet:
  Built-in procedures cannot solve this differential
  equation.

Out[3]=
      2      2
DSolve[(-n + x ) y[x] + x y'[x] + x y''[x] == 0,
  y[x], x]

In[4]:=
DSolve[
  (1-x^2)y''[x]-2x y'[x]+
  (n^2+n-m^2/(1-x^2))y[x]==0,y[x],x
]

DSolve::NotYet:
  Built-in procedures cannot solve this differential
  equation.

Out[4]=
      2      2
DSolve[(n + n - ----) y[x] - 2 x y'[x] +
      2
      1 - x
      2
      (1 - x ) y''[x] == 0, y[x], x]

```

The command `DSolve` in Version 1.2 is unable to solve either Bessel's equation or Legendre's equation.

□ Example:

- In Version 2.0, the command `DSolve` has been improved. In fact, exact solutions can be computed for Bessel's equation, Legendre's equation, and Airy's equation:

Version2.0MoreDifferentialEquat

`In[1]:= DSolve[x^2 y''[x]+x y'[x]+(x^2-n^2)y[x]==0,y[x],x]`

`Out[1]= {{y[x] -> BesselY[Sqrt[n^2], x] C[1] + BesselJ[Sqrt[n^2], x] C[2]}}`

Version 2.0 computes the exact solution of the differential equation $x^2y''+xy'+(x^2-n^2)y=0$.

`In[9]:= n=2; m=4; lf=DSolve[(1-x^2)y''[x]-2x y'[x]+(n^2+n-m^2/(1-x^2))y[x]==0,y[x],x]`

`Out[9]= {{y[x] -> ((x C[1] / ((-1+x)^(3/2) (1+x)^(3/2) + (x (1/x + 3 x - x^3 + x^5/5 - 1/C[1]) - 3 C[1] + C[1]^3 - C[1]^5/5) C[2]) / E^(3 (Log[-1+x] + Log[1+x]))/2) / Sqrt[-1+x^2]}}`

In this example, we define $n=2$ and $m=4$. Remember that in order to suppress the output from a command, a semi-colon must be placed at the end of it. Version 2.0 computes the exact solution of $(1-x^2)y''-2xy'+(n^2-\frac{m^2}{1-x^2})y=0$.

```

In[73]:=
c1={-1,2,-3,4};
c2={0,1,-2,3};
grays=Table[GrayLevel[j/30],{j,0,15}];

graphs=Table[If[[1,1,2]]
/. C[1]->c1[[j]] /. C[2]->c2[[k]],
{j,1,4},{k,1,4}
] // Flatten;

Short[graphs,2]

```

Out[73]:=Short=

$$\left(-\frac{x}{(-1+x)^{3/2}(1+x)^{3/2}\sqrt{-1+x^2}} \right), \ll 14 \gg,$$

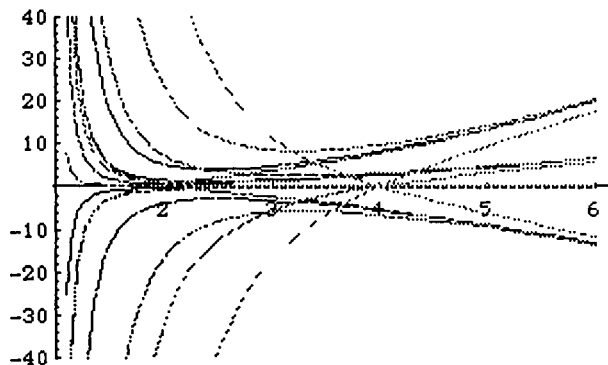
$$\frac{4x}{(-1+x)^{3/2}(1+x)^{3/2}\sqrt{-1+x^2}} + \ll 1 \gg$$

graphs is a table of solutions of Legendre's equation with $C[1]$ and $C[2]$ replaced by the values of **c1** and **c2**. The command **Flatten** must be used to remove brackets from the nested list resulting from the table command. **graphs** is a list of expressions in terms of x .

```

In[74]:=
Plot[Evaluate[graphs],{x,1,6},
PlotRange->{-40,40},PlotStyle->grays]

```



Out[74]=
-Graphics-

```

In[75]:=
airy=DSolve[y''[x]-x y[x]==0,y[x],x]

```

```

Out[75]=
{{y[x] -> AiryBi[x] C[1] + AiryAi[x] C[2]}}

```

computes the general solution of the differential equation $y''-xy=0$.

■ 6.8 Systems of Linear Differential Equations

The general form of the first-order linear system of n dimensions is $\dot{x} = A(t)x$ where $A(t) = [a_{i,j}(t)]$ is an $n \times n$ matrix with each $a_{i,j}$ a function of t and $x(t)$ is a column vector of the n dependent variables. The general form of the first-order linear system may

$$\text{be written as: } \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{1,1}(t) & a_{1,2}(t) & \cdots & a_{1,n}(t) \\ a_{2,1}(t) & a_{2,2}(t) & \cdots & a_{2,n}(t) \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1}(t) & a_{n,2}(t) & \cdots & a_{n,n}(t) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

■ Application: Spring Problems

An application of second-order linear differential equations with constant coefficients is the differential equation of the vibrations of a mass on a spring. The problem to be solved is as follows:

A coil spring is suspended from a point on a rigid support such as a ceiling or beam. A mass is then attached to the spring and allowed to come to rest in an equilibrium position. The system is then set into motion in one of two manners: (1) the mass is pulled below (or pushed above) its equilibrium and released with a zero or nonzero initial velocity at $t=0$, or (2) the mass is forced out of its equilibrium position by giving it a nonzero (downward or upward) initial velocity at $t=0$. The problem is to determine the motion of the mass on the spring which results.

By making use of Newton's second law and Hooke's Law, and by determining the forces acting

upon the mass, the differential equation for this problem is found to be $m \frac{d^2x}{dt^2} + a \frac{dx}{dt} + kx = F(t)$.

where m = the mass attached to the spring, a = the damping constant,

k = the spring constant determined with Hooke's Law, and

$F(t)$ = the function which describes any external force acting on the spring.

□ Compare the Effects of Damping

An interesting problem to consider is that of investigating the effect that different values of the damping constant, a , have on the resulting motion of the mass on the spring. Consider the following problem:

A 3-kilogram mass is attached to a spring having spring constant, $k = 12$. Determine the equation of the motion which results if the motion starts at $x(0) = 3$ with zero initial velocity if the damping constant is (i) $a = 6$ and (ii) $a = 12$. Plot the solutions obtained.

The differential equation with initial conditions for (i) is $3x'' + 6x' + 12x = 0$, $x(0) = 3$, $x'(0) = 0$.

This initial value problem is solved below using `DSolve`. The solution is called `sol` for later use. Notice that `sol` is a list of one element which is a list made up of two parts. Therefore, `sol[[1,1,2]]` is used to extract the solution from `sol`. The name `eq1` is given to this expression for convenience when plotting.

SpringApplication
⏏ ⏪ ⏩ ⏴ ⏵

```

In[7]:=
sol=DSolve[{3x''[t]+6 x'[t]+12x[t]==0,
x[0]==3,x'[0]==0},x[t],t]

Out[7]=
{{x[t] ->
((3 + Sqrt[-3])
(((-2 - 2 Sqrt[-3]) t)/2
E
) \
/ 2 + ((3 - Sqrt[-3])
(((-2 + 2 Sqrt[-3]) t)/2
E
) \
/ 2)}}

In[8]:=
eq1=sol[[1,1,2]]

Out[8]=
((3 + Sqrt[-3])
(((-2 - 2 Sqrt[-3]) t)/2
E
) / 2\
+ ((3 - Sqrt[-3])
(((-2 + 2 Sqrt[-3]) t)/2
E
) / 2

```

Remember that RETURN gives a new line. Don't forget to include the double equals sign to denote the equations.

*sol is the solution to the differential equation
 $3x''(t)+6x'(t)+12x(t)=0$ subject to the initial conditions
 $x(0)=3$ and $x'(0)=0$.*

Since sol is a list, sol[[1,1,2]] gives the solution as an expression. We name the solution eq1 so we can refer to it later when it is graphed.

⏴ ⏵
⏏ ⏪ ⏩

The differential equation for (ii) is similar to that of (i) with the exception that the damping constant, $a = 12$. Hence the coefficient of x' is 12 in this case instead of 6. The initial conditions are the same as those used in the calculations for (i). After the solution of (ii) is obtained in an identical manner as (i), the solutions to (i) and (ii) are plotted simultaneously. Note that the graph of the solution with $a = 6$ is the darker of the two curves since it has a `GrayLevel[0]` as compared to `GrayLevel[0.3]` for the problem with $a = 12$. The graph illustrates that the spring approaches its equilibrium more quickly when the damping constant is increased.

```

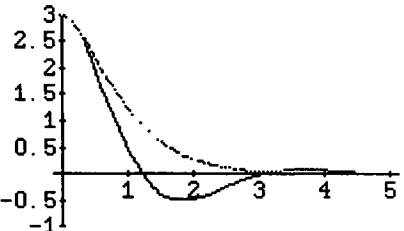
In[14]:=
sol2=DSolve[
  {3 x''[t]+12x'[t]+12 x[t]==0,
   x[0]==3,x'[0]==0},x[t],t]
Out[14]=
{{x[t] ->  $\frac{3}{2} e^{-2t} + \frac{6t}{2} e^{-2t}$ }}
In[15]:=
eq2=sol2[[1,1,2]]
Out[15]=
 $\frac{3}{2} e^{-2t} + \frac{6t}{2} e^{-2t}$ 
In[16]:=
plot2=Plot[{eq1,eq2},{t,0,5},
  PlotRange->{-1,3},PlotStyle->
  {GrayLevel[0],GrayLevel[.3]}]
Out[16]=
-Graphics-

```

solves the differential equation $3x''(t)+12x'(t)+12x(t)=0$ subject to the initial conditions $x(0)=3$ and $x'(0)=0$. The solution (which is a list) is named `sol2`.

We extract the solution from `sol2` using the command `sol2[[1,1,2]]` and name it `eq2`.

Finally, we can graph the different solutions and compare them.



□ Compare Different Initial Velocities

The effect that a change in initial velocity has on the subsequent motion is yet another problem of interest. Again, consider the problem stated above for the damping problem with $a = 6$. In this case, however, consider the problem in which the damping and initial position, $x(0) = 3$, remain unchanged while the initial velocity is varied. First, determine the motion when the initial velocity is $x'(0) = 2$ and then again when $x'(0) = -2$. Plot the solutions to compare the effects that a change in initial velocity has on the motion of the mass on the spring.

The method of solution is similar to that of the previous problem. The initial value problem which must be solved when $x'(0) = 2$ is as follows :

$$3x'' + 6x' + 12x = 0, \quad x(0) = 3, \quad x'(0) = 2.$$

This problem is solved below with **DSolve**, and the resulting expression is named **soln1** for later use. As in the previous example, the equation of the solution is extracted from **soln1** with **soln1[[1,1,2]]**. This equation is assigned the name **eq1**.

```

In[37]:=
soln1=DSolve[
  {3x''[t]+6 x'[t]+12x[t]==0,
   x[0]==3,x'[0]==2},x[t],t]
Out[37]=
{{x[t] ->
  ((9 + 5 Sqrt[-3])
    ((-2 - 2 Sqrt[-3]) t)/2
   E
    ) \
  / 6 + ((9 - 5 Sqrt[-3])
    ((-2 + 2 Sqrt[-3]) t)/2
   E
    ) \
  / 6}}

In[38]:=
eqn1=soln1[[1,1,2]]
Out[38]=
((9 + 5 Sqrt[-3])
  ((-2 - 2 Sqrt[-3]) t)/2
 E
  ) / 6\
+ ((9 - 5 Sqrt[-3])
  ((-2 + 2 Sqrt[-3]) t)/2
 E
  ) / 6

```

*solves the differential equation $3x''(t) + 6x'(t) + 12x(t) = 0$ subject to the initial conditions $x(0) = 3$ and $x'(0) = 2$. The resulting list is named **soln1**.*

*The solution is extracted from the list **soln1** with the command **soln1[[1,1,2]]** and named **eqn1**.*

The solution to the initial value problem in which $x'(0) = -2$ is found in a similar manner. The value of the initial velocity is simply changed and the same commands executed.

```

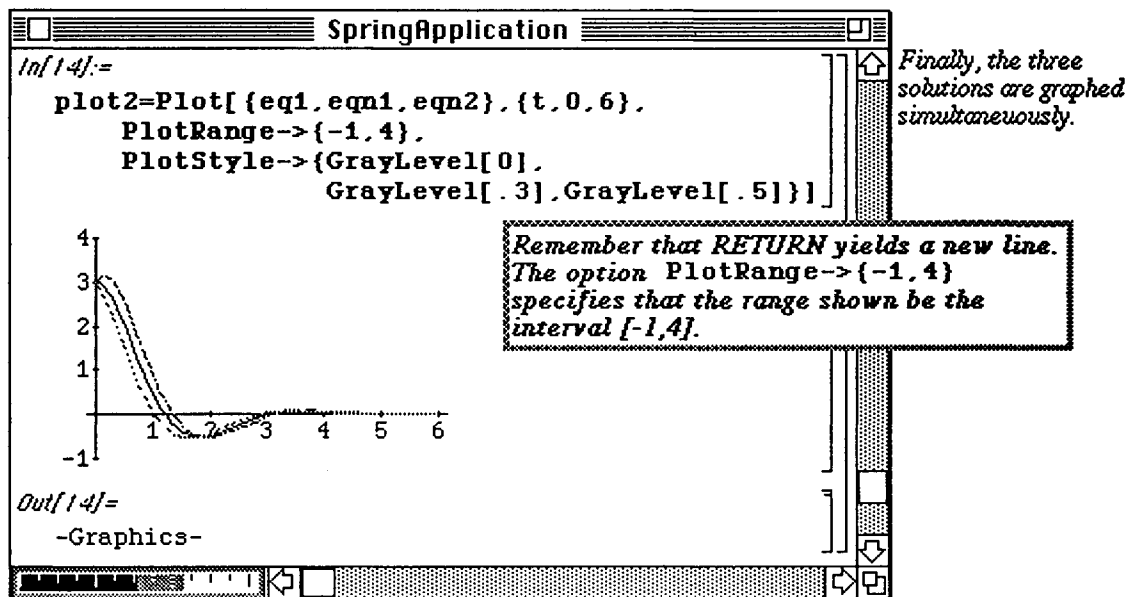
In[12]:=
soln2=DSolve[{3x''[t]+6 x'[t]+12x[t]==0,
x[0]==3,x'[0]==-2},x[t],t]
Out[12]=
{{x[t] ->
((9 + Sqrt[-3])
((-2 - 2 Sqrt[-3]) t)/2
E
) \
/ 6 + ((9 - Sqrt[-3])
((-2 + 2 Sqrt[-3]) t)/2
E
) \
/ 6}}
In[13]:=
eqn2=soln2[[1,1,2]]
Out[13]=
((9 + Sqrt[-3])
((-2 - 2 Sqrt[-3]) t)/2
E
) / 6\
+ ((9 - Sqrt[-3])
((-2 + 2 Sqrt[-3]) t)/2
E
) / 6

```

soln2 is the solution to the differential equation $3x''(t) + 6x'(t) + 12x(t) = 0$ subject to the initial conditions $x(0) = 3$ and $x'(0) = -2$.

gives the solution as an expression which can be graphed.

When both problems have been solved, the solutions can be compared by observing their graphs. These solutions are shown below with the solution to the problem in which $x'(0) = 0$ (which was solved in (i) in the previous example). Note how a positive initial velocity ($x'(0) = 2$) affects the motion as compared to an initial velocity in the negative direction. (Recall, `eq1` is the solution to the problem with zero initial velocity. It is the darkest of the three curves below.) Note also that the change in initial velocity eventually has little effect on the motion of the mass. (i.e., the three curves appear to overlap for larger values of t .)



■ Application: Classification of Equilibrium Points

The **equilibrium points** of the general first-order system

$\dot{x} = X(x, y)$, $\dot{y} = Y(x, y)$ are the points where $X(x, y) = 0$ and $Y(x, y) = 0$. If the system

$\dot{x} = X(x, y)$, $\dot{y} = Y(x, y)$ has an equilibrium point (x_0, y_0) the

linear approximation to the system in the neighborhood of the equilibrium point is defined as the system

$\dot{x} = ax + by$, $\dot{y} = cx + dy$, where $a = \frac{\partial X}{\partial x}(x_0, y_0)$, $b = \frac{\partial X}{\partial y}(x_0, y_0)$, $c = \frac{\partial Y}{\partial x}(x_0, y_0)$, and $d = \frac{\partial Y}{\partial y}(x_0, y_0)$.

The eigenvalues of the system $\dot{x} = ax + by$, $\dot{y} = cx + dy$ are the

eigenvalues of the matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$. If the eigenvalues of the

system are complex, the phase diagram in a neighborhood of the equilibrium point is an unstable spiral. If the eigenvalues are real, distinct, and have the same sign, the equilibrium point is a node; if the eigenvalues are real, distinct, and have opposite signs, the equilibrium point is a saddle. If the eigenvalues are the same, no conclusion can be drawn.

□ Example:

Classify the equilibrium points of the system $\dot{x} = -x - 5y$, $\dot{y} = x + 3y$.

First, the critical points of the system must be located. This is accomplished by solving the linear system

$$-x - 5y = 0$$

$$x + 3y = 0$$

for x and y using `Solve[-x-5y==0, x+3y==0], {x, y}`. Once the critical point (0,0) is found, it can be

classified by investigating the eigenvalues of the matrix of coefficients $\begin{bmatrix} -1 & -5 \\ 1 & 3 \end{bmatrix}$.

Recall that a matrix is represented as a list with *Mathematica* where each element of the list is a row of the matrix. Hence, the matrix of coefficients, `matrixxy`, is `{{-1, -5}, {1, 3}}`. The eigenvalues are easily determined with `Eigenvalues[matrixxy]`. The eigenvalues are complex conjugates with positive real part. Therefore, (0,0) is classified as an unstable spiral.

```

EquilibriumPoints
In[1]:=
Solve[{-x-5y==0, x+3y==0}, {x, y}]
Out[1]=
{{x -> 0, y -> 0}}
In[2]:=
matrixxy={{-1, -5}, {1, 3}}
Out[2]=
{{-1, -5}, {1, 3}}
In[3]:=
Eigenvalues[matrixxy]
Out[3]=
{1 + I, 1 - I}

```

To locate the equilibrium points, we

$$\text{solve the system } \begin{cases} -x-5y=0 \\ x+3y=0 \end{cases}$$

Thus the equilibrium point is (0,0)

The eigenvalues of the system

$$\begin{cases} \dot{x} = -x-5y \\ \dot{y} = x+3y \end{cases} \text{ are the eigenvalues} \\ \text{of the matrix } \begin{bmatrix} -1 & -5 \\ 1 & 3 \end{bmatrix}$$

Thus, we first define `matrixxy` and then we use the command `Eigenvalues[matrixxy]` to compute the eigenvalues.

Notice that we may solve the system using the command `DSolve`:

Each equation is entered separately. Remember that since x and y both depend on the variable t , they must be defined as functions of t in the two equations using square brackets. For convenience, the equations are assigned the names, `eq1` and `eq2`, respectively. Hence, the command

`DSolve[{eq1, eq2}, {x[t], y[t]}, t]`

solves the system for $x[t]$ and $y[t]$. The resulting expression is named `complexsol` for later use. By entering the command `Short[complexsol, 4]` in the same input cell, only a portion of the solution (four lines) is displayed on the screen.

In order to simplify this solution which involves complex numbers, some of the *Mathematica* commands found in the package `Trigonometry.m` must be used. This package is located in the `Algebra` folder under `Packages` and must be located by the user when loading.

Linearization

```

In[72]:=
Clear[eq1, eq2, complexsol, sol]
eq1=x'[t]==-x[t]-5y[t]

Out[72]=
x'[t] == -x[t] - 5 y[t]

In[73]:=
eq2=y'[t]==x[t]+3y[t]

Out[73]=
y'[t] == x[t] + 3 y[t]

In[74]:=
complexsol=DSolve[{eq1, eq2}, {x[t], y[t]}, t]
Short[complexsol, 4]

Out[74]//Short=
{{x[t] ->
  1      (1 - I) t
  ((- - I) E      +
  2

  1      (1 + I) t
  (- + I) E      ) C[1] +
  2

  <<2>>, y[t] -> <<1>>}}

In[75]:=
<<Trigonometry.m
  
```

In order to solve the linear system, we first define the differential equations.

Be sure to include the "double-equals" sign between the left- and right-hand side of each equation.

Use the command DSolve to solve the linear system.

Notice that eq1 and eq2 are to be solved for x[t] and y[t] in terms of t.

The solution is very long and expressed in exponential notation. The command Short[complexsol, 4] displays an abbreviated form of complexsol on approximately 4 lines.

To see that the solution to the differential equation is a real-valued function, we load the package Trigonometry.m so that we may use several commands to simplify trigonometric and exponential expressions.

The command `ComplexToTrig [complexsol]` uses Euler's formula $e^{i\theta} = \cos\theta + i \sin\theta$ to eliminate complex exponents from the solution. Again, a shortened form of the result is requested with `Short[solution, 5]`. The solution is simplified further with `Simplify[solution]`. This yields a real-valued solution. Recall that if the complex function

$\begin{bmatrix} f_1(t) + i g_1(t) \\ f_2(t) + i g_2(t) \end{bmatrix}$ is a solution of a linear system of first-order differential equations,

the real part $\begin{bmatrix} f_1(t) \\ f_2(t) \end{bmatrix}$ and the imaginary part $\begin{bmatrix} g_1(t) \\ g_2(t) \end{bmatrix}$ are linearly independent solutions of the

system. Therefore, the general solution is $\begin{bmatrix} x \\ y \end{bmatrix} = C[1] \begin{bmatrix} f_1(t) \\ f_2(t) \end{bmatrix} + C[2] \begin{bmatrix} g_1(t) \\ g_2(t) \end{bmatrix}$

Thus, **Simplify[solution]** collects the real and imaginary parts of **solution** and expresses the general solution in the appropriate manner.

```
In[76]:=
solution=ComplexToTrig[complexsol]
Short[solution,5]
```

```
Out[76]//Short=
```

```
{x[t] ->
      I t
C[2] (- E
      2
      (Cos[t] + -I Sin[t]) +
      -I t
-- E (Cos[t] + I Sin[t]))\
      2
+ C[1] <<1>>, y[t] -> <<1>>
}
}
```

```
In[77]:=
```

```
Simplify[solution]
```

```
Out[77]=
```

```
{x[t] ->
      t
E (C[1] Cos[t] -
      2 C[1] Sin[t] + C[2] Sin[t])
, y[t] ->
      t
E (C[2] Cos[t] -
      5 C[1] Sin[t] +
      2 C[2] Sin[t])}
```

After loading the package

Trigonometry.m

we use the command

ComplexToTrig

to convert the exponential form

of the solution to the trigonometric

form of the solution.

solution

is still long so the command

Short[solution,5]

is used to show an abbreviated form of

solution

on approximately five lines.

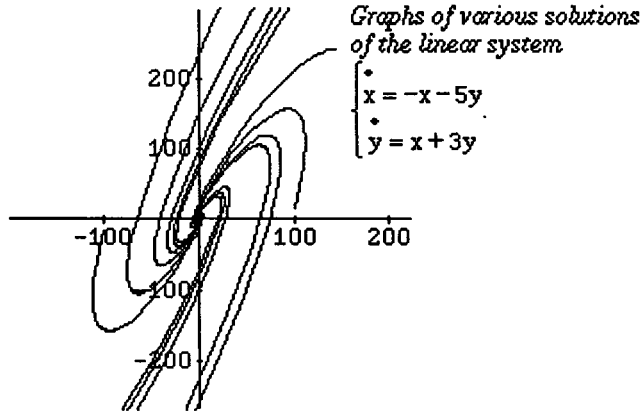
Use the command **Simplify**

to simplify the trigonometric form

of the solution. The result is expressed

in terms of real numbers.

Various values of C[1] and C[2] may be substituted into the solution and then graphed to verify that the equilibrium point is an unstable spiral.



□ **Example:**

Locate and classify the equilibrium points of the (non-linear) system $\begin{cases} \dot{x} = 2x - y^3 \\ \dot{y} = 2 - 3xy \end{cases}$.

First, **Solve** is used to find the equilibrium points of the system by setting each equation equal to zero and solving for x and y . Remember, equilibrium points are real solutions to the system, so complex values are disregarded. Since the solution set of this system was named **eqpts**, the two real points (the first and third elements in **eqpts**) can be extracted from the list with **eqpts[[1]]** and **eqpts[[3]]**. After locating the

two equilibrium points, the nonlinear system $\dot{x} = X(x,y)$, $\dot{y} = Y(x,y)$ must be linearized

about these points using the Jacobian matrix $\begin{bmatrix} \frac{\partial X(x,y)}{\partial x} & \frac{\partial X(x,y)}{\partial y} \\ \frac{\partial Y(x,y)}{\partial x} & \frac{\partial Y(x,y)}{\partial y} \end{bmatrix}$.

The easiest way to obtain this matrix is to assign names to the functions $X(x,y)$ and $Y(x,y)$. In this problem, $X(x,y) = 2x - y^3$ and $Y(x,y) = 2 - 3xy$ so the assignment of

the names x_t and y_t to these functions, respectively, allows for the Jacobian to be determined with $\{D[x_t, x], D[x_t, y]\}, \{D[y_t, x], D[y_t, y]\}$.

This matrix is named `matrix` for later use and displayed in `MatrixForm` to verify that the desired matrix has been obtained.

EquilibriumPoints

```

In[45]:=
  eqpts=Solve[{2x -y^3==0, 2-3 x y==0}, {x, y}] // N
Out[45]=
  {{x -> 0.620403, y -> 1.07457},
   {x -> -0.620403 I,
    y -> 1.07457 I},
   {x -> -0.620403, y -> -1.07457},
   {x -> 0.620403 I,
    y -> -1.07457 I}}

In[46]:=
  xt=2x -y^3
  yt=2-3 x y;
In[47]:=
  matrix={{D[xt, x], D[xt, y]}, {D[yt, x], D[yt, y]}}
  MatrixForm[matrix]
Out[47]//MatrixForm=
  2      2
  -3 y
  -3 y  -3 x
  
```

To locate the equilibrium points, we solve the system of equations

$$\begin{cases} 2x - y^3 = 0 \\ 2 - 3xy = 0 \end{cases}$$

The resulting list is named `eqpts`. Notice that the REAL solutions to the system of equations correspond to `eqpts[[1]]` and `eqpts[[3]]`.

To classify each equilibrium point, we linearize the system at each point.

Don't forget the space between the x and y to denote multiplication.

`matrix` is the matrix

$$\begin{bmatrix} \frac{\partial}{\partial x}(x_t) & \frac{\partial}{\partial y}(x_t) \\ \frac{\partial}{\partial x}(y_t) & \frac{\partial}{\partial y}(y_t) \end{bmatrix}$$

In order to linearize the nonlinear system about each equilibrium point, `matrix` must be evaluated at each point. Then, the eigenvalues for each system can be found with `Eigenvalues` to classify these equilibrium points. The matrix of coefficients for the linearized system about the first equilibrium point is easily found with `matrix/.eqpts[[1]]`.

These values are real and have opposite signs. Hence, this point is a saddle.

The linearized system about the second equilibrium point is found similarly. The eigenvalues for this system are complex with positive real part. Therefore, the second equilibrium point is classified as an unstable node.

```

In[48]:=
  matrix1=matrix /. eqpts[[1]]
  MatrixForm[matrix1]
Out[48]//MatrixForm=
  2      -3.4641
 -3.22371 -1.86121
In[49]:=
  Eigenvalues[matrix1]
Out[49]=
  {3.92873, -3.78994}
In[50]:=
  matrix3=matrix /. eqpts[[3]]
  MatrixForm[matrix3]
Out[50]//MatrixForm=
  2      -3.4641
  3.22371  1.86121
In[51]:=
  Eigenvalues[matrix3]
Out[51]=
  {1.9306 + 3.34102 I,
   1.9306 - 3.34102 I}

```

evaluates matrix for $x = .620403$ and $y = 1.07457$ and names the resulting matrix `matrix1`. `MatrixForm[matrix1]` expresses `matrix1` in traditional matrix form.

`Eigenvalues[matrix1]` computes the eigenvalues of `matrix1`.

evaluates matrix for $x = -.620403$ and $y = -1.07457$ and names the resulting matrix `matrix3`. `MatrixForm[matrix3]` expresses `matrix3` in traditional matrix form.

`Eigenvalues[matrix3]` computes the eigenvalues of `matrix3`.

6.9 Series Solutions to Ordinary Differential Equations

As was demonstrated in Chapter 3, *Mathematica* is very useful in the computation of power series solutions to ordinary differential equations.

Recall that if $x = x_0$ is an ordinary point of a differential equation, then a power series

solution of the form $\sum_{n=0}^{\infty} c_n (x - x_0)^n$ is assured.

Hence, the solution is obtained by finding the coefficient of each term in the series. Several examples are given below which illustrate how *Mathematica* is used to determine these values.

When using *Mathematica's* **Series** command, the solution is assumed to have the form

$$y = \sum_{n=0}^{\infty} \frac{y^{(n)}(x_0)}{n!} (x - x_0)^n.$$

□ Example:

Use power series to compute an approximation of the solution to the initial value problem

$$(x^3 + 1)y'' + 3\sin(x)y' = 0, \quad y(0) = 1 \text{ and } y'(0) = 2.$$

Mathematica is not able to solve this differential equation with **DSolve**, so an alternate approach must be taken. The problem is solved below using power series. When solving problems of this type by hand, the first step is to compute the appropriate derivatives of the assumed power series solution and substitute the solution and its derivatives into the differential equation. Then, the coefficients are determined by collecting like powers of x . This usually involves changing the indices in one or more terms in the equation. To better understand the method of solution given here, a brief reminder of *Mathematica's* **Series** command is given:

Series[f[x], {x, x0, n}] computes the Taylor series expansion of the function f about the point $x=x_0$ of order at most n . For example,

Series[y[x], {x, 0, 5}] yields the first 5 terms of the Taylor series expansion about $x = 0$:

$$y[0] + y'[0]x + \frac{y''[0]x^2}{2} + \frac{y^{(3)}[0]x^3}{6} + \frac{y^{(4)}[0]x^4}{24} + \frac{y^{(5)}[0]x^5}{120} + O[x]^6.$$

Therefore, the command

```
Series[(x^3+1)y''[x]+3Sin[x] y'[x],{x,0,5}]
```

accomplishes many steps at once. It computes the series expansions of $y[x]$ and $\text{Sin}[x]$, performs the necessary multiplication, and collects like terms to compute the series expansion which results from substitution into the left-hand side of the differential equation. This series is called `serapprox`.

The screenshot shows a Mathematica notebook window titled "SeriesSolutions".

Input 15:

```
DSolve[(x^3+1)y''[x]+3 Sin[x] y'[x]==0,y[x],x]
```

Output 16:

```
DSolve::NotYet:
  Built-in procedures cannot solve
  this differential equation.
```

Input 17:

```
serapprox=Series[(x^3+1)y''[x]+3Sin[x] y'[x],{x,0,5}]
```

Output 17:

$$y''[0] + (6 + y^{(3)}[0])x + \frac{(3y''[0] + \frac{y^{(4)}[0]}{2})x^2}{2} + (y''[0] + 3(-\frac{1}{3} + \frac{y^{(3)}[0]}{2}))x^3 + \dots$$

Two callout boxes provide additional information:

- The first callout box states: "Mathematica is unable to solve the differential equation $(x^3 + 1)y'' + 3\text{Sin}(x)y' = 0$ for y ."
- The second callout box states: "calculates the Taylor series about $x=0$ for $(x^3 + 1)y'' + 3\text{Sin}(x)y' = 0$."

The next step in the solution process is to equate the coefficients of **serapprox** to the corresponding coefficients of the series on the right-hand side of the equation by matching like powers of x . In this case, each coefficient on the right is 0. This is done below with **LogicalExpand**. The resulting expression is named **equations** since each term is, in fact an equation.

SeriesSolutions

In[18]:=

equations=LogicalExpand[serapprox==0]

Out[18]=

$y''[0] == 0 \ \&\& \ 6 + y^{(3)}[0] == 0 \ \&\&$

$3 y''[0] + \frac{y^{(4)}[0]}{2} == 0 \ \&\&$

$y''[0] + 3 \left(-\frac{1}{3} + \frac{y^{(3)}[0]}{2} \right) + \frac{y[0]}{6} == 0 \ \&\&$

$y[0] + 3 \left(-\frac{y''[0]}{6} + \frac{y^{(4)}[0]}{6} \right) + \frac{y^{(6)}[0]}{24}$

equates the coefficients of the power series serapprox and the power series 0. The resulting system of equations is named equations.

Represents the logical connective "and".

The initial conditions are entered so that **equations** can be solved for the unknown quantities. When **equations** is solved for

$$y''[0], y^{(3)}[0], y^{(4)}[0], \dots,$$

the solution is determined by substituting these values into the assumed solution

$$y(x) = \sum_{n=0}^{\infty} \frac{y^{(n)}[0]}{n!} x^n.$$

Notice that $y^{(7)}[0]$ is the last element in the list **values**. Hence, the solution can be approximated with the series expansion of at most order 7. **values[[1]]** extracts the appropriate list from **values**. Therefore, the approximate solution, a series of order 7, is found with **Series[y[x],{x,0,7}]/.values[[1]]**.

The expression which results from this command cannot be considered a function since it contains a remainder term. Hence, the remainder term is eliminated with **Normal[Series[y[x],{x,0,7}]/.values[[1]]**. This function is called **yapprox**.

SeriesSolutions

In[19]:=

```
y[0]=1
y'[0]=2
values=Solve[equations]
```

Out[19]=

(3)
 $\{y'[0] \rightarrow 0, y''[0] \rightarrow -6\}$

(4) (5)
 $y'''[0] \rightarrow 0, y^{(4)}[0] \rightarrow 60$

(6)
 $y^{(5)}[0] \rightarrow 144$

(7)
 $y^{(6)}[0] \rightarrow -1086\}$

In[20]:=

```
Series[y[x].{x,0,7}]/.values[[1]]
```

Out[20]=

$$1 + 2x - \frac{x^3}{2} + \frac{x^5}{5} + \frac{x^6}{840} - \frac{181x^7}{840} + O[x]^8$$

In[21]:=

```
yapprox[x_]=Normal[Series[y[x].{x,0,7}]/.values[[1]]]
```

Out[21]=

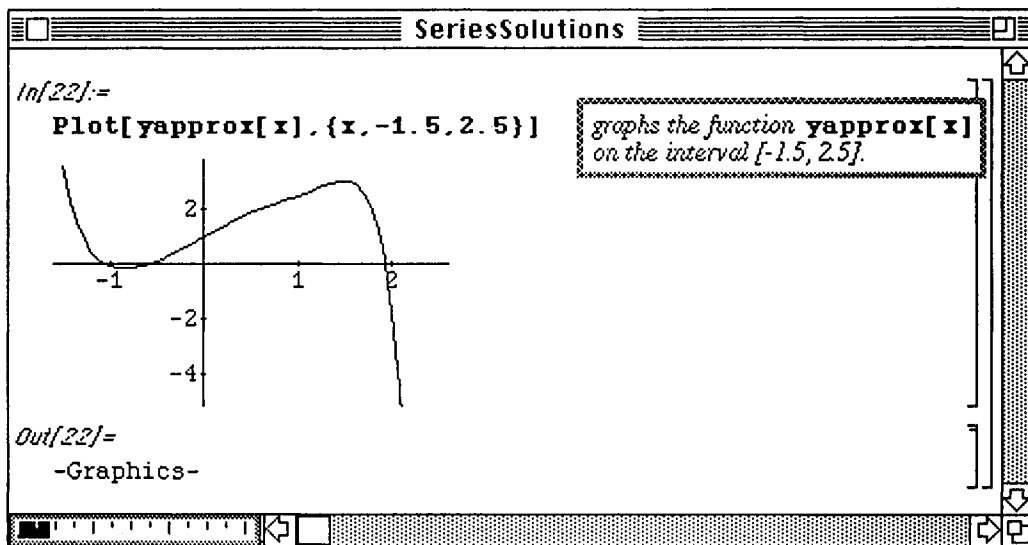
$$1 + 2x - \frac{x^3}{2} + \frac{x^5}{5} + \frac{x^6}{840} - \frac{181x^7}{840}$$

*In order to solve the above system of equations and compute an approximation of the solution, we first define $y(0)=1$ and $y'(0)=2$. The command **Solve[equations]** solves the system of equations for the unknowns. The resulting list is named **values**.*

*computes the first seven terms of the Taylor series for $y(x)$ about $x=0$ and then substitutes the values from the list **values**. Notice that the result cannot be treated as a function since the remainder term is included.*

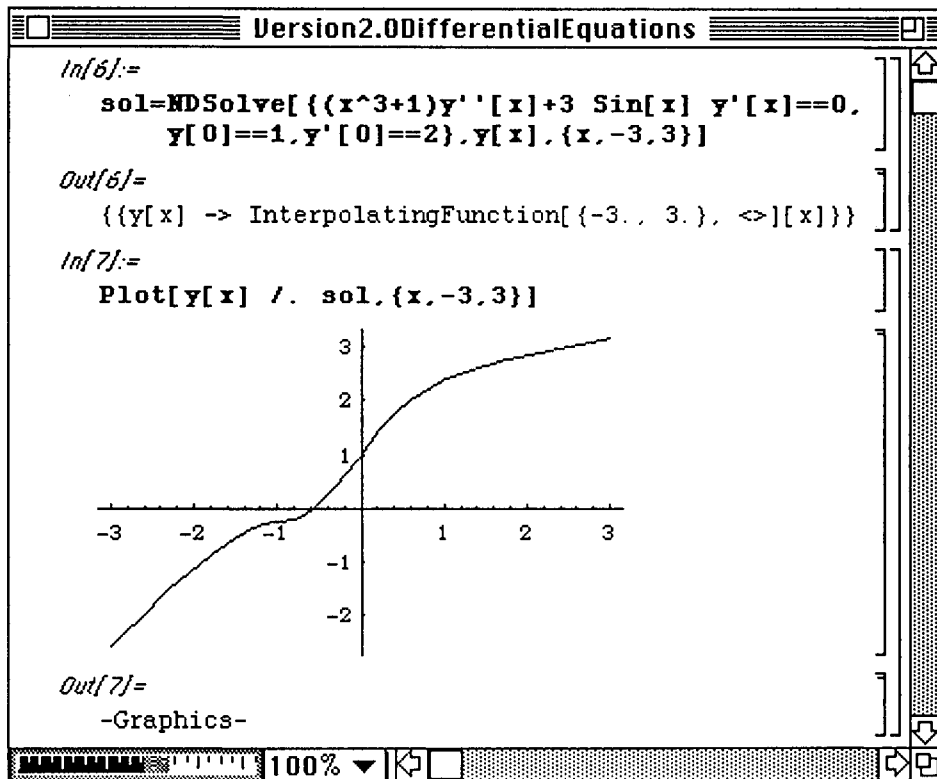
*The remainder term is removed with the command **Normal**; the resulting function is named **yapprox[x]**.*

The approximate solution can then be graphed.



- Although `DSolve` cannot be used to find an explicit solution of $(x^3 + 1)y' + 3\sin(x)y = 0$, Version 2.0 users can use the built-in command `NDSolve` to compute a numerical solution. For example, the command `sol=NDSolve[{x^3+1}y'[x]+3Sin[x]y[x]==0, y[0]==1, y'[0]==2], y[x], {x, -3, 3}]` computes a numerical solution of $(x^3 + 1)y' + 3\sin(x)y = 0$ satisfying $y(0) = 1$ and $y'(0) = 2$ on

the interval $[-3,3]$. The resulting interpolating function is named `sol` and is then graphed on the interval $[-3,3]$ with the command `Plot[y[x] /. sol, {x, -3, 3}]`.



Problems which involve arbitrary initial conditions can also be considered. With the help of *Mathematica*, approximate solutions can be found in terms of these conditions and plotted for various values. The following example illustrates this idea by using the procedures found in the previous problem.

□ Example:

Use power series to compute approximations of the solution to the initial value problems

$$y'' - 4x^2 y' - 4y = xe^x, \quad y(0) = i \text{ and } y'(0) = j, \quad i \text{ and } j \text{ both integers, } -2 \leq i \leq 2, \\ -2 \leq j \leq 2.$$

Again, this problem cannot be solved with `DSolve`. Hence, another method of solution must be used. Clearly, $x = 0$ is an ordinary point of this differential equation, so a power series solution can be assumed. The fifth-order power series expansions of the left and right-hand sides of the equation are computed with `Series[y'[x]-4x^2y'[x]-4y[x],{x,0,5}]` and `Series[x Exp[x],{x,0,5}]`, respectively. These expressions are named `ser1` and `ser2`, so the command `LogicalExpand[ser1==ser2]` equates the coefficients of like powers of x from `ser1` and `ser2`. This gives a sequence of equations which is called `equations`.

The screenshot shows a Mathematica notebook window titled "SeriesSolutions". The input and output cells are as follows:

Input 1:

```
Clear[ser1,ser2,y]
DSolve[y''[x]-4x^2 y'[x]-4y[x]==x Exp[x].y[x].x]
DSolve::NotYet:
  Built-in procedures cannot solve
  this differential equation.
```

Output 1:

```
DSolve[-4 y[x] - 4 x^2 y'[x] +
  y''[x] == E^x x. y[x], x]
```

Input 2:

```
ser1=Series[y''[x]-4 x^2 y'[x]-4y[x],{x,0,5}]
ser2=Series[x Exp[x],{x,0,5}]
equations=LogicalExpand[ser1==ser2]
```

Output 2:

```
-4 y[0] + y''[0] == 0 &&
(3)
-1 - 4 y'[0] + y''[0] == 0 &&
-1 - 4 y'[0] - 2 y''[0] +
(4)
y'''[0]
```

Three callout boxes provide additional information:

- The first callout box states: "Mathematica is unable to solve the differential equation $y'' - 4x^2 y' - 4y = xe^x$."
- The second callout box explains: "ser1 is the first five terms of the Taylor series for $y'' - 4x^2 y' - 4y$ about $x=0$. ser2 is the first five terms of the Taylor series for xe^x about $x=0$."
- The third callout box states: "LogicalExpand[ser1==ser2] equates coefficients of the two power series ser1 and ser2."

The following function, called **solutions**, is defined with a **Block** and solves **equations** to determine the coefficients of the power series solution using the initial conditions $y[0]=i$ and $y'[0]=j$. It also computes the approximate solution of order five by eliminating the remainder term. A table of functions, **conditions**, is then created for several values of i and j using **solutions**[i, j].

- o In Version 2.0, the command **Block** has been replaced by the command **Module**.

SeriesSolutions

```

In[4]:=
  solutions[i_, j_] :=
    Block[{y, values},
      y[0]=i;
      y'[0]=j;
      values=Solve[equations];
      Normal[Series[y[x], {x, 0.5}] /. values[[1]]]
    ]

In[5]:=
  conditions=Table[solutions[i, j],
    {i, -2, 2}, {j, -2, 2}];

```

solutions[i, j]
is a function that first solves the system of equations equations for the unknowns when $y(0)=i$ and $y'(0)=j$. It then computes a series approximation of $y(x)$ using the obtained values. The remainder is removed and the result is a function of x .

conditions is a table of functions of x . For each i and j , the corresponding element of **conditions** is an approximation of the solution of the differential equation
 $y'' - 4x^2 y' - 4y = xe^x$
 satisfying $y(0)=i$ and $y'(0)=j$.

Since **conditions** is a list of lists, the *Mathematica* command **Flatten** can be used to obtain a list of functions which is needed to complete the problem. A simple example is given to illustrate this command. Hence, **Flatten[conditions]** yields a list of functions called **solutionlist**. (Only a portion of this list is shown below.)

SeriesSolutions

```

In[6]:=
  Flatten[{{1,2},{3,4},{5,6}}]
Out[6]=
  {1, 2, 3, 4, 5, 6}
In[7]:=
  solutionlist=Flatten[conditions]
Out[7]=

```

conditions is actually a list of lists. To convert **conditions** from a nested list of functions to a list of functions, we use the command **Flatten**. Notice that **Flatten[{{1,2},{3,4},{5,6}}]** produces the list **{1,2,3,4,5,6}**.

converts the list of lists of functions **conditions** to a list of functions by removing curly brackets. The resulting list of functions is named **solutionlist**.

$$\begin{aligned}
 & \{-2 - 2x - 4x^2 - \frac{7x^3}{6} - \frac{23x^4}{12}, \\
 & \frac{217x^5}{120}, -2 - x - 4x^2 - \frac{x^3}{2}, \\
 & \frac{19x^4}{12} - \frac{67x^5}{40}, \\
 & -2 - 4x^2 + \frac{x^3}{6} - \frac{5x^4}{4} - \frac{37x^5}{24}, \\
 & -2 + x^2 - 4x^3 + \frac{5x^4}{6} - \frac{11x^5}{12}\}
 \end{aligned}$$

In order to plot these functions, a **GrayLevel** table is useful. So that a **GrayLevel** can be assigned to each function in the list, the length of **solutionlist** must be known. Once this length is found to be 25, a table of 25 **GrayLevel** assignments is created and called **graylist**. Hence, the list of approximate solutions can be plotted and identified by referring to the **GrayLevel** of each curve.

- o In Version 2.0, the command **Release** has been replaced by the command **Evaluate**.

SeriesSolutions

```

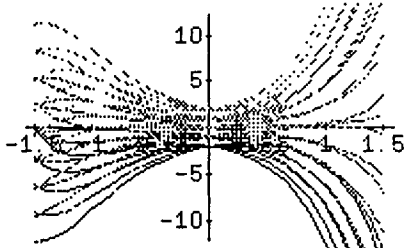
In[8]:=
  Length[solutionlist]
Out[8]=
  25
In[9]:=
  graylist=Table[
In[10]:=
    GrayLevel[i/36].{i,0,24}];
  Plot[Release[solutionlist],{x,-1.5,1.5}
    PlotStyle->graylist]

```

Length[solutionlist]
gives the number of elements in the list **solutionlist**.

gives the table
{GrayLevel[0],
GrayLevel[1/36], ,
GrayLevel[24/36]}
and names the result **graylist**.
Remember that a semi-colon placed at the end of a command suppresses the output.

Finally, we plot the table of functions **solutionlist** in shades of gray according to **graylist**. Be sure to remember to include the command **Release** when graphing tables of functions.



```

Out[10]=
  -Graphics-

```

■ 6.10 Series Solutions to Partial Differential Equations

■ Application: Two-Dimensional Wave Equation in a Circular Region

The **Bessel equation** is the differential equation $x^2y'' + xy' + (x^2 - \alpha^2)y = 0$.

If α is a positive integer, the **Bessel Function** $J_\alpha(x)$ of the first kind of order α is defined

by the series $J_\alpha(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n! (n + \alpha)!} \left(\frac{x}{2}\right)^{2n + \alpha}$.

The Bessel Functions of the first kind have the following properties:

$$(i) \quad \frac{d}{dx} \left(x^{-\mu} J_\mu(x) \right) = -x^{-\mu} J_{\mu+1}(x); \quad \text{and} \quad (ii) \quad \frac{d}{dx} \left(x^\mu J_\mu(x) \right) = x^\mu J_{\mu-1}(x).$$

The **Bessel Function** $Y_\alpha(x)$ of the second kind of order α is defined

by the integral $Y_\alpha(x) = J_\alpha(x) \int \frac{dx}{x(J_\alpha(x))^2}$.

It is well-known that the general solution of Bessel's equation, $x^2y'' + xy' + (x^2 - \alpha^2)y = 0$,

is given by $y_\alpha(x) = C_1 J_\alpha(x) + C_2 Y_\alpha(x)$.

The *Mathematica* function for the Bessel Function of the first kind of order **alpha** as a function of **x** is given by the command **BesselJ[alpha, x]**; the *Mathematica* function for the Bessel Function of the second kind of order **alpha** as a function of **x** is given by **BesselY[alpha, x]**.

□ Example:

Graph $J_0(x)$, $J_2(x)$, $J_4(x)$, $J_6(x)$ and $J_8(x)$ on the interval $[0,15]$.

This is accomplished below by creating a table of the 5 Bessel functions in `tableb` as well as a table of GrayLevel assignments in `tablec`.

○ If using Version 2.0, use `Evaluate` instead of `Release`.

The screenshot shows a Mathematica notebook window titled "BesselExample". The input field contains the following code:

```
In[7]:=
Clear[tableb, tablec]
tablec=Table[GrayLevel[i/10],{i,0,5}]
tableb=Table[BesselJ[alpha,x],{alpha,0,8,2}]

Out[7]=
{BesselJ[0,x], BesselJ[2,x],
 BesselJ[4,x], BesselJ[6,x],
 BesselJ[8,x]}
```

The input field then contains:

```
In[8]:=
Plot[Release[tableb],{x,0,15},
PlotStyle->tablec]
```

The output field shows a plot of five Bessel functions: $J_0(x)$, $J_2(x)$, $J_4(x)$, $J_6(x)$, and $J_8(x)$ on the interval $[0,15]$. The plot shows the functions oscillating with decreasing amplitude as the order increases. The y-axis ranges from -0.4 to 1.0, and the x-axis ranges from 0 to 15.

Annotations on the right side of the notebook:

- `tablec` is the table consisting of the elements `{GrayLevel[0], GrayLevel[1/10], ..., GrayLevel[5/10]}`.
- Don't forget to include the `Release` when graphing tables of functions.
- graphs the table of Bessel functions of the first kind, $J_0(x)$, $J_2(x)$, $J_4(x)$, $J_6(x)$, and $J_8(x)$, on the interval $[0,15]$. Notice that the graph of `Bessel[j,x]` is shaded `GrayLevel[j/10]`.

The output field shows `Out[8]=` followed by `-Graphics-`.

In solving numerous problems in applied mathematics, polar or cylindrical coordinate systems are often convenient to use. For example, the wave equation in a circular membrane lends itself quite naturally to the use of polar coordinates. The two-dimensional wave equation in a circular region which is radially symmetric (no dependence on θ) with boundary and initial conditions is easily expressed in polar coordinates as follows :

$$(i) \quad \frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} \right) \quad 0 < r < R, \quad 0 < t;$$

$$(ii) \quad u(R, t) = 0 \quad 0 < t;$$

$$(iii) \quad |u(0, t)| \text{ bounded, } \quad 0 < t;$$

$$(iv) \quad u(r, 0) = f(r) \quad 0 < r < R; \text{ and}$$

$$(v) \quad \frac{\partial u}{\partial t}(r, 0) = g(r) \quad 0 < r < R.$$

Using the method of separation of variables with $u(r, t) = F(r) W(t)$ leads to the two ordinary differential equations :

$$\frac{\partial^2 G}{\partial t^2} + \lambda^2 G = 0 \text{ where } \lambda = ck; \text{ and } \frac{\partial^2 W}{\partial r^2} + \frac{1}{r} \frac{dW}{dr} + k^2 W = 0 \text{ where } -k^2 \text{ is the}$$

constant of separation. $\frac{\partial^2 W}{\partial r^2} + \frac{1}{r} \frac{dW}{dr} + k^2 W = 0$ is Bessel's equation of order zero and,

thus, has solutions of the form $W(r) = C_1 J_0(kr) + C_2 Y_0(kr)$ where $J_0(kr)$ and $Y_0(kr)$ are Bessel functions of order zero of the first and second kind, respectively:

In order to determine the constants C_1 and C_2 in $W(r)$, the following graphs are necessary:

BesselExample

```

In[2]:=
plotb10=Plot[BesselJ[0, r], {r, 0, 20}]

Out[2]=
-Graphics-

In[3]:=
plotb20=Plot[BesselY[0, r], {r, 0, 20},
PlotStyle->GrayLevel[.2]]

BesselJ::branch:
BesselY[0, 0.]
has a branch point at 0..

BesselJ::branch:
BesselY[0., 0.]
has a branch point at 0..

Plot::notnum:
BesselY[0, r]
does not evaluate to a real
number at r=0..

Out[3]=
-Graphics-
```

graphs the Bessel function of the first kind, $J_0(x)$, on the interval $[0,20]$. The graph is named **plotb10**.

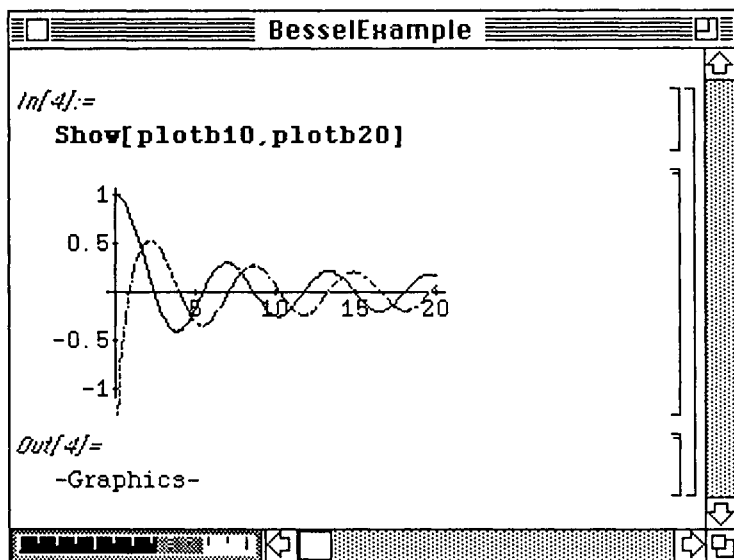
graphs the Bessel function of the second kind, $Y_0(x)$, in gray. Notice that since $Y_0(0)$ is not defined, Mathematica produces appropriate error messages. Nevertheless, the displayed graph is correct. The graph is named **plotb20**.

graphs the Bessel function of the second kind, $Y_0(x)$, in gray. Notice that since $Y_0(0)$ is not defined, Mathematica produces appropriate error messages. Nevertheless, the displayed graph is correct. The graph is named **plotb20**.

graphs the Bessel function of the second kind, $Y_0(x)$, in gray. Notice that since $Y_0(0)$ is not defined, Mathematica produces appropriate error messages. Nevertheless, the displayed graph is correct. The graph is named **plotb20**.

Notice the error messages which accompany the second plot. These, of course, are due to the fact that the Bessel function of the second kind is unbounded near $r = 0$.

These two functions can be plotted simultaneously (Note that the interval for the variable r is chosen to avoid $r = 0$. The darker function is the Bessel function of the first kind.) :



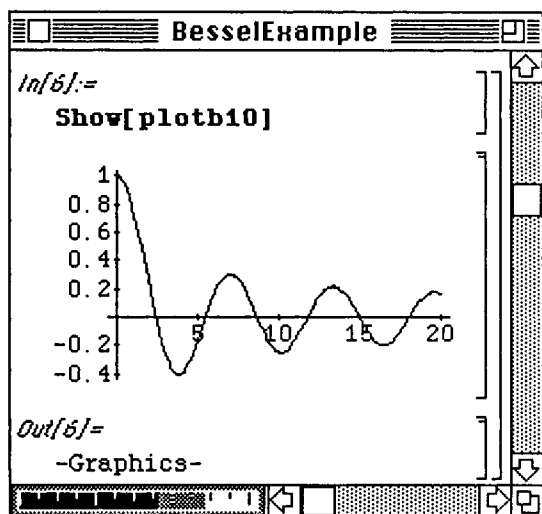
`Show[plotb10, plotb20]`
shows both graphs simultaneously.

By using the graphs above, the coefficient of the Bessel function of the second kind must be zero in the equation $W(r) = C_1 J_0(kr) + C_2 Y_0(kr)$. Otherwise, the solution is unbounded at the origin which contradicts the boundary condition. Applying the other boundary condition, $u(R, t) = 0$, leads to the equation $J_0(kR) = 0$. Hence,

$k_m = \frac{\alpha_m}{R}$ where $\alpha_m = m^{\text{th}}$ zero of $J_0(kr)$ and $m = 1, 2, 3, \dots$. These zeros of the

Bessel function can be located using *Mathematica* as opposed to simply looking them up in a table. Hopefully, some of the artificial nature of these values is alleviated by using this approach to the problem. In order to find the zeros of the Bessel function of the first kind, the *Mathematica* command

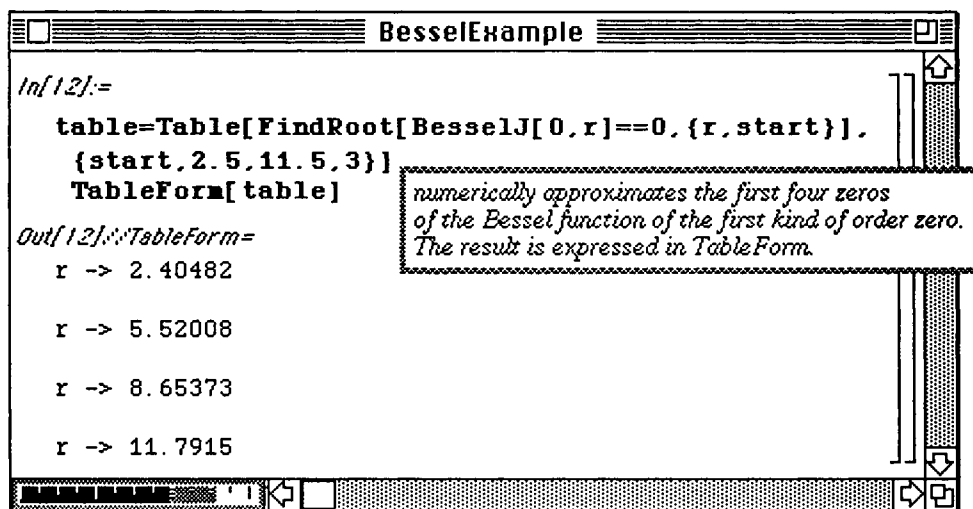
FindRoot[equation, {variable, firstguess}] is used. **FindRoot** depends on an initial guess to the root of the equation. This initial guess is obtained from the graph **plotb10**.



Show[plotb10]

redispays the graph of the Bessel function of the first kind of order zero on the interval [0,20].

Using the plot shown above, the Bessel function of the first kind appears to have its first four roots near $r = 2.5$, 5.5, 8.5, and 11.5. Therefore, the following command numerically determines approximations of the first four roots by using the command **FindRoot** with the initial guesses **start**. Note that **start** obtains values on the interval 2.5 to 11.5 using a stepsize of 3 units. Hence, the four initial guesses are the same as those given earlier, 2.5, 5.5, 8.5, and 11.5. (Notice the double equals sign which must always be used with an equation) :



These values will be used to determine the coefficients in the solution.

The solution to $\frac{\partial^2 G}{\partial t^2} + \lambda^2 G = 0$ is clearly $G_m(t) = A_m \cos(\lambda_m t) + B_m \sin(\lambda_m t)$.

Thus, the functions $u_m(r, t) = W_m(t) G_m(t) = (A_m \cos \lambda_m t + B_m \sin \lambda_m t) J_0(k_m r)$ are solutions of (i) satisfying the boundary conditions (ii)-(iii). To obtain a solution which also satisfies the initial conditions

(iv)-(v), the series $u(r, t) = \sum_{m=1}^{\infty} u_m(r, t) = \sum_{m=1}^{\infty} (A_m \cos \lambda_m t + B_m \sin \lambda_m t) J_0(k_m r)$

must be considered. In applying the initial condition (iv), the equation

$u(r, 0) = \sum_{m=1}^{\infty} A_m J_0(k_m r) = f(r)$ is obtained. Using the orthogonality properties of the Bessel

functions, the coefficients A_m and B_m

in $u(r, t) = \sum_{m=1}^{\infty} (A_m \cos \lambda_m t + B_m \sin \lambda_m t) J_0(k_m r)$ are found with the integral formula

$A_m = \frac{2}{R^2 J_1^2(\alpha_m)} \int_0^R r f(r) J_0\left(\frac{\alpha_m}{R} r\right) dr$. The coefficient B_m is found with a similar formula:

$B_m = \frac{2}{c \alpha_m R J_1^2(\alpha_m)} \int_0^R r f(r) J_0\left(\frac{\alpha_m}{R} r\right) dr$. In most cases, these two formulas are difficult

to evaluate. For a limited number of functions, integration by parts using $\frac{d}{dr} [r^{\nu} J_{\nu}(r)] = r^{\nu+1} J_{\nu}(r)$

is possible. However, even when possible, this calculation is quite tedious and lengthy. Fortunately, *Mathematica* can ease the difficulty of the computation of the coefficients through the use of **NIntegrate**.

For example, consider equations (i)-(v) with $R = 1$, $c = 2$, initial position function

$f(r) = 1 - r^2$ and initial velocity function $g(r) = 0$. Clearly, $B_m = 0$ for all m and

$A_m = \frac{2}{J_1^2(\alpha_m)} \int_0^1 r (1 - r^2) J_0(\alpha_m r) dr$. This particular integral can be evaluated exactly with

integration by parts, but the determination of the approximate values of the coefficients will be demonstrated. The following examples illustrate how the zeros of the Bessel function are extracted from the table. Similar commands will be included in the calculations which follow:

```

BesselExample
In[2]:=
Print[table[[1.1,2]]]
Print[table[[4.1,2]]]

2.40482
11.7915
    
```

`Print[table[[1.1,2]]]`
`Print[table[[4.1,2]]]`
 prints the value of `table[[1.1,2]]`
 and then prints the value of `table[[4.1,2]]`.
 This means that the value of α_1 is approximately 2.40482 and the value of α_4 is approximately 11.7915.

To approximate the first coefficient with `NIntegrate`, the following command is entered :

```

BesselExample
In[21]:=
a[1]=(2/(BesselJ[1,table[[1.1,2]]])^2)*
NIntegrate[r(1-r^2)
BesselJ[0,table[[1.1,2]]r].{r,0,1}]

Out[21]=
1.10802
    
```

computes an approximation of

$$\frac{2}{J_1^2(\alpha_1)} \int_0^1 r(1-r^2) J_0(\alpha_1 r) dr$$
 and names the result `a[1]`.

Be particularly careful that brackets and parentheses are nested correctly.

The other coefficients are easily determined as well :

BesselExample

In[22]:=

$$a[2] = (2 / (\text{BesselJ}[1, \text{table}[[2, 1, 2]]])^2) * \text{NIntegrate}[r (1-r^2) \text{BesselJ}[0, \text{table}[[2, 1, 2]] r], \{r, 0, 1\}]$$

Out[22]=
 -0.139778

In[23]:=

$$a[3] = (2 / (\text{BesselJ}[1, \text{table}[[3, 1, 2]]])^2) * \text{NIntegrate}[r (1-r^2) \text{BesselJ}[0, \text{table}[[3, 1, 2]] r], \{r, 0, 1\}]$$

Out[23]=
 0.0454765

In[24]:=

$$a[4] = (2 / (\text{BesselJ}[1, \text{table}[[4, 1, 2]]])^2) * \text{NIntegrate}[r (1-r^2) \text{BesselJ}[0, \text{table}[[4, 1, 2]] r], \{r, 0, 1\}]$$

Out[24]=
 -0.0209908

computes an approximation of

$$\frac{2}{J_1^2(\alpha_2)} \int_0^1 r (1-r^2) J_0(\alpha_2 r) dr$$

and names the result a[2].

computes an approximation of

$$\frac{2}{J_1^2(\alpha_3)} \int_0^1 r (1-r^2) J_0(\alpha_3 r) dr$$

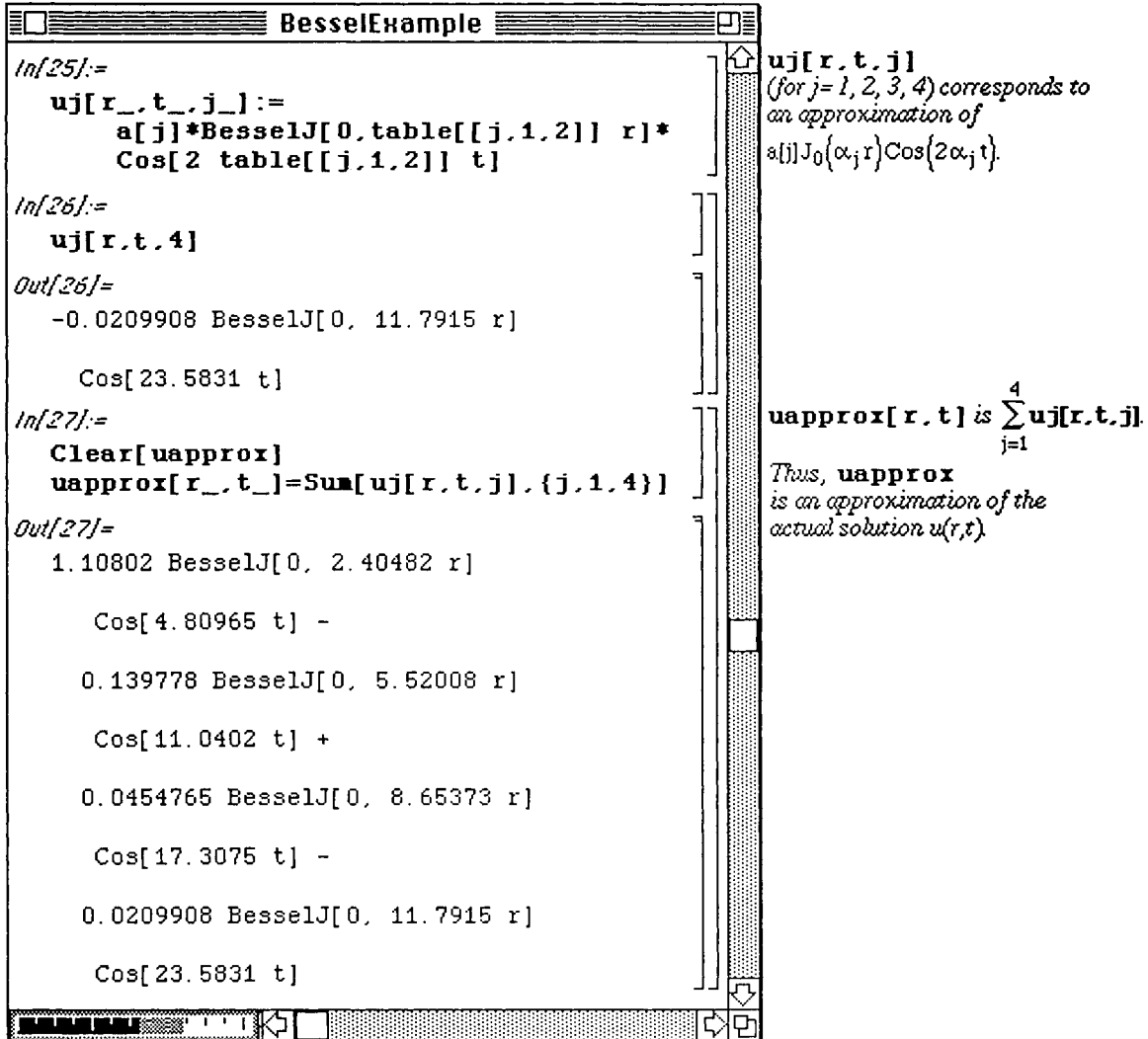
and names the result a[3].

computes an approximation of

$$\frac{2}{J_1^2(\alpha_4)} \int_0^1 r (1-r^2) J_0(\alpha_4 r) dr$$

and names the result a[4].

Hence, the first four terms of the solution are found with the function $u_j[r, t, j]$ defined below. This function is then used to compute the approximate solution by adding the first terms of the series. (Note that in this case only the first four coefficients have been calculated. Therefore, if more terms are desired, similar steps may be followed to compute more zeros of $\text{BesselJ}[0, x]$ and more series coefficients.)



```

In[25]:=
uj[r_, t_, j_] :=
  a[j]*BesselJ[0, table[[j, 1, 2]] r]*
  Cos[2 table[[j, 1, 2]] t]

In[26]:=
uj[r, t, 4]

Out[26]=
-0.0209908 BesselJ[0, 11.7915 r]

Cos[23.5831 t]

In[27]:=
Clear[uapprox]
uapprox[r_, t_] = Sum[uj[r, t, j], {j, 1, 4}]

Out[27]=
1.10802 BesselJ[0, 2.40482 r]

Cos[4.80965 t] -

0.139778 BesselJ[0, 5.52008 r]

Cos[11.0402 t] +

0.0454765 BesselJ[0, 8.65373 r]

Cos[17.3075 t] -

0.0209908 BesselJ[0, 11.7915 r]

Cos[23.5831 t]

```

$u_j[r, t, j]$
(for $j=1, 2, 3, 4$) corresponds to
an approximation of
 $a[j] J_0(\alpha_j r) \cos\{2\alpha_j t\}$.

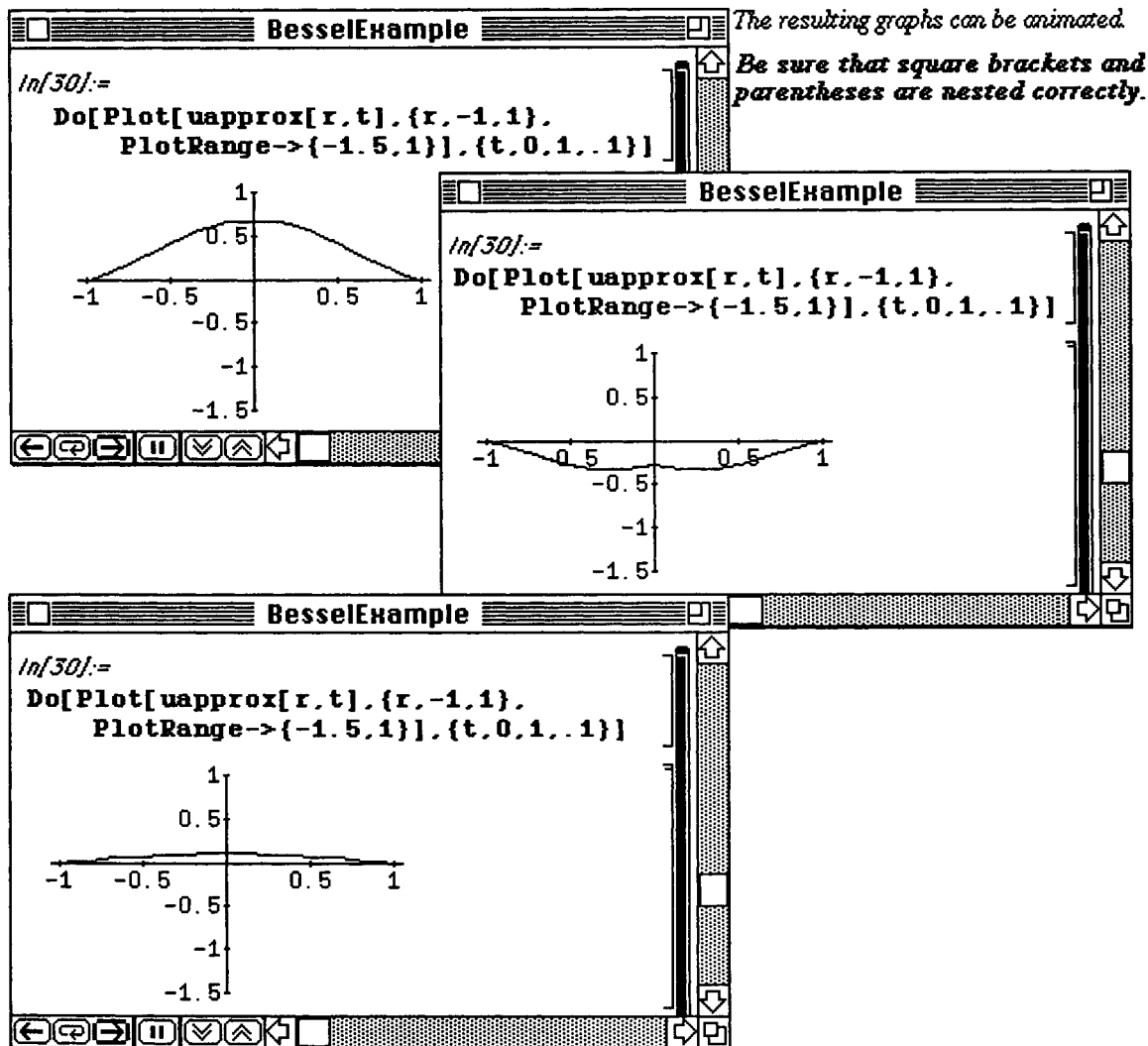
$u_{\text{approx}}[r, t]$ is $\sum_{j=1}^4 u_j[r, t, j]$.

Thus, u_{approx}
is an approximation of the
actual solution $u(r, t)$.

Solutions of this form are hard to visualize. In an attempt to bring true meaning to the "circular drumhead" problem, *Mathematica* can be used to actually see the drumhead either from a side view in two-dimensions or a full view in three.

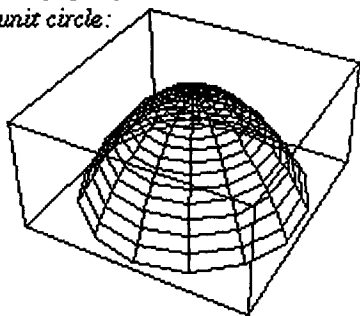
Plotting in two-dimensions is simple. The question which needs to be answered is: "What shape does the drumhead assume at a particular time t ?" This can be answered using several different methods. The easiest approach involves plotting the solution individually for various values of t . However, this involves much more time than a second method which takes advantage of a `Do` loop. Once plotted, these graphs can be animated to see the actual movement of the drumhead. The following command plots the solution for values of t between 0 and 1 using increments of 0.1. (Notice that r assumes values from $r = -1$ to $r = 1$. This seems to contradict the idea that r represents a nonnegative distance from the origin. However, $u(r, t)$ is symmetric about the y -axis.)

Several of these graphs are shown below. These graphs can then be animated.

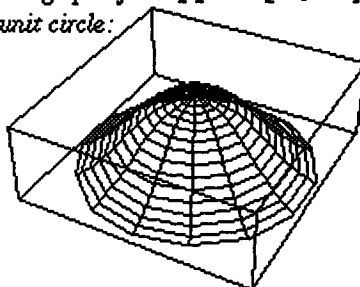


Several of the three-dimensional plots are shown below. These plots were obtained using the command **solidrev**. The command **solidrev** is discussed in the **Appendix**.

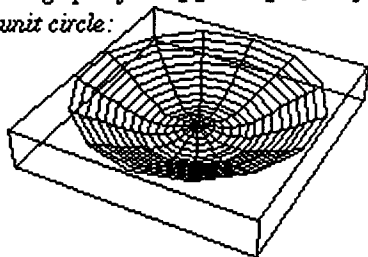
*This is the graph of **uapprox[r, 0]**
on the unit circle:*



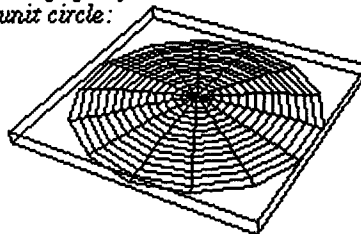
*This is the graph of **uapprox[r, . 2]**
on the unit circle:*



*This is the graph of **uapprox[r, . 4]**
on the unit circle:*



*This is the graph of **uapprox[r, 1]**
on the unit circle:*



6.11 Numerical Solutions of Differential Equations

Version 2.0 of *Mathematica* contains the command **NDSolve** which numerically solves ordinary differential equations with initial conditions. This command is particularly useful when working with nonlinear equations which **DSolve** is unable to solve. As was the case with **DSolve**, **NDSolve** can be used with single equations as well as systems. (Note that enough initial conditions must accompany the differential equation to completely solve the problem in order for **NDSolve** to be successful.)

Application: The Damped Pendulum Equation

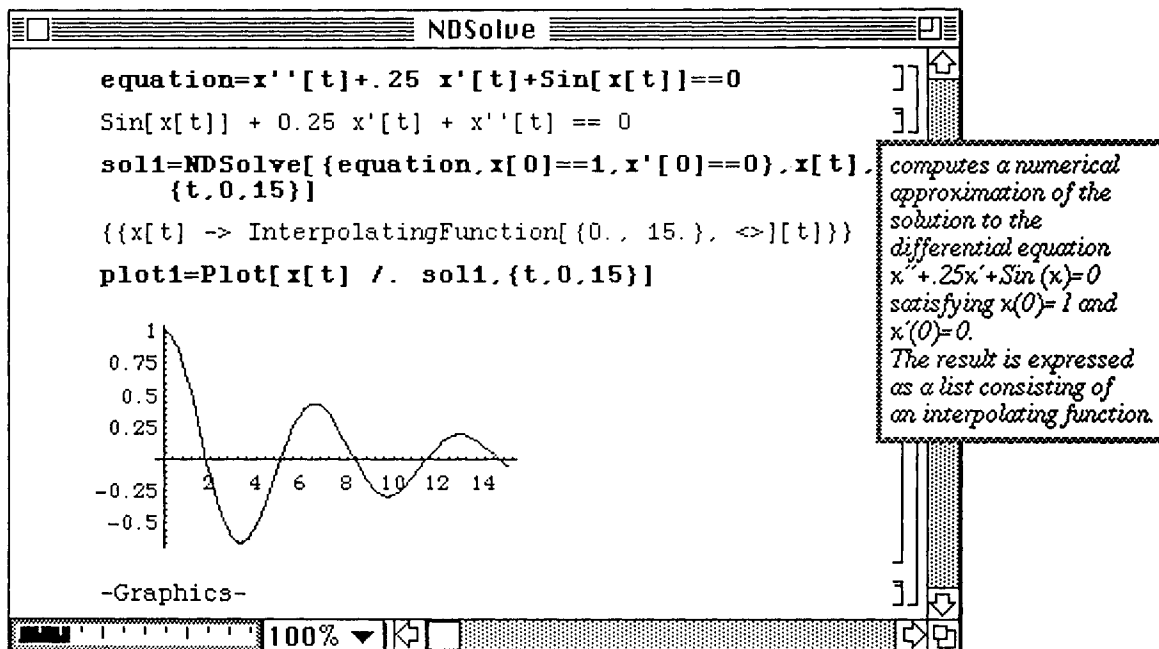
In order to illustrate **NDSolve**, consider the nonlinear pendulum equation, $x'' + .25 x' + \sin(x) = 0$ with initial conditions $x(0) = 1$, $x'(0) = 0$. The differential equation is defined below as **equation**. (Note the square brackets which must be used with the dependent variable **x[t]** in the definition.)

The syntax for **NDSolve** (to solve a second order initial value problem with dependent variable **x[t]**) as is the case here) is as follows:

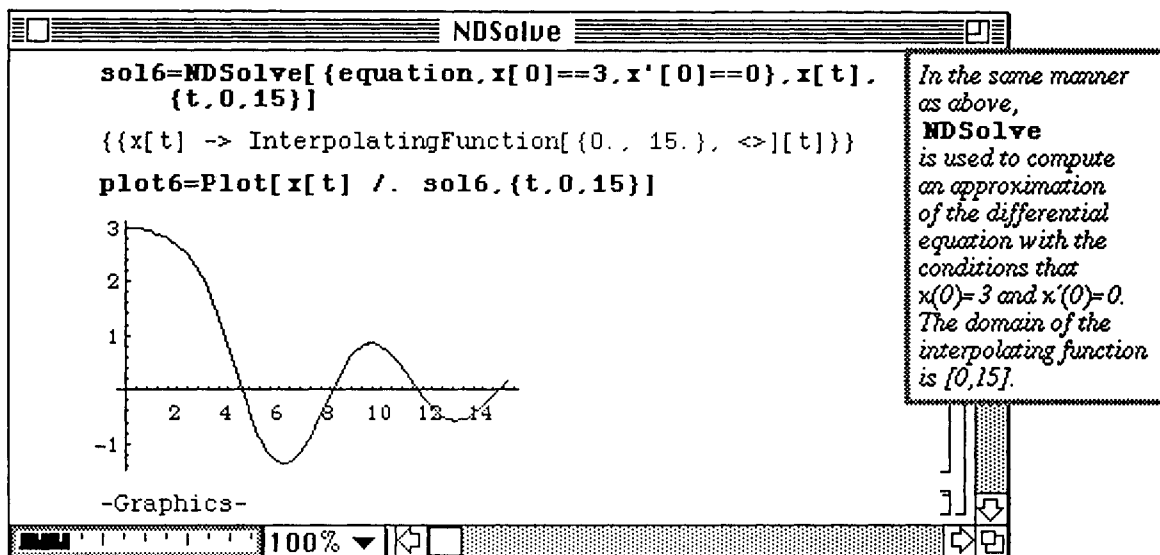
```
NDSolve [{eqn, x[t0]==c0, x' [t0]==c1}, x[t], {t, t0, t1}].
```

This finds a numerical solution to **eqn** which is valid over the interval **{t0, t1}** and satisfies the given initial conditions. Since the results are numerical, they are given in terms of the list

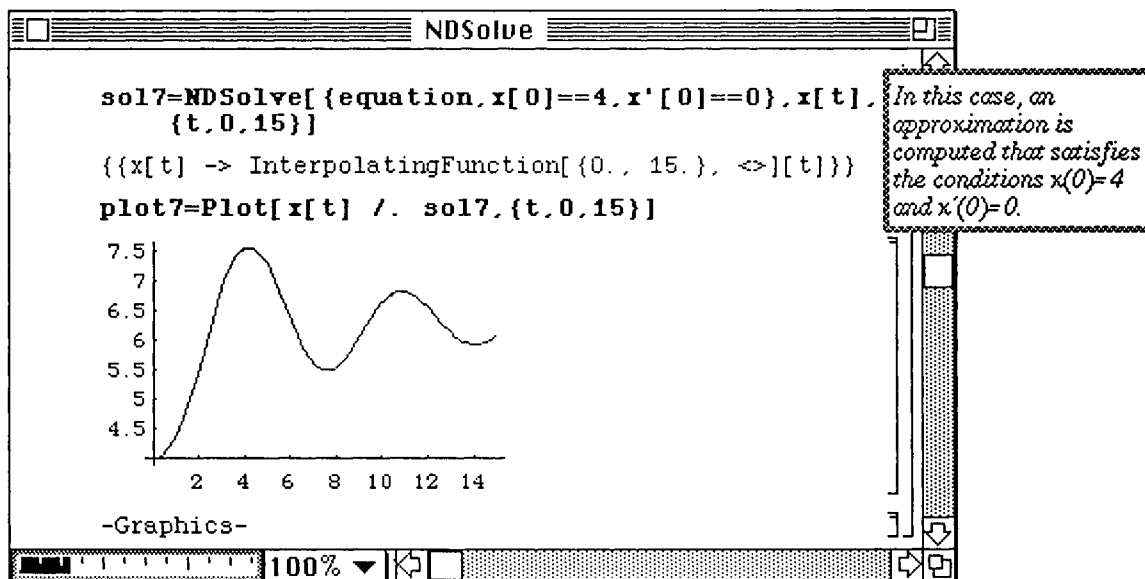
{x[t] -> InterpolatingFunction[{t0, t1}, <>][t]}. The pendulum equation given earlier is solved and plotted below. The list which results from **NDSolve** is called **sol1** while the graph is assigned the name **plot1**. (Note the manner in which the **Plot** command involving the interpolating function is stated.)



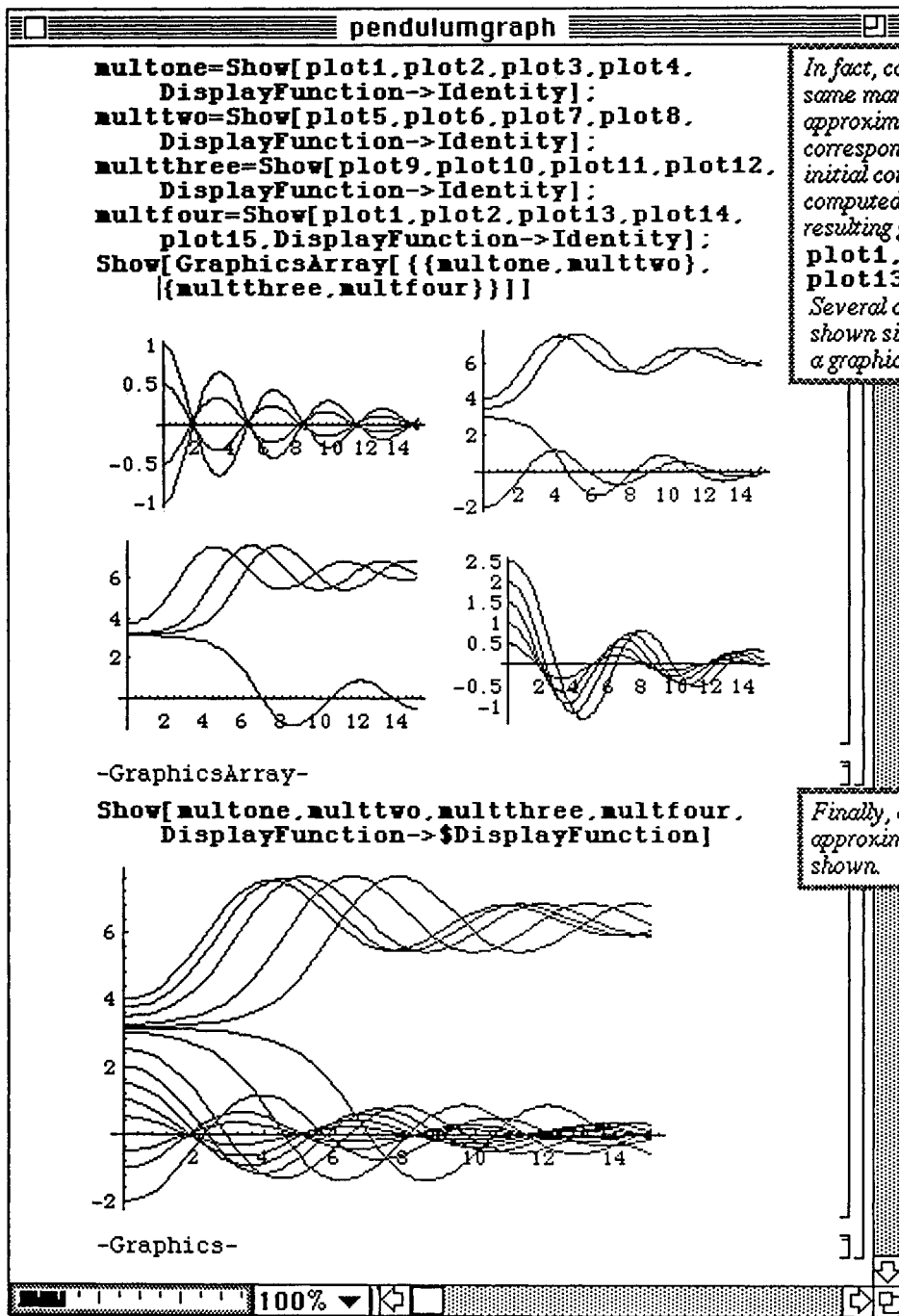
Next, consider the same equation defined in **equation** with the initial conditions $x(0) = 3$, $x'(0) = 0$. This initial value problem is solved and plotted in the same manner as the previous example. These results are named **sol6** and **plot6**, respectively.



Similarly, the initial value problem with $x(0) = 4$ and $x'(0) = 0$ is solved and plotted below in `sol7` and `plot7`. Note the effect that the nonlinear term in `equation` has on the behavior of the solution. The solutions to this equation differ considerably from solutions to the second-order linear differential equations with constant coefficients discussed earlier in *Mathematica By Example*.



Solutions to the pendulum equation for varying initial conditions are computed with `NDSolve`. The plots for eight other solutions are given in `plot5`, `plot6`, ..., `plot13`, and `plot14`. These solutions are viewed simultaneously with `Show`. First, the graphs are shown in a single graphics cell in groups of four in `multone`, `multtwo`, `multthree`, and `multfour`. In the final command, however, all fourteen graphs are shown simultaneously.



In fact, continuing in the same manner as above, approximations of solutions corresponding to a variety of initial conditions can be computed. In this case, the resulting graphs were named plot1, plot2, ..., plot13, and plot14. Several of the graphs are shown simultaneously in a graphics array.

Finally, all fourteen approximations are shown.

● 6.12 Numerical Solutions of Systems of Differential Equations

● Application: Van der Pol's Equation

As indicated earlier `NDSolve` can be used to solve systems of ordinary differential equations. Of course, there is a slight difference in the syntax from the previous case. For a first-order system of two equations with dependent variables $\mathbf{x}[t]$ and $\mathbf{y}[t]$, the correct command is as follows:

`NDSolve[{eq1, eq2, x[t0]==c0, y[t0]==c1}, {x[t], y[t]}, {t, t0, t1}]`. Again, the results are given as an interpolating function and are only valid over the interval $\{t_0, t_1\}$. Solving a system of differential equations with `NDSolve` is illustrated below with Van der Pol's equation, $x'' + e(x^2 - 1)x' + x = 0$, $x(0) = c_0$, $x'(0) = c_1$. This second-order equation can be transformed into a system of first-order equations with the substitution, $x' = y$. Hence, the following first-order system is obtained: $x' = y$, $y' = e(1 - x^2)y - x$, $x(0) = c_0$, $y(0) = c_1$.

In the steps which follow, the parameter "e" is assumed to equal 1 ($e=1$), and the equations are named `eq1` and `eq2`, respectively. (Note the square brackets which must accompany the dependent variables, $\mathbf{x}[t]$ and $\mathbf{y}[t]$.) Van der Pol's equation with initial conditions $x(0) = .25$, $y(0) = 0$ is solved in `sol` below. In this case, the solution is made up of the ordered pair $\{\mathbf{x}[t], \mathbf{y}[t]\}$. Hence, `ParametricPlot` is used to graph the numerical solutions in `plotone`.

Next, the same equation is solved using the initial conditions $x(0) = 0, y(0) = -2$. This solution is determined in the same manner as above and is plotted in `plottwo`.

The screenshot shows a Mathematica window titled "NDSolve". It contains two code blocks and their corresponding plots.

Code Block 1:

```
e=1;
eq1=x'[t]==y[t];
eq2=y'[t]==e (1-x[t]^2)y[t]-x[t];

sol=NDSolve[{eq1,eq2,x[0]==.25,y[0]==0}
  {x[t],y[t]},{t,0,10}]
plotone=ParametricPlot[Evaluate[
  {x[t],y[t]}/.sol],{t,0,10}]
```

Plot 1: A parametric plot showing a closed loop in the xy-plane. The x-axis ranges from -2 to 2, and the y-axis ranges from -2 to 2. The loop starts at (25, 0) and ends at (0, 0).

Code Block 2:

```
-Graphics-
sol=NDSolve[{eq1,eq2,x[0]==0,y[0]==-.2}
  {x[t],y[t]},{t,0,13}];
plottwo=ParametricPlot[Evaluate[
  {x[t],y[t]}/.sol],{t,0,13}]
```

Plot 2: A parametric plot showing a closed loop in the xy-plane. The x-axis ranges from -2 to 2, and the y-axis ranges from -2 to 2. The loop starts at (0, -2) and ends at (0, -2).

Code Block 3:

```
-Graphics-
```

The window also shows a status bar at the bottom with a zoom level of 100% and navigation icons.

Computes a numerical approximation of the solution of the system of differential equations

$$\begin{cases} x' = y, & y' = (1-x^2)y - x \\ x(0) = .25, & y(0) = 0 \end{cases}$$

$$0 \leq t \leq 10.$$

The resulting solution is graphed using `ParametricPlot` and named `plotone`.

In the exact same manner as above, `NDSolve` is used to compute a numerical approximation of the solution of the system of differential equations

$$\begin{cases} x' = y, & y' = (1-x^2)y - x \\ x(0) = 0, & y(0) = -2 \end{cases}$$

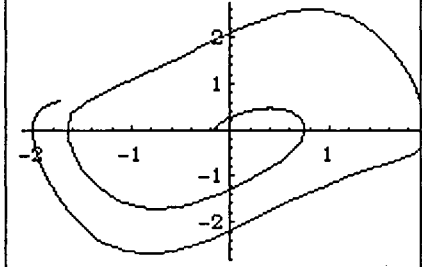
$$0 \leq t \leq 13.$$

The resulting solution is graphed using `ParametricPlot` and named `plottwo`.

plotthree and **plotfour** are given below. These correspond to solutions to Van der Pol's equation with initial conditions $x(0) = -.15, y(0) = 0$ and $x(0) = 0, y(0) = .1$, respectively.

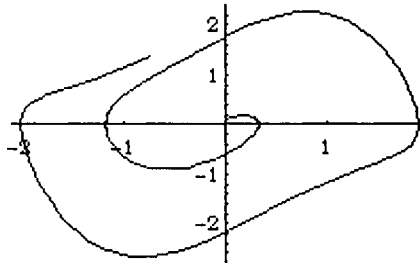
```
sol=NDSolve[{eq1,eq2,x[0]==-.15,y[0]==0},
{x[t],y[t]},{t,0,13}];
plotthree=ParametricPlot[Evaluate[
{x[t],y[t]} /. sol],{t,0,13}]
```

In the same manner as above, two other approximations are computed and graphed.



-Graphics-

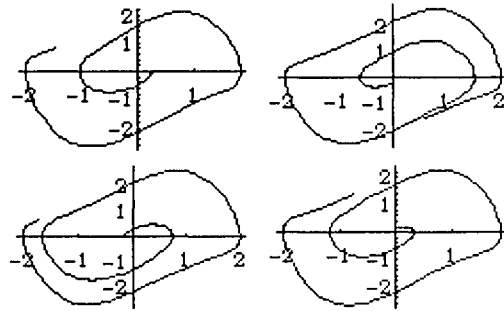
```
sol=NDSolve[{eq1,eq2,x[0]==0,y[0]==.1},
{x[t],y[t]},{t,0,13}];
plotfour=ParametricPlot[Evaluate[
{x[t],y[t]} /. sol],{t,0,13}]
```



-Graphics-

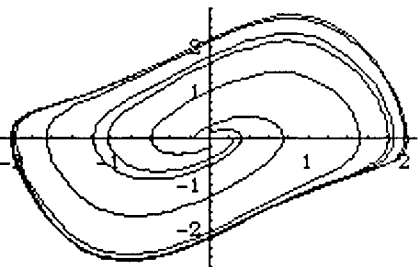
The four approximate solutions to Van der Pol's equation computed to this point with `NDSolve` are displayed in a single cell below using `GraphicsArray` within the `Show` command. This is named `setone`. The graphs are also shown simultaneously in `partone` by using the `Show` command without `GraphicsArray`.

```
setone=Show[GraphicsArray[{{plotone,plottwo},
{plotthree,plotfour}}]]
```



`-GraphicsArray-`

```
partone=Show[plotone,plottwo,
plotthree,plotfour]
```



`-Graphics-`

We can use the command `GraphicsArray` to show all four graphics in a single graphics cell.

We can use the command `Show` to show all four graphics cells simultaneously.

100%

By choosing other initial conditions, several solutions to Van der Pol's equation are computed in the same manner as before. The graphs of these approximate solutions are named `cycle1`, `cycle2`, `cycle3`, and `cycle4`. They are first shown below in the single graphics cell, called `settwo`, and are then displayed simultaneously in `parttwo`. In the final command, the eight approximate solutions to Van der Pol's equation which have been computed with `NDSolve` are displayed simultaneously in `partthree`.

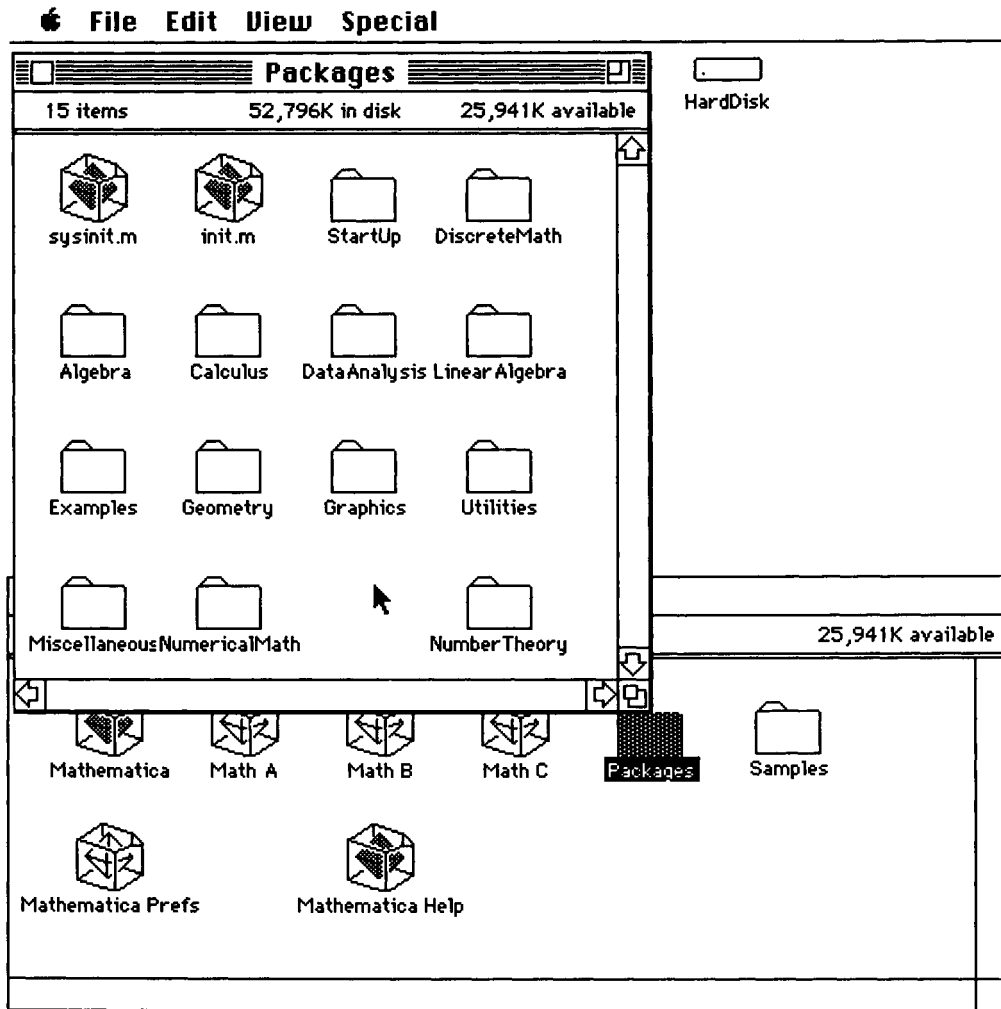
```
functiongraphs
settwo=Show[GraphicsArray[{{cycle1,cycle2},
{cycle3,cycle4}}]]
-GraphicsArray-
parttwo=Show[cycle1,cycle2,
cycle3,cycle4]
-Graphics-
partthree=Show[partone,parttwo]
-Graphics-
```

*In the same manner as above we use **NDSolve** to compute numerical approximations of solutions to Van der Pol's equation. We name the graphs **cycle1**, **cycle2**, **cycle3**, and **cycle4**.*

Finally, we show all eight graphs simultaneously.

Chapter 7 Introduction to *Mathematica* Packages

Many useful commands and functions which are not automatically performed by *Mathematica* can be found in **Packages**. The following window is obtained by opening the **Packages** folder as shown below. There are thirteen folders located in **Packages** in Version 1.2 (fourteen folders are located in Version 2.0 **Packages**); each folder contains several packages. The contents of each folder can be seen by double-clicking on the appropriate icon.

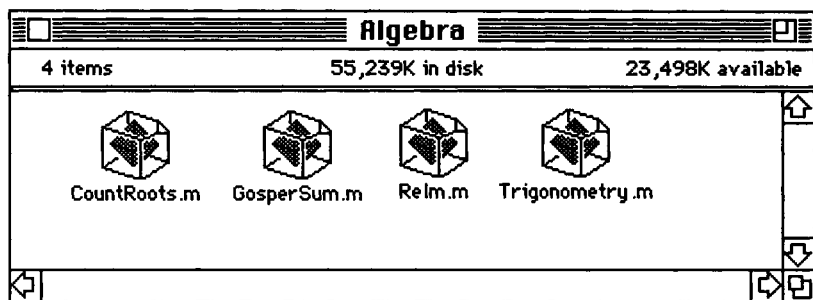


Chapters 7, 8, and 9 contain discussions of some *Mathematica* packages. **Chapter 7** discusses elementary packages from the **Algebra**, **Linear Algebra**, **Calculus**, and **Discrete Math** folders; **Chapter 8** discusses some of the packages contained in the **Graphics** folder; and **Chapter 9** discusses some of the more specialized packages contained in the **Numerical Math** and **Data Analysis** folders.

■ A Note Regarding Packages:

When directly opening a package, notice that the functions defined within the package are listed along with their definitions. Consequently, users can usually determine the purpose of a package by reading the beginning statements and experimenting.

■ 7.1 Algebra



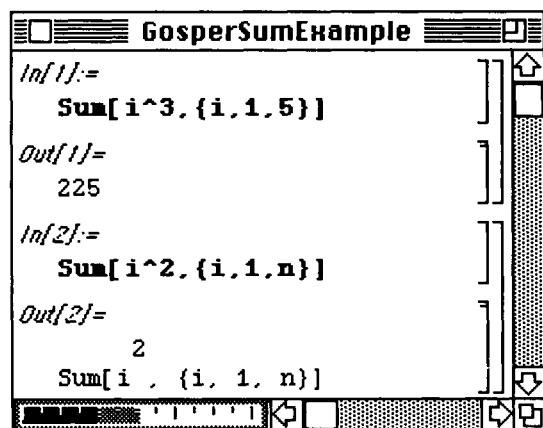
*The package **Trigonometry.m** has been used throughout Mathematica by Example. Hence, here we discuss the packages **ReIm.m** and **GosperSum.m**. Note that the package **GosperSum.m** is **NOT** included in Version 2.0. Instead, it is replaced by the (considerably) expanded package **SymbolicSum.m**.*

■ GosperSum.m

- o The **GosperSum.m** package is not included with Version 2.0; instead it is superseded by the package **SymbolicSum.m**.

The symbolic summation of some series of the form $\sum_{k=1}^n a_k$, which are useful in many areas of mathematics can be determined with the use of the **GosperSum.m** package.

These calculations are not possible without this package. The built-in *Mathematica* command **Sum** can be used for finite sums as illustrated in the first example below. However, this command cannot find a closed form of the summation in the second example. Therefore, the **GosperSum.m** package must be loaded.



*Although Mathematica is able to compute $\sum_{i=1}^5 i^3$ with the command **Sum[i^3, {i, 1, 5}]**, Mathematica is unable to compute a closed form for $\sum_{i=1}^n i^2$, n any integer.*

□ Example:

The command which can be used to determine a closed form expression of some sums of the form

$\sum_{k=kmin}^{kmax} a[k]$ is `GosperSum[a[k], {k, kmin, kmax}]`. After the package `GosperSum .m`

is loaded, a table is constructed below which consists of the summation formula for each of the series

$\sum_{k=1}^n k^i$ for $i=2, 4, 6,$ and 8 . Since `Print` is used in the `Table` command, the results

appear in a print cell and can be accessed for later use. Although quite useful in many cases, `GosperSum` cannot compute the closed form summation for many series as illustrated in the second example.

```

In[16]:=
<<GosperSum.m
In[17]:=
Table[Print[GosperSum[k^i, {k, 1, n}]], {i, 2, 8, 2}]

n (1 + 3 n + 2 n )
-----
6

n (-1 + 10 n + 15 n + 6 n )
-----
30

n (1 - 7 n + 21 n + 21 n + 6 n )
-----
42

n (-3 + 20 n - 42 n + 60 n + 45 n + 10 n )
-----
90

Out[17]=
{Null, Null, Null, Null}

In[21]:=
GosperSum[Sin[k], {k, 1, n}]

Out[21]=
Sum[Sin[k], {k, 1, n, 1}]

```

closed form for $\sum_{i=1}^n i^2$, n any integer.

closed form for $\sum_{i=1}^n i^4$, n any integer.

closed form for $\sum_{i=1}^n i^6$, n any integer.

closed form for $\sum_{i=1}^n i^8$, n any integer.

Even GosperSum cannot find a closed form for $\sum_{k=1}^n \text{Sin}(k)$, n any integer.

● SymbolicSum.m

- In Version 2.0 the package **SymbolicSum.m** replaces the package **GosperSum.m** from previous versions of *Mathematica*. In general, *Mathematica*'s standard built-in commands cannot compute symbolic sums

of the form $\sum_{k=n_1}^{n_2} f(k)$ when n_1 and n_2 are not specific numbers.

The command **SymbolicSum[f[k], {k, n1, n2}]** attempts to write the symbolic sum

$\sum_{k=n_1}^{n_2} f[k]$ in a closed form when $f[k]$ is a rational function.

- Example:

Find closed forms for (i) $\sum_{n=1}^k (n+2)(n-3)$ and (ii) $\sum_{n=1}^k \frac{1}{n+3}$. For (ii), evaluate when $k=100$.

Version2.0SymbolicSum		
<i>In[1]:=</i>	<code><<SymbolicSum.m</code>	
<i>In[18]:=</i>	<code>SymbolicSum[(n+2)(n-3), {n, 1, k}]</code>	
<i>Out[18]=</i>	$\frac{k(-19 + k^2)}{3}$	computes a closed form for $\sum_{n=1}^k (n+2)(n-3)$.
<i>In[21]:=</i>	<code>sum=SymbolicSum[1/(n+3), {n, 1, k}]</code>	computes a closed form for $\sum_{n=1}^k \frac{1}{n+3}$
<i>Out[21]=</i>	$\frac{-11 + 6 \text{ EulerGamma}}{6} + \text{PolyGamma}[0, 4 + k]$	
<i>In[22]:=</i>	<code>sum /. k->100</code>	computes $\sum_{n=1}^{100} \frac{1}{n+3}$ by replacing k by 100 in <code>sum</code> .
<i>Out[22]=</i>	$\frac{1513497672673382743451892618928750372177 \backslash 18429 / 2901204254058795599770580857416215575 \backslash 6055616 - \text{EulerGamma} + -11 + 6 \text{ EulerGamma}}{6}$	

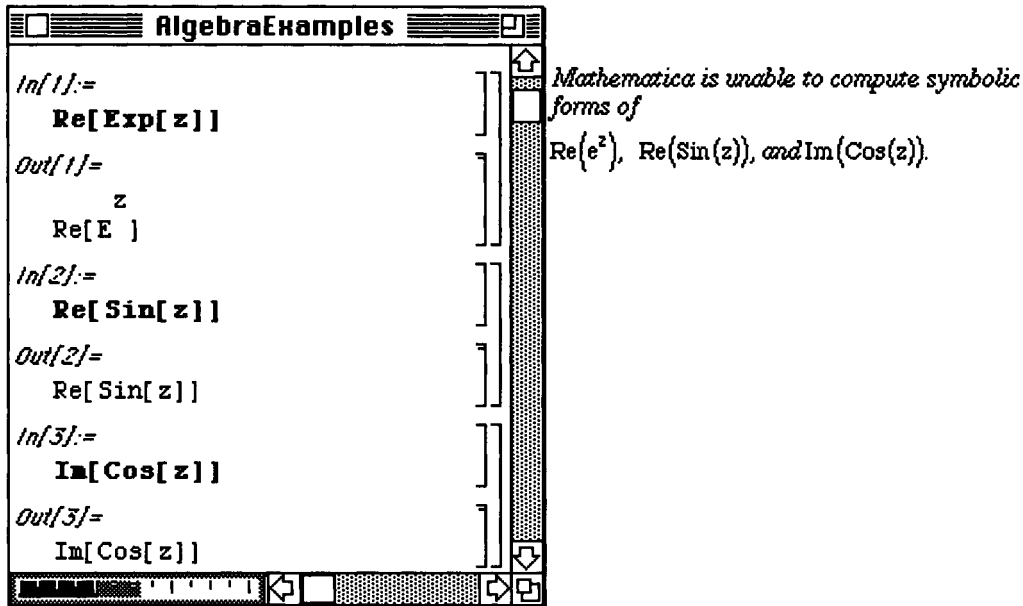
Since $\sum_{n=1}^{100} \frac{1}{n+3}$ is not a symbolic sum, the commands **Sum** and **NSum** can be used to

calculate $\sum_{n=1}^{100} \frac{1}{n+3}$:

<pre>In[23]:= N[EulerGamma] Out[23]= 0.577216 In[24]:= sum /. k->100 // N Out[24]= 3.38346 In[25]:= Sum[1/(n+3), {n, 1, 100}] Out[25]= 981610226095936883493952795069110849982\ 83133 / 2901204254058795599770580857416215575\ 6055616 In[26]:= N[Sum[1/(n+3), {n, 1, 100}]] Out[26]= 3.38346</pre>	<p>EulerGamma is Euler's constant with approximate value 0.577.</p> <p>sum /. k->100 // N computes a numerical approximation of $\sum_{n=1}^{100} \frac{1}{n+3}$.</p> <p>Although the command cannot be used to compute symbolic sums, it computes the exact value of $\sum_{n=1}^{100} \frac{1}{n+3}$.</p> <p>computes a numerical approximation of $\sum_{n=1}^{100} \frac{1}{n+3}$. The same results would be obtained with the command NSum[1/(n+3), {n, 1, 100}]</p>
---	--

■ **ReIm.m**

The built-in *Mathematica* functions **Re[z]** and **Im[z]** compute the real and imaginary parts of the complex number **z**. However, these commands are not helpful when working with complex-valued functions as shown below. Since determining the real and imaginary parts of functions is important in many problems such as the solution of differential equations, a technique to find the quantities is necessary.



The package **ReIm.m** is loaded so that many of the identities from complex analysis can be used. Hence, simplification can be accomplished in the following way:

$$e^z = e^{x+iy} = e^x e^{iy} = e^x (\cos y + i \sin y).$$

Therefore, the real and imaginary parts are determined where $x = \text{Re}[z]$ and $y = \text{Im}[z]$ below. Similar calculations yield $\text{Re}[\text{Sin}[z]]$ and $\text{Im}[\text{Cos}[z]]$.

```

AlgebraExamples

In[4]:=
  <<ReIm.m

In[5]:=
  Re[Exp[z]]

Out[5]=
  Re[z]
  E      Cos[Im[z]]

In[6]:=
  Re[Sin[z]]

Out[6]=
  -Im[z]      Im[z]
  ((E      + E      )
      Sin[Re[z]]) / 2

In[7]:=
  Im[Cos[z]]

Out[7]=
  -Im[z]      Im[z]
  -((-E      + E      )
      Sin[Re[z]]) / 2

```

The package **ReIm.m** contains familiar identities from complex analysis.

After loading the package **ReIm.m** Mathematica is able to compute

$$\text{Re}(e^z) = e^{\text{Re}(z)} \text{Cos}(\text{Im}(z)),$$

$$\text{Re}(\text{Sin}(z)) = \frac{(e^{-\text{Im}(z)} + e^{\text{Im}(z)}) \text{Sin}(\text{Re}(z))}{2},$$

$$\text{Im}(\text{Cos}(z)) = \frac{(e^{-\text{Im}(z)} - e^{\text{Im}(z)}) \text{Sin}(\text{Re}(z))}{2}, \text{ and}$$

other such formulas.

□ Example:

These techniques are useful in determining the solutions to differential equations with complex eigenvalues. Consider the second order linear equation with constant coefficients, $y''+y'+y=0$. Problems of this form arise when modeling spring problems. In this case, a spring of mass, m , and spring constant, k , is displaced in a medium with damping coefficient, a , where $m = k = a$. The solution of the equation is known to be

$$y(x) = e^{-x/2} \left(c_1 \cos\left(\frac{\sqrt{3}x}{2}\right) + c_2 \sin\left(\frac{\sqrt{3}x}{2}\right) \right).$$
 Unfortunately, since this equation has complex

eigenvalues, **DSolve** yields a solution involving complex terms. The **Trigonometry.m** package is useful in the application of DeMoivre's formula. Yet, the results obtained with the command **ComplexToTrig** are less than desirable. Hence, an alternate approach is demonstrated.

```

In[8]:=
list=DSolve[y''[x]+y'[x]+y[x]==0,y[x],x]
Out[8]=
{{y[x] ->
  E^((-1 - Sqrt[-3]) x)/2 C[1] +
  E^((-1 + Sqrt[-3]) x)/2 C[2]}}
In[9]:=
<<Trigonometry.m
In[14]:=
expression=list[[1,1,2]]
Out[14]=
((-1 - Sqrt[-3]) x)/2 E C[1] +
((-1 + Sqrt[-3]) x)/2 E C[2]
In[15]:=
ComplexToTrig[expression]
Out[15]=
-x/2 - (Sqrt[-3] x)/2 E C[1] +
-x/2 + (Sqrt[-3] x)/2 E C[2]

```

solves the differential equation $y''+y'+y=0$ for y . Mathematica's expresses the solution as a list containing imaginary numbers. For convenience, the list is named **list**.

The package **Trigonometry.m** contains the command **ComplexToTrig** which applies DeMoivre's formulas to expressions.

expression=list[[1,1,2]] extracts the solution from the nested list.

In this case, **ComplexToTrig[expression]** does not produce the desired simplification.

The real part of the solution, **reexpression**, is obtained with **Re[expression]**. Note that in the calculation of **reexpression**, the variable x is considered a complex number.

```

In[16]:=
rexpression=Re[expression]
Out[16]=
(Sqrt[3] Im[x] - Re[x])/2
E
      -Im[x] - Sqrt[3] Re[x]
Cos[-----] Re[C[1]] +
      2

(-(Sqrt[3] Im[x]) - Re[x])/2
E
      -Im[x] + Sqrt[3] Re[x]
Cos[-----] Re[C[2]] -
      2

```

computes the real part of expression.

Then, the imaginary part of **expression** is found with **Im[expression]** and called **iexpression**.

```

AlgebraExamples
In[17]:=
iexpression=Im[expression]
Out[17]=
(Sqrt[3] Im[x] - Re[x])/2
E
      -Im[x] - Sqrt[3] Re[x]
Cos[-----] Im[C[1]] +
      2

(-(Sqrt[3] Im[x]) - Re[x])/2
E
      -Im[x] + Sqrt[3] Re[x]
Cos[-----] Im[C[2]] +
      2

(Sqrt[3] Im[x] - Re[x])/2
E
      Re[C[1]]

      -Im[x] - Sqrt[3] Re[x]
Sin[-----] +
      2

(-(Sqrt[3] Im[x]) - Re[x])/2
E
      Re[C[2]]

```

computes the imaginary part of expression.

The expressions which result from the commands **Re** and **Im** can be further simplified by establishing several rules. This is done below with **rule**. Since **x** is a real number, **rule** replaces **Im[x]** with **0** and **Re[x]** with **x**. Also, the constants **C[1]** and **C[2]** are real, so they should have zero imaginary parts. This is done in **rule** with **Im[C[1]] -> 0**, **Im[C[2]] -> 0**, **Re[C[1]] -> C[1]**, and **Re[C[2]] -> C[2]**. Therefore, the command **reexpression /. rule** applies **rule** to **reexpression**. The result is called **solone**.

```

In[18]:=
  rule={Im[x]->0,Re[x]->x,Im[C[1]]->0,
        Re[C[1]]->C[1],Im[C[2]]->0,
        Re[C[2]]->C[2]}

Out[18]=
  {Im[x] -> 0, Re[x] -> x, Im[C[1]] -> 0,
   Re[C[1]] -> C[1], Im[C[2]] -> 0,
   Re[C[2]] -> C[2]}

In[19]:=
  solone=reexpression /. rule

Out[19]=
  -(Sqrt[3] x)
  C[1] Cos[-----]
             2
  ----- +
             x/2
             E

  Sqrt[3] x
  C[2] Cos[-----]
             2
  -----
             x/2
             E
  
```

applies rule to expression.

Next, `solone` is simplified and factored with `TrigReduce[solone]`. The resulting expression involves the constant, `C[1]+C[2]` which can be replaced with a single constant `C[0]`. This is done with `%/. (C[1]+C[2]) ->C[0]`. This gives the real part of the solution to the differential equation and is named `solutionone`.

```

AlgebraExamples
In[20]:=
  TrigReduce[solone] // Factor
Out[20]=
      Sqrt[3] x
(C[1] + C[2]) Cos[-----]
                    2
-----
      x/2
      E

In[21]:=
  solutionone=% /. (C[1]+C[2]) ->C[0]
Out[21]=
      Sqrt[3] x
C[0] Cos[-----]
            2
-----
      x/2
      E

```

applies basic trigonometric identities and factors `solone`.

replaces `C[1]+C[2]` by `C[0]` in the preceding calculation. The result is named `solutionone`.

A similar approach leads to the imaginary part of the solution. First, `soltwo` is obtained by applying `rule` to `iexpression`. Then, `soltwo` is simplified and factored. This yields an expression involving the constant, `C[1]-C[2]`. This constant is replaced with the imaginary constant, `I C[1]`, in order to obtain the desired results. The expression which results after substitution of this constant is called `solutiontwo`. Notice that `solutiontwo` involves `I`.

```

AlgebraExamples

In[30]:=
  soltwo=iexpression /. rule
Out[30]=
  -(Sqrt[3] x)      Sqrt[3] x
  C[1] Sin[-----]  C[2] Sin[-----]
          2              2
  ----- + -----
      x/2              x/2
      E                E

In[31]:=
  TrigReduce[soltwo] // Factor
Out[31]=
  Sqrt[3] x
  (C[1] - C[2]) Sin[-----]
                    2
  -(-----)
      x/2
      E

In[32]:=
  solutiontwo=X /. (C[1]-C[2]) ->I C[1]
Out[32]=
  Sqrt[3] x
  -I C[1] Sin[-----]
                2
  -----
      x/2
      E

```

The general solution to the differential equation, **solution**, is therefore obtained with the linear combination of **solutionone** and **solutiontwo**, **solutionone+I solutiontwo**. The **solution** is then verified by the substitution of **solution** into the original differential equation.

□
□
AlgebraExamples

In[33]:=

solution=solutionone+I solutiontwo

Out[33]=

$$\begin{array}{c}
 \text{C[0] } \cos\left[\frac{\sqrt{3} x}{2}\right] + \text{C[1] } \sin\left[\frac{\sqrt{3} x}{2}\right] \\
 \hline
 e^{x/2} + e^{x/2}
 \end{array}$$

In[34]:=

D[solution, {x, 2}] + D[solution, x] + solution

Out[34]=

0

the solution to the differential equation $y''+y'+y=0$ is

solution=solutionone+I solutiontwo.

*verifies that **solution** is the solution to the differential equation $y''+y'+y=0$.*

■ 7.2 Linear Algebra

● Cholesky.m

- **Cholesky.m** is contained within the **Linear Algebra** folder in Version 2.0; **Cholesky.m** is not available with Version 1.2.

The **complex conjugate transpose** of an $m \times n$ matrix A is the $n \times m$ matrix

\overline{A}^T obtained from A by taking the complex conjugate of each element of A and transposing the result.

An $m \times n$ matrix A is **Hermitian** means $A = \overline{A}^T$. An $n \times n$ Hermitian matrix A is **positive definite** means the eigenvalues of A are positive. This is equivalent to saying

$\langle Ax, x \rangle = Ax \cdot \overline{x}$ is positive for every nonzero vector x .

The concept of positive definite matrices is of importance in many areas such as physics and geometry. An $n \times n$ matrix is **symmetric** if A is equal to its transpose.

The **Cholesky.m** package contains the command **CholeskyDecomposition[a]** which yields a matrix u such that

$$u^T u = a \quad \text{where } u^T = \text{the transpose of } u.$$

In order to determine this decomposition, however, a must be a symmetric, positive definite (and, hence, real) matrix. Therefore, the command **CholeskyDecomposition[a]** can serve as a test to determine if the matrix a is positive definite.

o Example:

This command is illustrated below with the 2x2 matrix, $\{\{2, 1\}, \{1, 2\}\}$. The matrix which results is called **u** so that the property stated above can be verified with **a==Transpose[u].u**. Since a value of True is obtained, the result is correct.

The screenshot shows a Mathematica notebook window titled "CholeskyDecomp". The notebook contains the following code and outputs:

```

In[25]:=
<<Cholesky.m
Out[25]:=
The package Cholesky.m
is contained in the folder LinearAlgebra.

In[27]:=
a={{2,1},{1,2}};
MatrixForm[a]
Out[27]:=MatrixForm=
  2  1
  1  2

In[28]:=
u=CholeskyDecomposition[a]
Out[28]=
  (Sqrt[2], 1/Sqrt[2]),
  {0, Sqrt[3/2]}

In[29]:=
Transpose[u]
Out[29]=
  (Sqrt[2], 0),
  {1/Sqrt[2], Sqrt[3/2]}

In[30]:=
a==Transpose[u].u
Out[30]=
True

```

Callouts explaining the code:

- Line 25:** The package **Cholesky.m** is contained in the folder **LinearAlgebra**.
- Line 27:** Defines **a** to be the matrix $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and then expresses **a** in matrix form.
- Line 28:** In this case, the command **Eigenvalues[a]** yields 3 and 1.
- Line 28:** Computes the Cholesky decomposition of **a** to be $\begin{bmatrix} \sqrt{2} & 1/\sqrt{2} \\ 0 & \sqrt{3/2} \end{bmatrix}$ and names the result **u**.
- Line 29:** Compute the transpose of **u**.
- Line 30:** Tests the equality of **a** and **Transpose[u].u**.

As mentioned above, the **CholeskyDecomposition** command only applies to symmetric, positive definite matrices. However, the command does not check for these properties before trying to perform the decomposition. This is demonstrated in the two examples below. In the following example, a non-symmetric matrix is considered. The resulting matrix is incorrect since a **False** response to **a==Transpose[u].u** is given:

```
In[33]:=
```

```
Clear[a,u];
a={{2,1},{0,2}};
MatrixForm[a]
```

```
Out[33]//MatrixForm=
```

```
2 1
```

```
0 2
```

```
In[34]:=
```

```
u=CholeskyDecomposition[a]
```

```
Out[34]=
```

```
{(Sqrt[2],  $\frac{1}{\text{Sqrt}[2]}$ ),
```

```
{0, Sqrt[-]}}
```

```
In[35]:=
```

```
a==Transpose[u].u
```

```
Out[35]=
```

```
False
```

Clears all prior definition of a and u, then defines a to be the matrix $\begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$ and expresses the result in matrix form.

Computes the Cholesky decomposition of to be $\begin{bmatrix} \sqrt{2} & 1/\sqrt{2} \\ 0 & \sqrt{3}/2 \end{bmatrix}$ and names the result u.

In this case, a and Transpose[u].u are not the same.

In the next case, the matrix is not positive definite, so several error messages result:

```

In[38]:=
Clear[a,u];
a={{1,-1,-1},{-1,1,1},{-1,1,1}};
MatrixForm[a]

Out[38]//MatrixForm=
  1   -1  -1
 -1   1   1
 -1   1   1

In[39]:=
u=CholeskyDecomposition[a]

Power::infy:
          1
Infinite expression -
          0
encountered.

Infinity::indet:
Indeterminate expression
0 ComplexInfinity
encountered.

Out[39]=
{{1, -1, -1},
 {0, 0, Indeterminate}}.

```

In this case, the Cholesky decomposition cannot be computed.

The matrix which results from the decomposition may contain imaginary numbers. This is shown below with the matrix **a**. However, the resulting matrix is verified since **Transpose[r].r==a**.

```

In[42]:=
Clear[a];
a={{2,-1,-1},{-1,1,-1},{-1,-1,2}}
MatrixForm[a]

Out[42]//MatrixForm=
  2   -1  -1
  -1   1  -1
  -1  -1   2

In[43]:=
r=CholeskyDecomposition[a]

Out[43]=
  1
  { {Sqrt[2], -(-----),
      Sqrt[2]
    },
    { 0, -----, ----- },
    { 0, 0, I Sqrt[3] } }
  Sqrt[2]
  Sqrt[2] Sqrt[2]

In[44]:=
a==Transpose[r].r

Out[44]=
True

```

After defining **a**
to be the matrix

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

computes the Cholesky
decomposition of **a**
to be

$$\begin{bmatrix} \sqrt{2} & -1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & -3/\sqrt{2} \\ 0 & 0 & i\sqrt{3} \end{bmatrix}$$

and names the result **r**.

Tests the equality of **Transpose[r].r**
and **a**.

○ **Application: Quadratic Equations**

An application of positive definite matrices is the analysis of quadratic equations.

For any symmetric matrix A , the product $f = x^T Ax$ is a pure quadratic form :

$$\begin{aligned} x^T Ax &= [x_1, x_2, \dots, x_n] \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= a_{11}x_1^2 + a_{12}x_1x_2 + a_{21}x_2x_1 + \cdots + a_{nn}x_n^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j . \end{aligned}$$

If the matrix A is positive definite, then $f > 0$ for all values of x . Consider the following quadratic equation:

$$2x^2 - 2xy + 2y^2 - 2xz + 2yz + 2z^2 = -1.$$

We attempt to determine if this equation has any real roots by considering the matrix A . If the symmetric matrix

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

which results when this equation is represented as $x^T Ax$ is positive definite, then $x^T Ax > 0$ for all x .

Hence, $x^T Ax \neq -1$ for any values of x .

The matrix A is defined below as a . Note that this matrix yields the appropriate quadratic form as shown with `Expand[Transpose[vec].a.vec]` where $\text{vec}=\{x,y,z\}$. The matrix A is then shown to be positive definite by using `CholeskyDecomposition`. Therefore, this equation has no real solutions.

```

In[47]:=
Clear[a];
a={{2,-1,-1},{-1,2,1},{-1,1,2}};
vec={x,y,z};

In[48]:=
Expand[Transpose[vec].a.vec]

Out[48]=

$$2x^2 - 2xy + 2y^2 - 2xz + 2yz + 2z^2$$


In[49]:=
r=CholeskyDecomposition[a]

Out[49]=

$$\left\{ \left\{ \sqrt{2}, -\frac{1}{\sqrt{2}} \right\}, \right.$$


$$\left. -\left( \frac{1}{\sqrt{2}} \right) \right\},$$


$$\left\{ 0, \sqrt{-\frac{3}{2}}, \frac{1}{\sqrt{6}} \right\},$$


$$\left\{ 0, 0, \frac{2}{\sqrt{3}} \right\}$$


In[50]:=
a==Transpose[r].r

Out[50]=
True

```

■ CrossProduct.m

Neither Version 1.2 nor Version 2.0 contain a built-in command for computing the cross product of two three-dimensional vectors. In order to compute cross products of vectors, Version 1.2 users must load the package **Cross.m** in the **Linear Algebra** folder to use the command **Cross[vec1,vec2]** which computes the cross product of vectors **vec1** and **vec2**; Version 2.0 users must use the package **CrossProduct.m** in the **Linear Algebra** folder to use **CrossProduct[vec1,vec2]**. Of course, the vector which results from the cross product is orthogonal to each of the original vectors. This is verified below as is the property that the cross product of two parallel vectors is the zero vector.

□ Example:

*If using Version 2.0, the package **CrossProduct.m** contains the command **CrossProduct**.*

computes $\text{Det} \begin{bmatrix} i & j & k \\ 2 & 1 & 2 \\ 3 & 1 & -3 \end{bmatrix}$, where

$i = \{1,0,0\}$, $j = \{0,1,0\}$, and $k = \{0,0,1\}$.

verifies $\{2,1,2\}$ and $\{-1,12,-5\}$ are perpendicular.

verifies that $\{2,1,2\}$ and $\{4,2,4\}$ are parallel.

□ Example:

One application of the cross product is in computing the area of a triangle the vertices of which are the points P, Q, and R. This area is given by the formula

$$\text{Area} = \frac{1}{2} |\overline{PQ} \times \overline{PR}|.$$

Mathematica does not contain a built-in function to calculate the length of vectors. Hence, the function **length[v]** which yields this value is defined as the square root of the dot product of **v** with itself. This function is illustrated with the vector $\{1, 2, -1\}$ to show that the exact value results. There is no restriction on the dimension of vectors to be used with **length** as seen below with $\{1, 3, -2, 4, 0, 8\}$.

The area of the triangle is computed by, first, defining the points \mathbf{p} , \mathbf{q} , and \mathbf{r} ; and then determining the vectors \mathbf{pq} and \mathbf{pr} . The numerical approximation of the area is easily found with

`N[.5 length[Cross[pq,pr]]]`.

```
In[5]:=
  length[v_]:=Sqrt[v.v]
  length[{1,2,-1}]

Out[5]=
  Sqrt[6]

In[6]:=
  length[{1,3,-2,4,0,8}]

Out[6]=
  Sqrt[94]

In[7]:=
  p={4,-3,1}
  q={6,-4,7}
  r={1,2,2}
  pq=q-p
  pr=r-q:

In[8]:=
  area=N[.5 length[Cross[pq,pr]]]

Out[8]=
  18.775
```

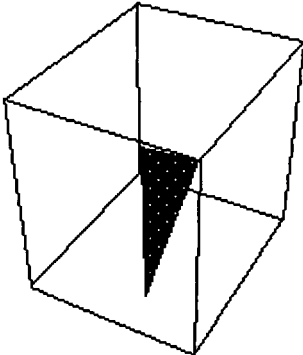
`length[v]` computes $\sqrt{v \cdot v}$.

Defines $\mathbf{p} = \{4, -3, 1\}$, $\mathbf{q} = \{6, -4, 7\}$,
 $\mathbf{r} = \{1, 2, 2\}$, $\mathbf{pq} = \mathbf{q} - \mathbf{p}$, and
 $\mathbf{pr} = \mathbf{r} - \mathbf{p}$.

Then the area of the triangle with
 vertices \mathbf{p} , \mathbf{q} , \mathbf{r} is given by

$$\frac{1}{2} \|\mathbf{pq} \times \mathbf{qr}\|.$$

This triangle is shown below:

<pre>In[32]:= Show[Graphics3D[Polygon[{{4,-3,1},{6,-4,7},{1,2,2}}]]]</pre>  <pre>Out[32]= -Graphics3D-</pre>	<p>The triangle with vertices p, q, and r can be visualized using the Show, Graphics3D, and Polygon commands.</p>
---	---

Another similar application of the cross product is in finding the distance from a line containing the points P and Q to a point R not on the line. This distance is known to be

$$d = \frac{1}{|\overline{PQ}|} |\overline{PQ} \times \overline{PR}|.$$

A function which computes this distance is defined as **distance** below and is illustrated by finding the distance from the point (2,1,-2) to the line through the points (3,-4,1) and (-1,2,5). Since **distance** is defined in terms of the points P, Q, and R, this distance is found by simply entering the appropriate points.

<pre>In[9]:= distance[p_, q_, r_] := Block[{pq, pr, cross, d}, pq = q - p; pr = r - p; cross = Cross[pq, pr]; d = N[length[cross]/length[pq]]]</pre> <pre>In[10]:= distance[{2,1,-2},{3,-4,1},{-1,2,5}]</pre> <pre>Out[10]= 7.36012</pre>	<p>The variables pq, pr, cross and d are local to the user-defined function distance.</p> <p>computes the distance from the point (2,1,-2) to the line passing through the points (3,-4,1) and (-1,2,5)</p>
--	--

A useful feature of the `Cross[v1, v2]` command is that when numerical vectors are not given, *Mathematica* gives the formula used to compute the cross product. This can be used as a tool in finding vectors with certain properties. For example, the function `cp[p, q, r]` defined below computes the cross product of the vectors \mathbf{pq} and \mathbf{pr} . The arbitrary points $\{p_1, p_2, p_3\}$, $\{q_1, q_2, q_3\}$, and $\{r_1, r_2, r_3\}$ are used below to show that the cross product formula is obtained. Then, this formula is used to determine the value(s) of p_3 such that the triangle formed by the points $\{1, -1, p_3\}$, $\{1, 2, 0\}$, and $\{-3, 5, -1\}$ has area a . (Hence, the cross product has magnitude $2a$.) This is accomplished by substituting the known points into `cp` and solving for p_3 .

```
In[11]:=
Clear[p, q, r];
cp[p_, q_, r_] := Cross[q - p, r - p]
vec = cp[{1, -1, p3}, {1, 2, 0}, {-3, 5, -1}]
```

```
Out[11]=
{-3 + 3 p3, 4 p3, 12}
```

```
In[12]:=
l = length[vec]
```

```
Out[12]=
Sqrt[144 + 16 p32 + (-3 + 3 p3)2]
```

```
In[14]:=
sol = Simplify[Solve[l == 2a, p3]]
```

```
Out[14]=
{{p3 ->  $\frac{18 + \text{Sqrt}[-14976 + 400 a]}{50}$ },
{p3 ->  $\frac{18 - \text{Sqrt}[-14976 + 400 a]}{50}$ }}
```

The following calculations reveal two possibilities for **p3**. However, these roots are imaginary if **a** < 6.11882 (Negative values of **a** are disregarded.) Therefore, only values of **a** greater than 12.2376 can be considered. An example is worked and verified for **a=7**.

```

In[15]:=
  Solve[-14976+400a^2==0,a]//N
Out[15]=
  {{a -> 6.11882}, {a -> -6.11882}}
In[16]:=
  sol=Solve[l==14,p3]//N
Out[16]=
  {{p3 -> -1.}, {p3 -> 1.72}}
In[17]:=
  p=sol[[1,1,2]]
Out[17]=
  -1.
In[18]:=
  test=cp[{1,-1,p},{1,2,0},{-3,5,-1}]
Out[18]=
  {-6., -4., 12.}
In[19]:=
  length[test]//N
Out[19]=
  14.

```

● MatrixManipulation.m

- Version 2.0 contains the package `MatrixManipulation.m` which contains several commands useful for manipulating matrices. `MatrixManipulation.m` is not contained with Version 1.2. The command `AppendColumns[m1, m2, m3, ...]` yields a new matrix composed of the submatrices `m1, m2, ...`, by joining the columns of `m1, m2, ...` while `AppendRows[m1, m2, ...]` performs a similar operation by joining the rows of the matrices. In each command, the submatrices must have the same number of columns or rows, respectively. Several examples of these commands are given below and viewed in `MatrixForm` to better understand the results. After defining the 2x2 matrices `a` and `b`, the command `AppendColumns[a, b]` appends the columns of `b` to the columns of `a`, and `AppendRows[a, b]` adds the rows of `b` to those of `a`.

○ Example:

```

In[9]:=
  <<MatrixManipulation.m

In[12]:=
  a={{1,2},{3,4}};
  b={{5,6},{7,8}};
  AppendColumns[a,b]//MatrixForm

Out[12]//MatrixForm=
  1  2
  3  4
  5  6
  7  8

In[13]:=
  AppendRows[a,b]//MatrixForm

Out[13]//MatrixForm=
  1  2  5  6
  3  4  7  8
  
```

Defines $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

AppendsColumns[a,b]//MatrixForm produces $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$

AppendsRows[a,b]//MatrixForm produces $\begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$

The matrix **c** is then defined to illustrate that more than two matrices can be used as arguments as well as the fact that correct dimensions must be used. The command **AppendColumns[a, c]** is not evaluated since **c** has more columns than **a**.

```

In[15]:=
  c={{11,12,13},{14,15,16}};
  AppendRows[a,b,c]//MatrixForm

Out[15]//MatrixForm=
  1   2   5   6   11  12  13
  3   4   7   8   14  15  16

In[16]:=
  AppendColumns[a,c]

Out[16]=
  AppendColumns[{{1, 2}, {3, 4}},
                {{11, 12, 13}, {14, 15, 16}}]
  
```

Defines $c = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \end{bmatrix}$.

AppendRows[a,b,c]//MatrixForm
 produces $\begin{bmatrix} 1 & 2 & 5 & 6 & 11 & 12 & 13 \\ 3 & 4 & 7 & 8 & 14 & 15 & 16 \end{bmatrix}$.

AppendColumns[a,c] produces
 nothing since **a** and **c** do not
 have the same number of
 columns.

○ **Application:** Computing the Adjacency Matrix of a Graph

An application of these commands is the manipulation of the adjacency matrix of a graph. Recall that two vertices of a graph are said to be **adjacent** if there is at least one edge joining them. Consider the graph **G** with no loops and **n** vertices labeled 1, 2, ..., **n**. The **adjacency matrix of G** is the **n x n** matrix in which the entry in row **i** and column **j** is the number of edges joining the vertices **i** and **j**. For example, suppose that a graph has the

$$\text{adjacency matrix } A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix}.$$

This matrix is represented as **adj** below. Then, suppose that two more vertices are added to the graph with vertex 5 adjacent to vertices 2 and 3; and vertex 6 adjacent to vertices 1 and 4. Instead of defining a new adjacency matrix for the revised graph (which can be quite cumbersome in many cases), these additions can be made with **AppendColumns** and **AppendRows**.

This is done below in `addtocols` and `addtorows`. The rows added in `addtocols` represent the edges from vertices 5 and 6 to vertices 1, 2, 3, and 4 while the columns added in `addtorows` give these same edges from the original set of vertices to vertices 5 and 6.

```

In[18]:=
  adj={{0,1,0,1},{1,0,1,2},{0,1,0,1},{1,2,1,0}};
  addtocols=AppendColumns[adj,{{0,1,1,0},{1,0,0,1}}]
Out[18]=
  {{0, 1, 0, 1}, {1, 0, 1, 2}, {0, 1, 0, 1}, {1, 2, 1, 0},
   {0, 1, 1, 0}, {1, 0, 0, 1}}
In[20]:=
  addtorows=AppendRows[addtocols,
    {{0,1},{1,0},{1,0},{0,1},{0,1},{1,0}}];
  addtorows//MatrixForm
Out[20]//MatrixForm=
  0  1  0  1  0  1
  1  0  1  2  1  0
  0  1  0  1  1  0
  1  2  1  0  0  1
  0  1  1  0  0  1
  1  0  0  1  1  0

```

An interesting fact concerning an adjacency matrix M is that the (i,j) th-element of the k th power of M represents the number of walks of length k from vertex i to vertex j . A **walk of length k** in a graph is a succession of k edges. This is important in problems in which the number of ways to travel between two locations must be determined.

Using the matrix given in `addtorows`, the number of walks of length 2 between every vertex pair is determined from `twowalks`. For example, there are 4 walks of length 2 from vertex 4 to vertex 5 as seen with `twowalks[[4,5]]`. The number of walks of length 3 are found in `threewalks`.

```

In[21]:=
  twowalks=addtorows.addtorows
Out[21]=
  {{3, 2, 2, 3, 2, 1}, {2, 7, 3, 2, 1, 4},
   {2, 3, 3, 2, 1, 2}, {3, 2, 2, 7, 4, 1},
   {2, 1, 1, 4, 3, 0}, {1, 4, 2, 1, 0, 3}}
In[22]:=
  twowalks[[4,5]]
Out[22]=
  4
In[23]:=
  threewalks=twowalks.addtorows
Out[23]=
  {{6, 13, 7, 10, 5, 8},
   {13, 10, 10, 23, 14, 5},
   {7, 10, 6, 13, 8, 5},
   {10, 23, 13, 10, 5, 14},
   {5, 14, 8, 5, 2, 9}, {8, 5, 5, 14, 9, 2}}

```

As the power increases, the built-in command `MatrixPower[matrix,k]` is useful. This is used below to find the number of walks of length 10.

```

In[24]:=
  tenwalks=MatrixPower[adj,10]
Out[24]=
  {{17408, 28160, 17408, 28160},
   {28160, 46080, 28160, 45056},
   {17408, 28160, 17408, 28160},
   {28160, 45056, 28160, 46080}}

```

MatrixManipulation.m also contains several well-known matrices such as the Hilbert matrix. The

Hilbert matrix H is given by $H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \dots \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \dots \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$.

The $n \times n$ Hilbert matrix is found with **HilbertMatrix[n]** and is illustrated with **HilbertMatrix[3]** below. This matrix is named **hm3** and its inverse is called **hm3inv**. This is verified with **hm3.hm3inv** which yields the 3×3 identity matrix. Next, the 10×10 Hilbert matrix, **hm10**, is computed so that the system of equations $\mathbf{Ax} = \mathbf{b}$ can be solved where $\mathbf{A} = \mathbf{hm10}$ and $\mathbf{b} = \{1, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$.

```
In[34]:=
```

```
hm3=HilbertMatrix[3]
```

```
Out[34]=
```

```

  1  1  1  1  1  1  1  1  1
{{1, -, -}, {-, -, -}, {-, -, -}}
  2  3  2  3  4  3  4  5

```

Produces $\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$

and names the result **hm3**.

```
In[35]:=
```

```
hm3inv=Inverse[hm3]
```

```
Out[35]=
```

```
{{9, -36, 30}, {-36, 192, -180}, {30, -180, 180}}
```

Computes the inverse of **hm3** and names the result **hm3inv**.

```
In[36]:=
```

```
hm3.hm3inv
```

```
Out[36]=
```

```
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

```
In[38]:=
```

```
hm10=HilbertMatrix[10];
Short[hm10,3]
```

Computes the 10×10 Hilbert matrix and displays the matrix in an abbreviated three-line form consisting of the first and last rows.

```
Out[38]::Short=
```

```

  1  1  1  1  1  1  1  1  1  1
{{1, -, -, -, -, -, -, -, -}, <<8>>,
  2  3  4  5  6  7  8  9  10

```

```

  1  1  1  1  1  1  1  1  1  1
{--, --, --, --, --, --, --, --, --, --}
 10 11 12 13 14 15 16 17 18 19

```

```
In[40]:=
```

```
b={1,0,0,0,0,0,0,0,0,0};
LinearSolve[hm10,b]
```

```
Out[40]=
```

```
{100, -4950, 79200, -600600, 2522520, -6306300,
9609600, -8751600, 4375800, -923780}
```

This system is also solved with $\mathbf{b1} = \{.75, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$ to illustrate that this system is sensitive to small changes.

```

In[40]:=
  b={1,0,0,0,0,0,0,0,0,0};
  LinearSolve[hm10,b]

Out[40]=
  {100, -4950, 79200, -600600, 2522520, -6306300,
    9609600, -8751600, 4375800, -923780}

In[42]:=
  b1={.75,0,0,0,0,0,0,0,0,0};
  LinearSolve[hm10,b1]

Out[42]=
  {75., -3712.5, 59400., -450450., 1.89189 106,
    -4.72972 106, 7.2072 106, -6.5637 106,
    3.28185 106, -692835.}

```

The sensitivity of the system $\mathbf{Ax}=\mathbf{b}$ can be measured with the **condition number** of \mathbf{A} . This number is defined in several ways, one of which is based on the 1-norm of \mathbf{A} :

For a nonsingular $m \times m$ matrix \mathbf{A} , the **condition number** of \mathbf{A} is denoted $c(\mathbf{A})$ and is defined

$$\text{by } c(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \text{ where } \|\mathbf{A}\| = \text{Max} \left\{ \sum_{i=1}^m |a_{ij}| \right\} \text{ for } j = 1, 2, \dots, m.$$

If $c(\mathbf{A})$ is small, then \mathbf{A} is said to be **well-conditioned**; \mathbf{A} is **ill-conditioned** otherwise.

The function `norm[matrix]` is defined below to compute the 1-norm of `matrix`. This definition is given in terms of a general $m \times n$ matrix. Note that this definition involves the built-in *Mathematica* command `Dimensions[matrix]` which gives the dimensions of an $m \times n$ matrix in the form $\{m, n\}$ as demonstrated below with `Dimensions[hm3]`.

The 1-norm is calculated for `hm3` and `hm3inv` using the function `norm`. The condition number of `matrix` is then defined as the product of the norm of `matrix` and the norm of its inverse. This function is called `cnum` and is illustrated with `hm3` to reveal a very large number. This is expected based on the results from the previous problem in which small changes in the original system involving `hm3` led to large changes in the solution.


```

In[43]:=
  Dimensions[hm3]

Out[43]=
  {3, 3}

In[44]:=
  norm[matrix_] :=
    Module[{dim, row, col, sums, max},
      dim = Dimensions[matrix];
      row = dim[[1]];
      col = dim[[2]];
      sums = Table[Sum[Abs[matrix[[i, j]]],
                    {i, 1, row}], {j, 1, col}];
      max = Max[sums]
    ]

In[45]:=
  norm[hm3]

Out[45]=
  11
  —
  6

In[46]:=
  norm[hm3inv]

Out[46]=
  408

In[47]:=
  cnum[matrix_] := N[norm[matrix]
                    norm[Inverse[matrix]]]

In[48]:=
  cnum[hm3]

Out[48]=
  748.

```

The variables `dim`, `row`, `col`, `sums`, and `max` are declared to be local to the user defined function `norm`.

Produces the condition number of a matrix. In this case, the condition number of the 3 x 3 Hilbert matrix is computed to be 748.

Mathematica has a built-in command which can be used in determining the condition number.

SingularValues[*m*] returns the list {*u*, *w*, *v*} where *w* is a list of non-zero singular values. The condition number is the ratio of the largest singular value of the matrix to the smallest one. The other information in the output of **SingularValues**[*m*] can be used to represent the matrix *m* as

Transpose[*u*].**DiagonalMatrix**[*w*].*v*. This is known as the singular value decomposition of *m*.

Hence, the condition number can be found by determining the ratio of the values in the list *w*. This is shown below with matrix *a*. The definition of condition number used in the built-in function differs from the one stated earlier since the value obtained with **SingularValues** and that found with **cnum**[*a*] differ. At any rate, each method yields a very large condition number. Hence, *a* is ill-conditioned, so numerical methods used to solve systems involving *a* are unreliable.

Unfortunately, the built-in command does not work with all matrices. An error message is given with `SingularValues[hm3]`. Therefore, the earlier procedure for calculating the condition number may prove to be more useful.

```

In[50]:=
  a={{34.9,23.6},{22.9,15.6}};
  s=SingularValues[a]

Out[50]=
  {{{-0.835497, -0.549495}, {-0.549495, 0.835497}},
   {50.4255, 0.0793249},
  {{-0.827801, -0.561022}, {-0.561022, 0.827801}}}

In[51]:=
  s[[2]]

Out[51]=
  {50.4255, 0.0793249}

In[52]:=
  cdn=Max[s[[2]]]/Min[s[[2]]]

Out[52]=
  635.683

In[53]:=
  cnum[a]

Out[53]=
  845.325

In[54]:=
  sing=SingularValues[hm3]

SingularValues::svdf:
  SingularValues has received a matrix with
  infinite precision.

SingularValues::svdf:
  SingularValues has received a matrix with
  infinite precision.

Out[54]=
  SingularValues[{{1, -, -}, {-, -, -}, {-, -, -}}]
  1 1 1 1 1 1 1 1
  2 3 2 3 4 3 4 5

```

• Orthogonalization.m

Version 2.0's **Orthogonalization.m** package contains useful commands for working with vector spaces. The main command in this package is **GramSchmidt[veclist, options]** which produces an orthonormal basis for the vectors in **veclist**. Recall that two properties of an orthonormal basis are that the inner product of any two basis vectors is 0, and the norm of each basis vector is 1. In the first example below, a basis for the list of vectors in **vecs** is found with **GramSchmidt[vecs]**.

○ Example:

Orthogonalization

```

In[2]:=
<<Orthogonalization.m

In[10]:=
vecs={{1,1,1},{-1,0,-1},{-1,2,3}};

In[11]:=
basis=GramSchmidt[vecs]

Out[11]=
      1      1      1
      Sqrt[3] Sqrt[3] Sqrt[3]
      { (-----, -----, -----) },
      1      2      1
      { -(-----), Sqrt[-], -(-----) },
      Sqrt[6] 3      Sqrt[6]
      1      1
      { -(-----), 0, ----- }
      Sqrt[2] Sqrt[2]

```

After loading the package **Orthogonalization.m**, **vecs** is defined to be the (nested) list **{{1,1,1},{-1,0,-1},{-1,2,3}}** which corresponds to the three vectors

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}, \text{ and } \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix}.$$

basis=GramSchmidt[vecs] computes three orthonormal vectors and names the resulting list of vectors **basis**. Since **basis** is a list, **basis[[1]]** corresponds to the first element of **basis**, **basis[[2]]** corresponds to the second element of **basis**, and **basis[[3]]** corresponds to the third element of **basis**.

Some of the previously mentioned properties are verified using the orthonormal basis found in **basis**.

```

In[17]:=
  basis[[1]].basis[[2]]
  basis[[2]].basis[[3]]
  basis[[1]].basis[[3]]
  basis[[1]].basis[[1]]
  basis[[2]].basis[[2]]
  basis[[3]].basis[[3]]

Out[12]=
  0

Out[13]=
  0

Out[14]=
  0

Out[15]=
  1

Out[16]=
  1

Out[17]=
  1

```

*To verify that **basis** is a set of orthonormal vectors, each dot product is computed. Notice that each result is displayed since semi-colons are not included at the end of each command.*

◦ **Application: Distance**

One application of the determination of an orthonormal set of basis vectors is in the calculation of the distance of a point \mathbf{x} from a subspace W . If \mathbf{u} is an orthonormal basis for W , then the distance from \mathbf{x} to S is defined as the magnitude of the component of \mathbf{x} which is orthogonal to each vector in W . This vector is found by projecting \mathbf{x} onto W using the basis vectors in \mathbf{u} . More formally, the distance is given by the formula :

$\|\mathbf{x} - \text{proj}_W \mathbf{x}\|$ where $\text{proj}_W \mathbf{x}$ is the projection of \mathbf{x} onto W .

In the example which follows, the distance of the point $\mathbf{x} = (4,1,-7)$ to the subspace W which consists of all vectors of the form (a,b,b) is found. Every vector in W can be written as the linear combination of the vectors $\{1, 0, 0\}$ and $\{0, 1, 1\}$. Hence, the orthogonal basis for W is found in **onbasis** below using $\{1, 0, 0\}$ and $\{0, 1, 1\}$.

The elements of **onbasis** are extracted in the usual manner as illustrated with **onbasis[[1]]**. The vector **x** is defined and the projection of **x** onto **W** is determined in **proj**. The orthogonal component is given in **diff**, and finally, the length of this vector is found in the standard way by using the dot product.

```

In[18]:=
  onbasis=GramSchmidt[{{1,0,0},{0,1,1}}]
Out[18]=
  {{1, 0, 0}, {0,  $\frac{1}{\text{Sqrt}[2]}$ ,  $\frac{1}{\text{Sqrt}[2]}$ }}
In[19]:=
  onbasis[[1]]
Out[19]=
  {1, 0, 0}
In[21]:=
  x={4,1,-7};
  proj=Projection[x,onbasis[[1]]]+
  Projection[x,onbasis[[2]]]
Out[21]=
  {4, -3, -3}
In[22]:=
  diff=x-proj
Out[22]=
  {0, 4, -4}
In[23]:=
  distance=Sqrt[diff.diff]
Out[23]=
  5/2
  2

```

onbasis

is a set of orthonormal vectors constructed from the linearly independent set of vectors

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

proj is defined to be the vector

$$\text{pr} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} (x) + \text{pr} \begin{bmatrix} 0 \\ 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} (x).$$

The length of a vector **v** is given by

$$\|v\| = \sqrt{v \cdot v}.$$

The above calculations to determine the distance from a point to a subspace can be generalized in the function **distance** below. The arguments of **distance** are the point **v** and a list of basis vectors, **vecs**, for **W**. The function performs all of the necessary calculations to find the desired distance. To verify that **distance** does yield the correct output, it is demonstrated with the same problem as before and gives the same result. First, all previously used definitions are cleared.

```

In[24]:=
  Clear[v,proj,onbasis,vecs,x,diff,distance]

In[25]:=
  distance[v_,vecs_]:=Module[{d,onbasis,proj,diff},
    onbasis=GramSchmidt[Evaluate[vecs]];
    proj=Sum[Projection[v,onbasis[[i]]],
      {i,1,Length[onbasis]}];
    diff=v-proj;
    d=Sqrt[diff.diff]
  ]

In[26]:=
  distance[{4,1,-7},{1,0,0},{0,1,1}]

Out[26]=
  5/2
  2

```

One of the options available to **GramSchmidt** is that of **InnerProduct**. In the earlier examples the dot product (the default value of **InnerProduct**) was used as the inner product. However, with this option, other vector spaces such as function spaces can be considered.

The inner product of two continuous functions **f** and **g** is given by

$$(f,g) = \int_a^b w(x) f(x) g(x) dx$$

where $f, g \in C[a, b]$ and $w(x)$ is a nonnegative weight function on (a, b) .

The **InnerProduct** option must be given in terms of a pure function. In order to do this, the syntax **(option->(function&))** must be used. Hence, the inner product with $w(x) = 1$ is defined with

InnerProduct->(Integrate[#1 #2, {x, -1, 1}]&) where **#1** and **#2** represent the first and second variables of the inner product, respectively.

The first command below yields the orthonormal basis from $\{1, x, x^2, x^3, x^4\}$, the basis for the space of fourth-order polynomials. Another **GramSchmidt** option is **Normalized->False**. When this is used, the basis vectors which result are orthogonal but not of length 1. The first command is repeated with this command and called **legendre5**.

```
In[2]:=
GramSchmidt[{1, x, x^2, x^3, x^4},
  InnerProduct->(Integrate[#1 #2, {x, -1, 1}])&]
```

```
Out[2]=
```

$$\left\{ \frac{1}{\sqrt{2}}, \sqrt{\frac{-1}{2}} x, 3 \sqrt{\frac{-1}{8}} \left(-\frac{1}{3} + x\right), \right. \\ \left. 5 \sqrt{\frac{-1}{8}} \left(\frac{-3x}{5} + x\right), \frac{105 \left(-\frac{1}{5} + x\right)^4 - \frac{6 \left(-\frac{1}{3} + x\right)^2}{7}}{2^{7/2}} \right\}$$

```
In[3]:=
```

```
legendre5=GramSchmidt[{1, x, x^2, x^3, x^4},
  InnerProduct->(Integrate[#1 #2, {x, -1, 1}])&,
  Normalized->False]
```

```
Out[3]=
```

$$\left\{ 1, x, -\frac{1}{3} \left(-\frac{1}{3} + x\right), \frac{-3x}{5} + x, -\frac{1}{5} \left(-\frac{1}{5} + x\right), \right. \\ \left. -\frac{6 \left(-\frac{1}{3} + x\right)^2}{7} \right\}$$

On the other hand, when the option `Normalized->True` is used, the basis vectors which result are orthogonal with length 1. In the following example, several integrals are computed to verify that the resulting basis vectors are orthogonal and have length 1:

```

In[2]:=
  gs=GramSchmidt[{1,x,x^2,x^3,x^4},
    InnerProduct->(Integrate[#1 #2,{x,-1,1}]&),
    Normalized->True]
Out[2]=
  {1/Sqrt[2], Sqrt[3/2] x, 3 Sqrt[5/8] (-1/3 + x^2),
    5 Sqrt[7/8] (-3/5 x + x^3), 105 (-1/5) + x^4 - 6 (-1/3 + x^2) / 7} / 2^{7/2}

In[3]:=
  gs[[1]]
Out[3]=
  1/Sqrt[2]

In[4]:=
  Integrate[gs[[1]] gs[[2]],{x,-1,1}]
Out[4]=
  0
  computes  $\int_{-1}^1 \frac{1}{\sqrt{2}} \sqrt{\frac{3}{2}} x dx.$ 

In[5]:=
  Sqrt[Integrate[gs[[1]]^2,{x,-1,1}]]
Out[5]=
  1
  computes  $\sqrt{\int_{-1}^1 \left(\frac{1}{\sqrt{2}}\right)^2 dx}.$ 

In[6]:=
  Integrate[gs[[1]] gs[[4]],{x,-1,1}]
Out[6]=
  0
  computes  $\int_{-1}^1 \frac{1}{\sqrt{2}} 5 \sqrt{\frac{7}{8}} \left(x^3 - \frac{3}{5}x\right) dx.$ 

In[7]:=
  Sqrt[Integrate[gs[[4]]^2,{x,-1,1}]]
Out[7]=
  1
  computes  $\sqrt{\int_{-1}^1 \left(5 \sqrt{\frac{7}{8}} \left(x^3 - \frac{3}{5}x\right)\right)^2 dx}.$ 

```


The polynomials which result are normalized in `lpoly` so that each member equals 1 at $x=1$. This yields a list of the first five Legendre polynomials and is accomplished by dividing each entry in `legendre5` with its value at $x=1$. In this case, the Legendre polynomials were the result of an example illustrating the `InnerProduct` option with `GramSchmidt`. Nevertheless, *Mathematica* contains the built-in command `LegendreP[n, x]` which gives these polynomial.

```
In[4]:=
```

```
lpoly=Table[Simplify[legendre5[[i]]/
  (legendre5[[i]]/.x->1)],{i,1,5}]
```

```
Out[4]=
```

```
{1, x,  $\frac{-1 + 3x^2}{2}$ ,  $\frac{x(-3 + 5x^2)}{2}$ ,  $\frac{3 - 30x^2 + 35x^4}{8}$ }
```

```
In[10]:=
```

```
Table[LegendreP[n, x], {n, 0, 4}]/TableForm
```

```
Out[10]//TableForm=
```

```
1
```

```
x
```

```
 $\frac{-1 + 3x^2}{2}$ 
```

```
 $\frac{-3x + 5x^3}{2}$ 
```

```
 $\frac{3 - 30x^2 + 35x^4}{8}$ 
```

A useful purpose for obtaining an orthogonal set of basis vectors is in approximating functions with polynomials. If a function $f(x)$ is to be approximated with a polynomial of degree n , then the closest polynomial is computed by projecting $f(x)$ onto each of the first n (or $n+1$) basis members of the space of n -th order polynomials. Hence, approximation can be conducted by using a basis made up of the Legendre polynomials created earlier. In order to illustrate this technique, the command **Projection**[*vec1*, *vec2*, *options*] which also appears in **Orthogonalization.m** must be discussed. This command projects *vec1* onto *vec2* and can employ the same option of **InnerProduct** as was shown earlier with **GramSchmidt**. The function **proj** is defined below to project a vector (the function) *v* onto the basis vectors in *basis* using the appropriate inner product for function spaces. This function is then demonstrated by projecting **Exp[x]** onto the orthogonal basis obtained earlier in **lpoly**. Hence, this calculation results in the approximation of **Exp[x]** using the first 5 Legendre polynomials. The approximating polynomial is then simplified and expressed with numerical coefficients in **app**. Both functions are plotted simultaneously to show the closeness of the approximation, and then the difference is plotted to better illustrate the accuracy of the approximation with Legendre polynomials.

```

In[64]:=
Clear[v,basis];
proj[v_,basis_]:=Sum[Projection[v,basis[[i]],
InnerProduct->
(Integrate[#1 #2,{x,-1,1}&)],
{i,1,Length[basis]}]

In[65]:=
proj[Exp[x],lpoly]

Out[65]=

$$\frac{-\frac{1}{E} + E}{2} + \frac{3x}{E} + \frac{5\left(\frac{-7}{E} + E\right)(-1 + 3x^2)}{4} +$$


$$\frac{7\left(\frac{37}{E} - 5E\right)x(-3 + 5x^2)}{4} +$$


$$\frac{9\left(\frac{-266}{E} + 36E\right)(3 - 30x^2 + 35x^4)}{16}$$

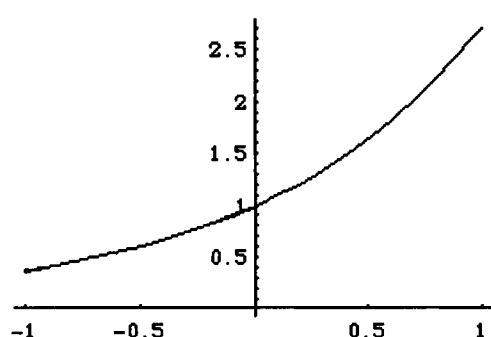

```

The approximating polynomial is then simplified and expressed with numerical coefficients in `app`.

```
In[66]:=
  app=Simplify[N[proj[Exp[x],lpoly]]]
Out[66]=
  1.00003 + 0.997955 x + 0.499352 x2 + 0.176139 x3 +
  0.0435974 x4
```

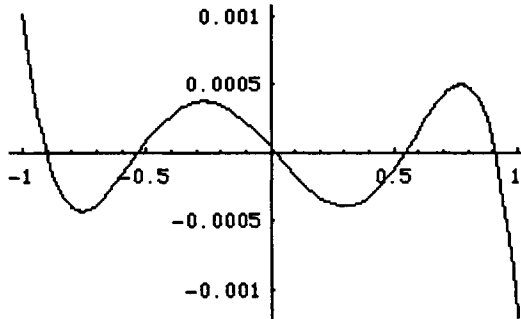
Both functions are plotted simultaneously to show the closeness of the approximation, and then the difference is plotted to better illustrate the accuracy of the approximation with Legendre polynomials.

```
In[67]:=
  Plot[{app,Exp[x]},{x,-1,1}]
Out[67]=
  -Graphics-
```



The graphs of `app` and `Exp[x]` are virtually identical.

```
In[68]:=
  Plot[app-Exp[x],{x,-1,1}]
Out[68]=
  -Graphics-
```



In fact, the difference between `Exp[x]` and `app` is reasonably small.

100%

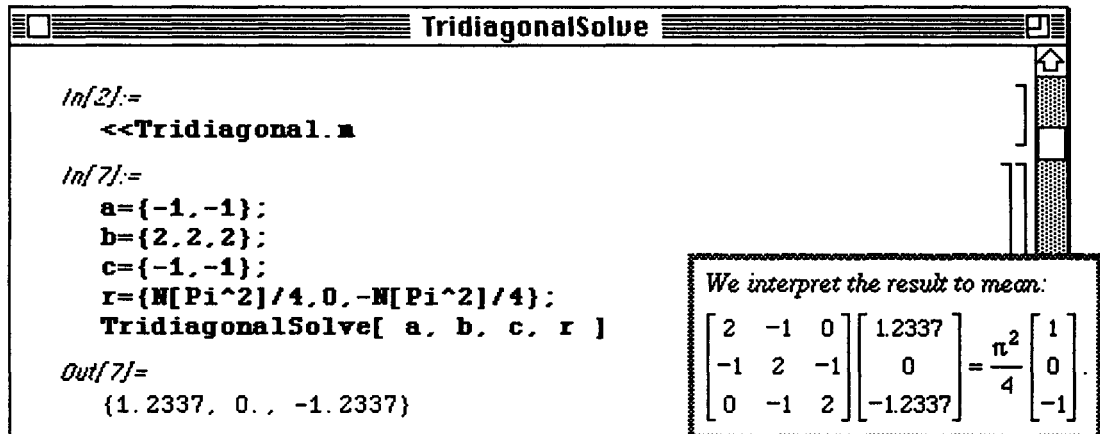
• Tridiagonal.m

Version 2.0's **Tridiagonal.m** package offers the command **TridiagonalSolve[a,b,c,r]** to solve the system of equations $\mathbf{Ax}=\mathbf{r}$ where \mathbf{A} is a tridiagonal matrix with diagonal \mathbf{b} , upper diagonal \mathbf{c} , and lower diagonal \mathbf{a} . Matrices of this type arise in many areas of applied mathematics and can be rather large in dimensions. Therefore, this command may be more useful than other commands available for solving systems of linear equations. In the first example below, the following system is solved:

○ Example:

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{\pi^2}{4} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

Notice that the solution of the system is given as the list $\{x_1, x_2, x_3\}$.



```

In[2]:=
  <<Tridiagonal.m

In[7]:=
  a={-1,-1};
  b={2,2,2};
  c={-1,-1};
  r={N[Pi^2]/4,0,-N[Pi^2]/4};
  TridiagonalSolve[ a, b, c, r ]

Out[7]=
  {1.2337, 0., -1.2337}

```

We interpret the result to mean:

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1.2337 \\ 0 \\ -1.2337 \end{bmatrix} = \frac{\pi^2}{4} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

□ Example:

As previously indicated, tridiagonal systems are common in many fields. One example of this is the numerical solution to differential equations. Consider the boundary value problem

$$-\frac{d^2u}{dx^2} = f(x), \quad 0 \leq x \leq 1$$

$$u(0) = 0, \quad u(1) = 0.$$

(This is the problem which describes the steady-state case of the heat equation with fixed end temperatures of zero degrees and heat source $f(x)$.) In order to solve this problem numerically, it must be changed from a continuous problem to a discrete one. This is accomplished first by providing a finite amount of information about f at the equally spaced points $x = h, x = 2h, \dots, x = nh$. An approximate solution is computed at these values of x . At the endpoints, $x = 0$ and $x = 1 = (n+1)h$, the solution must be zero from the boundary conditions. Hence, the approximate solutions at the endpoints are known to be

$$u_0 = 0 \text{ and } u_{n+1} = 0.$$

The derivatives in the differential equation are replaced by the difference quotients

$$\frac{du}{dx} = \frac{u(x+h) - u(x-h)}{2h}$$

and

$$\frac{d^2u}{dx^2} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}.$$

Hence, at a typical mesh point $x = jh$, the differential equation is replaced by the discrete analog

$$-u_{j+1} + 2u_j - u_{j-1} = h^2 f(jh).$$

There are n equations of the form, one for each value of $j = 1, \dots, n$. (Note that $h = 1/(n+1)$). Therefore, a numerical solution to this differential equation is obtained by solving the tridiagonal system which results from the discrete problem. The function `soln` which sets up and solves the system for any value of n and any function f is defined below. The output is given as a list of ordered pairs in which the first coordinate represents the meshpoint while the second coordinate gives the approximate solution at the corresponding meshpoint. In the example which follows, the boundary-value problem

$$-\frac{d^2u}{dx^2} = 4\pi^2 \sin(2\pi x), \quad 0 \leq x \leq 1$$

$$u(0) = 0, \quad u(1) = 0.$$

is solved. Of course, the exact solution to this problem is known to be $u(x) = \sin 2\pi x$. Hence, the approximate solutions found with this finite-difference method can be compared to the true solution.

The solution is approximated, first, for $n = 6$. The points given by `soln` are then plotted with `ListPlot` and shown simultaneously with the graph of the exact solution.

```

In[9]:=
Clear[a,b,c,r]
soln[f_,n_]:=
  Module[{h,a,b,c,r,vec,table},
    h=1/(n+1);
    a=Table[-1,{i,1,n-1}];
    b=Table[2,{i,1,n}];
    c=Table[-1,{i,1,n-1}];
    r=Table[h^2 N[f[i h]},{i,1,n}]/N;
    vec=TridiagonalSolve[a,b,c,r]/N;
    table=Table[{i h,vec[[i]]},{i,1,n}]
  ]

```

For the function `soln`, `h`, `a`, `b`, `c`, `r`, `vec` and `table` are defined to be local variables.

```

In[10]:=
f[x_]=4 Pi^2 Sin[2 Pi x]

```

For this example, define $f(x) = 4\pi^2 \sin(2\pi x)$.

```

Out[10]=
      2
4 Pi Sin[2 Pi x]

```

```

In[11]:=
soln[f,6]

```

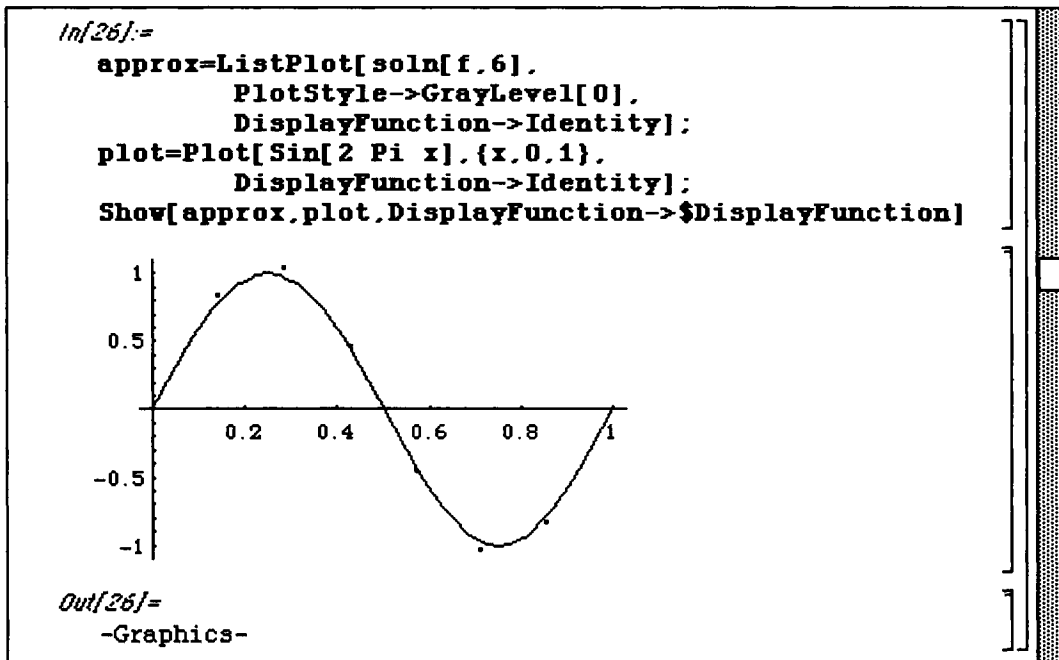
```

Out[11]=
      1           2           3           4
  {{-, 0.836508}, {-, 1.04311}, {-, 0.464227}, {-, -0.464227},
   7           7           7           7

      5           6
  {-, -1.04311}, {-, -0.836508}}
   7           7

```

The points given by `soln` are then plotted with `ListPlot` and shown simultaneously with the graph of the exact solution.



Similar steps are followed below with $n = 20$ and $n = 50$ to demonstrate the improved approximation as n increases.

```

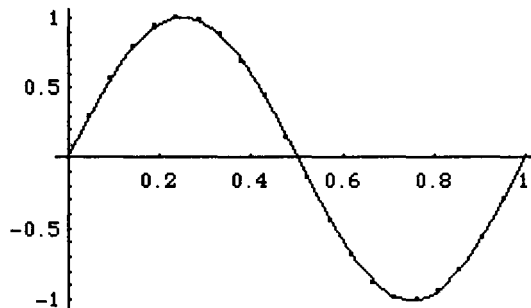
In[33]:=
sol=soln[f,20];
Short[sol,2]

Out[33]::Short=
      1          2          1
  {{-, 0.296964}, {-, 0.567541}, {-, 0.78769}, <<15>>,
      21          21          7

      19          20
  {{-, -0.567541}, {-, -0.296964}}
      21          21

In[36]:=
approx=ListPlot[sol,
PlotStyle->GrayLevel[0],
DisplayFunction->Identity];
plot=Plot[Sin[2 Pi x],{x,0,1},
DisplayFunction->Identity];
Show[approx,plot,DisplayFunction->${DisplayFunction}]

```



```

Out[36]=
-Graphics-

```


The calculations for $n=50$ are given below. Note the improvement of the approximation which results:

```

In[38]:=
  sol=soln[f,50];
  Short[sol,2]

Out[38]::Short=
  1           2           1
  {(-, 0.123044), (-, 0.244222), (-, 0.361699), <<45>>,
  51           51           17

  49           50
  {(-, -0.244222), (-, -0.123044)}
  51           51

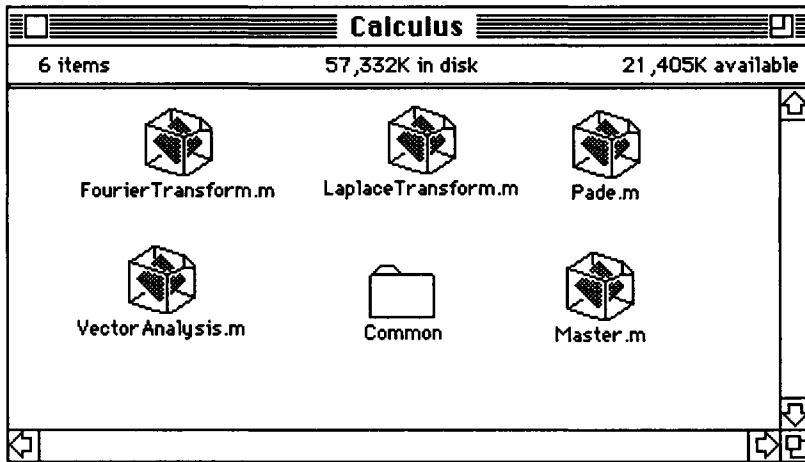
In[47]:=
  approx=ListPlot[sol,
    PlotStyle->GrayLevel[0],
    DisplayFunction->Identity];
  plot=Plot[Sin[2 Pi x],{x,0,1},
    DisplayFunction->Identity];
  Show[approx,plot,DisplayFunction->${DisplayFunction}]

Out[47]=
  -Graphics-

```

● 7.3 Version 2.0 Calculus

The packages contained in the Version 2.0 **Calculus** folder appear as follows:



The packages **FourierTransform.m** and **LaplaceTransform.m** are discussed here; **VectorAnalysis.m** was discussed in **Chapter 5**.

● LaplaceTransform.m

Commands which can be used to compute Laplace transforms and inverse Laplace transforms are located in the **LaplaceTransform.m** package. The command **InverseLaplaceTransform[f[s], s, t]** computes the inverse Laplace transform of $f[s]$ and the result is a function of t while **LaplaceTransform[g[t], t, s]** yields the Laplace transform of $g[t]$ as a function of s . Several examples are given below.

○ Example:

The package **LaplaceTransform.m** contains definitions of elementary Laplace transforms and their inverses.

Example 1:
In[158]:= InverseLaplaceTransform[1/s, s, t]
Out[158]=
 1
computes the inverse Laplace transform of $\frac{1}{s}$.

Example 2:
In[159]:= LaplaceTransform[t^4 Exp[a t], t, s]
Out[159]=
 24

 5
 (-a + s)
computes the Laplace transform of $t^4 e^{at}$, a constant.

Example 3:
In[160]:= LaplaceTransform[1/a Sin[a^2 t], t, s]
Out[160]=
 a

 4 2
 a + s
computes the Laplace transform of $\frac{1}{a} \sin(a^2 t)$, $a = a$ constant

Example 4:
In[161]:= InverseLaplaceTransform[1/(s^2+5)^2, s, t]
Out[161]=
 t (-Cos[Sqrt[5] t] + $\frac{\text{Sin}[\text{Sqrt}[5] t]}{\text{Sqrt}[5] t}$)

 10
computes the inverse Laplace transform of $\frac{1}{(s^2 + 5)^2}$.

Of course, an application of Laplace transforms is the solution of differential equations. In order to use these techniques, however, several important properties of Laplace transforms must be recalled. Two of these are investigated below with *Mathematica* and will be used later. Note how the derivative of the Laplace transform is represented in the second problem.

```

In[162]:=
Clear[x]
LaplaceTransform[x''[t], t, s]

```

computes the Laplace transform of $x''(t)$.

```

Out[162]=
2
s LaplaceTransform[x[t], t, s] - s x[0] -
x'[0]

```

```

In[163]:=
LaplaceTransform[t x'[t], t, s]

```

computes the Laplace transform of $t x'(t)$.

```

Out[163]=
-(LaplaceTransform[x[t], t, s] +
s LaplaceTransform(0,0,1)[x[t], t, s])

```

*is equivalent to
to the derivative of the Laplace
transform of $x(t)$:*

$$\frac{d}{ds} \text{LaplaceTransform}(x(t)).$$

◦ **Application:** Solutions of Ordinary Differential Equations

Consider the following second-order system of ordinary differential equations with initial conditions:

$$\begin{cases} x'' + 10x - 4y = 0, & x(0) = 0, & x'(0) = 3/2 \\ -4x + y'' + 4y = 0, & y(0) = 0, & y'(0) = -3/2 \end{cases}$$

Laplace transforms can be used to solve this system. After clearing all definitions, the Laplace transform of the first equation is found and called `leq1`. Similarly, the Laplace transform of the second equation is found and called `leq2`.

```

In[132]:=
Clear[x,y,t,leq1,leq2]
leq1=LaplaceTransform[
  x''[t]+10x[t]-4y[t]==0,t,s
]

Out[132]=
10 LaplaceTransform[x[t], t, s] +
  2
s LaplaceTransform[x[t], t, s] -
  4 LaplaceTransform[y[t], t, s] -
  s x[0] - x'[0] == 0

In[133]:=
leq2=LaplaceTransform[
  -4x[t]+y''[t]+4y[t]==0,t,s
]

Out[133]=
-4 LaplaceTransform[x[t], t, s] +
  4 LaplaceTransform[y[t], t, s] +
  2
s LaplaceTransform[y[t], t, s] -
  s y[0] - y'[0] == 0

```

In order to solve the system of differential equations
 $\begin{cases} x''(t) + 10x(t) - 4y(t) = 0 \\ -4x(t) + y''(t) + 4y(t) = 0 \end{cases}$, we begin by
 computing the Laplace transform of each equation. We name the Laplace transform of the equation $x'' + 10x - 4y = 0$ **leq1** and name the Laplace transform of the equation $-4x + y'' + 4y = 0$ **leq2**.

The usual convention when working with Laplace transforms is to use the capital letter X when referring to the Laplace transform of x. Hence, in the above calculations, `LaplaceTransform[x[t], t, s]` is replaced with `capx` and `LaplaceTransform[y[t], t, s]` is replaced with `capy`. Also the initial values $x[0]=0$, $x'[0]=3/2$, $y[0]=0$, and $y'[0]=-3/2$ can be substituted into the previous calculations.

These substitutions are made after defining the list of replacements, called **rule**, below. Then, **leq1/.rule** and **leq2/.rule** make all of the replacements given in **rule**. (**leq1/.rule** is an equation involving **capx** while **leq2/.rule** involves **capy**). Hence, the two equations which result are solved for **capx** and **capy**, respectively, in the **Solve** command. The result is named **solution**.

```
In[134]:=
rule={LaplaceTransform[x[t],t,s]->capx,
      x[0]->0,x'[0]->3/2,
      y[0]->0,y'[0]->-3/2,
      LaplaceTransform[y[t],t,s]->capy};
```

```
In[135]:=
solution=Solve[
  {leq1 /. rule,leq2 /. rule},
  {capx,capy}]
```

```
Out[135]=
{{capx ->
```

$$\frac{-24}{96 + 56 s^2 + 4 s^4} + \frac{3 (8 + 2 s^2)}{96 + 56 s^2 + 4 s^4}$$

. capy ->

$$\frac{24}{96 + 56 s^2 + 4 s^4} - \frac{3 (20 + 2 s^2)}{96 + 56 s^2 + 4 s^4}$$

We must solve **leq1** and **leq2** for **LaplaceTransform[x[t],t,s]** and **LaplaceTransform[y[t],t,s]** with the conditions that $x(0)=0$, $x'(0)=3/2$, $y(0)=0$, and $y'(0)=-3/2$. Hence, for convenience, define **rule** to substitute the appropriate values into **leq1** and **leq2**.

```
solution=Solve[
  {leq1 /. rule,
   leq2 /. rule},
  {capx,capy}]
```

first replaces the expressions **LaplaceTransform[x[t],t,s]**, **LaplaceTransform[y[t],t,s]**, **x[0]**, **x'[0]**, **y[0]**, and **y'[0]** by **capx**, **capy**, **0**, **3/2**, **0**, and **-3/2**, then solves the resulting system for **capx** and **capy**. The resulting list is named **solution**.

Note that `solution` is a list. Therefore, the formulas for `capx` and `capy` must be extracted. This is accomplished with `solution[[1,1,2]]` and `solution[[1,2,2]]`, respectively. This is shown below:

```
In[136]:=
  solution[[1,1,2]]
```

corresponds to the Laplace transform of $x(t)$.

```
Out[136]=
```

$$\frac{-24}{96 + 56 s^2 + 4 s^4} + \frac{3 (8 + 2 s^2)}{96 + 56 s^2 + 4 s^4}$$

```
In[137]:=
  solution[[1,2,2]]
```

corresponds to the Laplace transform of $y(t)$.

```
Out[137]=
```

$$\frac{24}{96 + 56 s^2 + 4 s^4} - \frac{3 (20 + 2 s^2)}{96 + 56 s^2 + 4 s^4}$$

The solution $(\mathbf{x}[t], \mathbf{y}[t])$ is now obtained through the inverse Laplace transform of the formulas of the Laplace transform of \mathbf{x} , `capx`, and that of \mathbf{y} , `copy`. These are determined below with the `InverseLaplaceTransform` command. Finally, the solution is graphed with `ParametricPlot` for values of t from $t = 0$ to $t = 2 \pi$.

```
In[138]:=
```

```
x[t_]=InverseLaplaceTransform[  
  solution[[1,1,2]],s,t]
```

```
Out[138]=
```

$$\frac{-3 \operatorname{Sin}[\operatorname{Sqrt}[2] t]}{10 \operatorname{Sqrt}[2]} + \frac{9 \operatorname{Sin}[2 \operatorname{Sqrt}[3] t]}{10 \operatorname{Sqrt}[3]}$$

```
In[139]:=
```

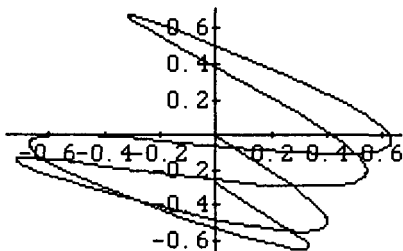
```
y[t_]=InverseLaplaceTransform[  
  solution[[1,2,2]],s,t]
```

```
Out[139]=
```

$$\frac{-3 \operatorname{Sin}[\operatorname{Sqrt}[2] t]}{5 \operatorname{Sqrt}[2]} - \frac{9 \operatorname{Sin}[2 \operatorname{Sqrt}[3] t]}{20 \operatorname{Sqrt}[3]}$$

```
In[141]:=
```

```
ParametricPlot[{x[t],y[t]},{t,0,2Pi}]
```



```
Out[141]=
```

```
-Graphics-
```

computes the inverse Laplace transform of `solution[[1,1,2]]` and names the resulting function (of t) `x[t]`.

computes the inverse Laplace transform of `solution[[1,2,2]]` and names the resulting function (of t) `y[t]`.

graphs $\{x(t), y(t)\}$ for $0 \leq t \leq 2\pi$.

● **FourierTransform.m**

The **FourierTransform.m** package contains two commands useful for the computation of Fourier series. The first of these is **FourierTrigSeries** [**f** [**x**], {**x**, -**L**, **L**}, **n**] which computes the first **n** terms (the **n**th partial sum) of the Fourier series of the periodic extension of **f** [**x**] on the interval {-**L**, **L**}. This series is given by the formula

$$a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right)$$

where

$$a_0 = \frac{1}{2L} \int_{-L}^L f(x) dx$$

$$a_n = \frac{1}{L} \int_{-L}^L f(x) \cos \frac{n\pi x}{L} dx, n = 1, 2, \dots$$

$$b_n = \frac{1}{L} \int_{-L}^L f(x) \sin \frac{n\pi x}{L} dx, n = 1, 2, \dots$$

Since these integrals are rather complicated in most cases, the **FourierTrigSeries** command is quite useful. This package also includes the command **NFourierTrigSeries** which gives the Fourier series in terms of numerically approximated Fourier series coefficients. These two commands are illustrated below.

□ Example:

First, the fourth partial sum of the Fourier series for the periodic extension of **f** [**x**] = **x** is computed with **FourierTrigSeries** and named **serexact**. Next, the same is given numerically for the function **f** [**x**] = **Abs** [**x**]. This result is called **serapprox**.

```
In[8]:=
serexact=FourierTrigSeries[x, {x, -Pi, Pi}, 4]
```

```
Out[8]=
2 Sin[x] - Sin[2 x] +  $\frac{2 \text{Sin}[3 x]}{3}$  -  $\frac{\text{Sin}[4 x]}{2}$ 
```

serexact is the fourth partial sum of the Fourier series for the periodic extension of $f(x)=x$ on the interval $[-\pi, \pi]$.

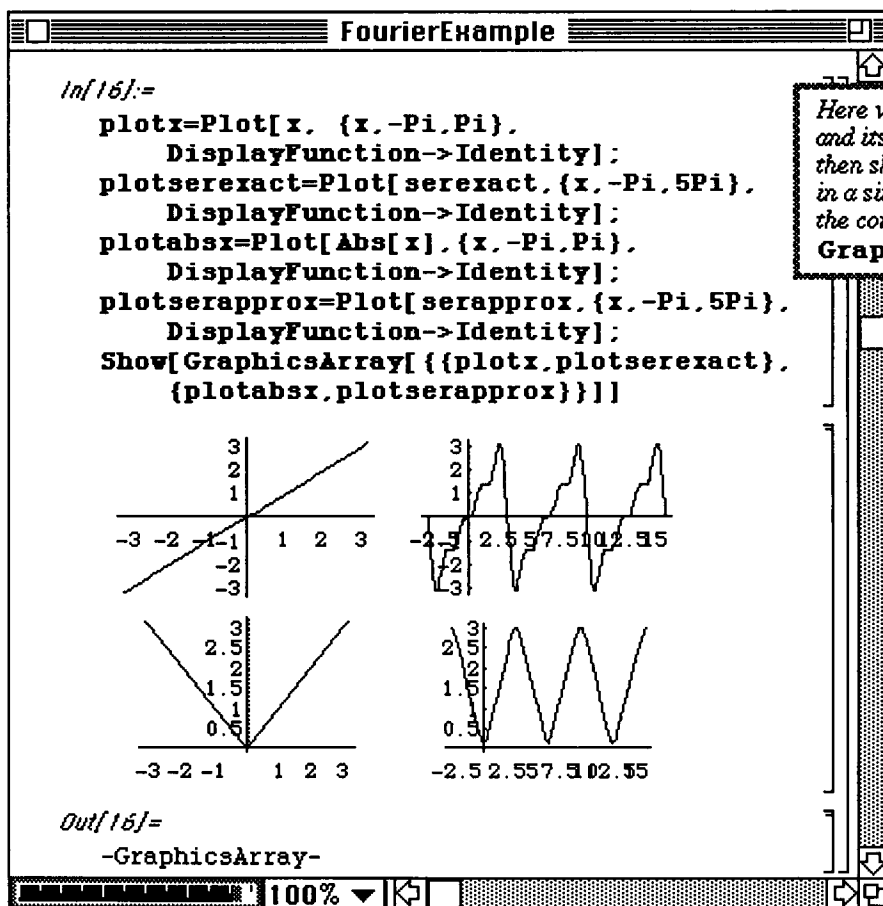
```
In[10]:=
serapprox=NFourierTrigSeries[Abs[x], {x, -Pi, Pi}, 4]
```

```
Out[10]=
0. +  $\frac{4.9348}{\text{Pi}}$  -  $\frac{4.00001 \text{Cos}[x]}{\text{Pi}}$  -
 $\frac{0.00000659848 \text{Cos}[2 x]}{\text{Pi}}$  -  $\frac{0.444451 \text{Cos}[3 x]}{\text{Pi}}$  -
 $\frac{0.00000659835 \text{Cos}[4 x]}{\text{Pi}}$ 
```

serapprox is an approximation of the fourth partial sum of the Fourier series for the periodic extension of $f(x)=|x|$ on the interval $[-\pi, \pi]$.

The graphs of $f[x] = x$ and $f[x] = \text{Abs}[x]$ along with the fourth partial sums of the corresponding Fourier series, `serexact` and `serapprox`, can be displayed in the same graphics cell. This is accomplished in the single command below. Recall that the `Plot` option `DisplayFunction->Identity` causes the graph to be suppressed initially. Then, the command

`Show[GraphicsArray[{{plotx, plotserexact}, {plotabsx, plotserapprox}}]]` displays all four graphs in one cell. Also, because of the grouping within `GraphicsArray`, the plots are displayed in the appropriate order. The Fourier series approximation of each function can easily be viewed in this manner.



□ Example:

The Fourier series of piecewise-defined functions can also be computed. This is illustrated with the function $f[x]$ below. After making the correct definition of this function, a table called **approx**s which consists of the second, sixth, and tenth partial sums of the Fourier series for the periodic extension of f is constructed with **MFourierSeries**. (A shortened output for **approx**s is given below.) Since **approx**s is a list of three elements, the second partial sum is given by **approx**s[[1]], the sixth by **approx**s[[2]], and the tenth by **approx**s[[3]].

FourierExamples

```
In[19]:=
Clear[f]
f[x_]:=1 /; 0 <=x <=1
f[x_]:=-x /; -1 <= x < 0

In[20]:=
approx=Table[
  MFourierTrigSeries[f[x],{x,-1,1},n],
  {n,2,10,4}];

In[30]:=
Short[approx,8]

Out[30]//Short=
{0.750409 - 0.201825 Cos[Pi x] +
  0.000817324 Cos[2 Pi x] +
  0.318309 Sin[Pi x] + 0.159153 Sin[2 Pi x],
  0.750409 + <<12>>,
  0.750409 - 0.201825 Cos[Pi x] +
  0.000817324 Cos[2 Pi x] + <<16>> +
  0.0353583 Sin[9 Pi x] +
  0.0318205 Sin[10 Pi x]}
```

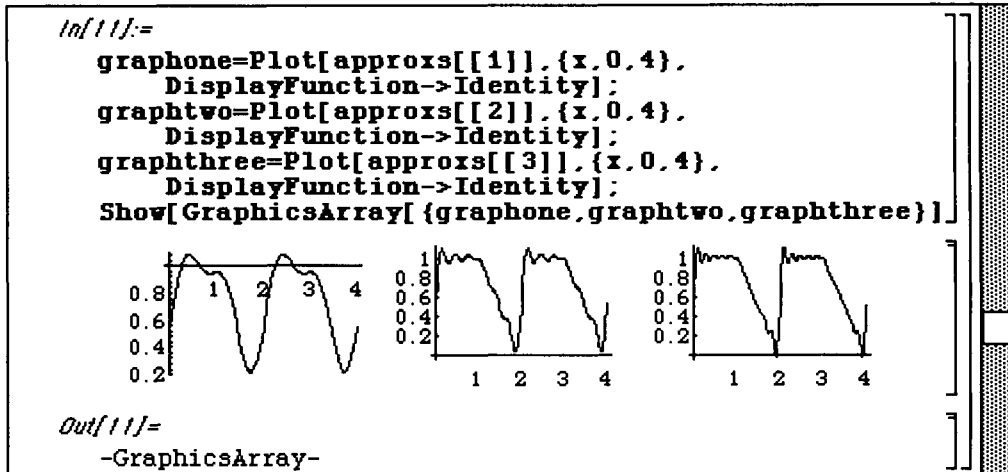
After clearing all prior definitions of f , we define

$$f(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ -x & \text{if } -1 \leq x < 0 \end{cases}$$

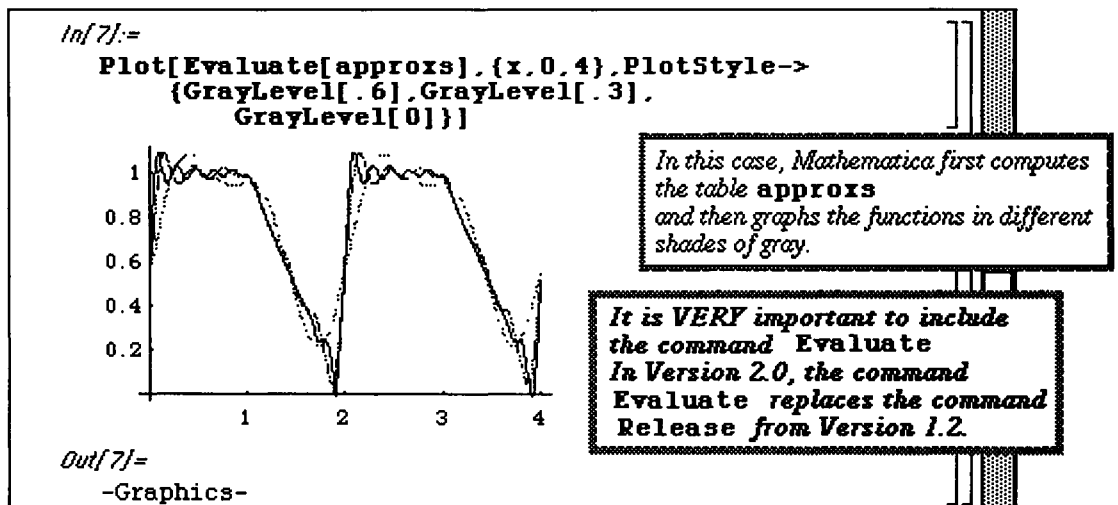
computes a table of approximations of the second, sixth, and tenth partial sums of the Fourier series for f . The resulting table is named **approx**s.

Short[approx, 8] prints an abbreviated form of **approx** on no more than eight lines.

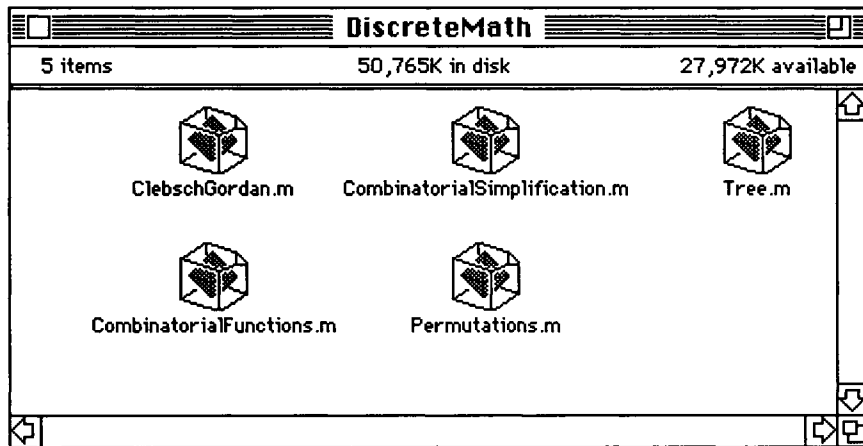
The accuracy of the approximation of $f(x)$ by the partial sums of the Fourier series can be seen by plotting `approxs[[1]]`, `approxs[[2]]`, and `approxs[[3]]`. The graphs below clearly show that the approximation becomes more accurate (the corresponding curves move closer to the function ϵ) as more terms of the Fourier series are used.



- o These three graphs can also be plotted simultaneously in the single command below. Note that `Evaluate[approxs]` must be used in the `Plot` command just as `Release` was used in Version 1.2. The three are given in varying `GrayLevel` with the tenth partial sum represented by the darkest curve.



■ 7.4 Discrete Math



The DiscreteMath folder is contained in the folder Packages.

■ CombinatorialFunctions.m

In addition to the built-in *Mathematica* functions **Binomial**[*m*, *n*] and **Multinomial**[*n*₁, *n*₂, *n*₃, ..., *n*_{*m*}], the package **CombinatorialFunctions.m**, included with both Version 1.2 and Version 2.0, provides commands which deal with subfactorials, Catalan and Fibonacci numbers as well as the Hofstadter functions. We first discuss the built-in functions.

Binomial[*n*, *m*] gives the binomial coefficient $\binom{n}{m} = \frac{n!}{m!(n-m)!}$. (The number of ways to choose *m* objects from a collection of *n* objects, without consideration to order.)

Multinomial[*n*₁, *n*₂, ..., *n*_{*m*}] gives the multinomial coefficient $\frac{N!}{n_1! n_2! \cdots n_m!}$, where

$\sum_{i=1}^m n_i = N$. Hence, the multinomial coefficient gives the number of ways to partition *N* distinct

objects into *m* sets, each having size *n*_{*i*}.

Several examples of these functions are given below prior to loading the package :

<code>In[1]:=</code>	<code>Binomial[3,3]</code>	<code>Binomial[3,3] computes</code>	$\binom{3}{3} = \frac{3!}{3!} = 1.$
<code>Out[1]=</code>	1		
<code>In[2]:=</code>	<code>Binomial[3,2]</code>	<code>Binomial[3,2] computes</code>	$\binom{3}{2} = \frac{3!}{(3-2)!2!}.$
<code>Out[2]=</code>	3		
<code>In[3]:=</code>	<code>Binomial[10,5]</code>	<code>Binomial[10,5] computes</code>	$\binom{10}{5} = \frac{10!}{(10-5)!5!}.$
<code>Out[3]=</code>	252		
<code>In[4]:=</code>	<code>Multinomial[2,4,7]</code>	<code>Multinomial[2,4,7] computes</code>	$\frac{(2+4+7)!}{2!4!7!}.$
<code>Out[4]=</code>	25740		
<code>In[5]:=</code>	<code>Multinomial[3,2,9,15,43]</code>	<code>Multinomial[3,2,9,15,43]</code>	<code>computes</code>
<code>Out[5]=</code>	177993982379513913738178749696000		$\frac{(3+2+9+15+43)!}{3!2!9!15!43!}.$

Typical problems which involve determining the binomial and multinomial coefficients are as follows :

□ Example:

In how many ways can an unordered 5-card poker hand be selected from a deck of 52 cards ?

Solution :

The answer is $\binom{52}{5}$ and is easily computed by entering `Binomial[52,5]`.

□ Example:

How many strings can be formed from the letters in *Mathematica* ?

Solution : The solution is determined using the multinomial coefficient. In the word *Mathematica*, the letter A appears 3 times while the letters M and T each appear twice. All others appear only once and there are 11 total letters. Therefore, the correct value is found with `Multinomial[3,2,2,1,1,1,1]` . (The 1's must not be omitted because the command uses them to determine the numerator in the calculation.)

Answers to both examples are computed below:

The screenshot shows a Mathematica notebook window titled "CombinatorialFunctions". The notebook content is as follows:

```
In[6]:=
  Binomial[52,5]
Out[6]=
  2598960

In[7]:=
  Multinomial[3,2,2,1,1,1,1]
Out[7]=
  1663200
```

To the right of the notebook window, the following mathematical formulas are displayed:

`Binomial[52,5]` calculates $\binom{52}{5} = \frac{52!}{(52-5)!5!}$.

`Multinomial[3,2,2,1,1,1,1]` calculates $\frac{(3+2+2+1+1+1+1)!}{3!2!2!}$.

□ Example:

A useful combinatorial function which can be used only after loading `CombinatorialFunctions.m`, is `Subfactorial[n]`. This gives the number of permutations of n objects which leave no object fixed and is determined using the formula $n! \text{ Sum}[(-1)^k/k!, \{k, 0, n\}]$.

After loading `CombinatorialFunctions.m`, several examples are given. `Subfactorial[4]` is also found using the formula $4! \text{ Sum}[(-1)^k/k!, \{k, 0, 4\}]$ to verify that the same value results from each command.

The screenshot shows a Mathematica notebook titled "CombinatorialFunctions" with the following content:

```

In[9]:=
  <<CombinatorialFunctions.m

In[9]:=
  Subfactorial[2]

Out[9]=
  1

In[10]:=
  Subfactorial[4]

Out[10]=
  9

In[11]:=
  4! Sum[(-1)^k/k!, {k, 0, 4}]

Out[11]=
  9

In[12]:=
  Subfactorial[10]

Out[12]=
  1334961

```

To the right of the notebook, the following text is displayed:

The definition of **Subfactorial** is contained in the package **CombinatorialFunctions.m**

Subfactorial[2] computes

$$2! \sum_{k=0}^2 \frac{(-1)^k}{k!}$$

Subfactorial[4] computes

$$4! \sum_{k=0}^4 \frac{(-1)^k}{k!}$$

4! Sum[(-1)^k/k!, {k, 0, 4}] also computes **Subfactorial[4]**.

Subfactorial[10] computes

$$10! \sum_{k=0}^{10} \frac{(-1)^k}{k!}$$

Another function found in `CombinatorialFunctions.m` determines the Catalan numbers. These numbers are found with the formula $\text{Binomial}[2n, n] / (n+1)$ and are named after the Belgian mathematician Eugene-Charles Catalan (1814-1894) who discovered an elementary derivation of the formula. The problem which led to this discovery was that of determining the number of ways to divide a convex $(n+2)$ -sided polygon, n greater than or equal to 1, into triangles by drawing $(n-1)$ lines through the corners that do not intersect in the interior of the polygon. This number is found with `CatalanNumber[n]` which gives the n th Catalan number.

□ Example:

For example, there are five ways to divide a convex pentagon ($n=3$) into triangles by drawing two nonintersecting lined through the corners. This is given with `CatalanNumber[3]`. A table of Catalan numbers can also be created as illustrated below.

```

In[13]:=
  CatalanNumber[3]
Out[13]=
  5

In[14]:=
  Table[CatalanNumber[i], {i, 1, 5}] // TableForm
Out[14] // TableForm =
  1
  2
  5
  14
  42

```

computes $\frac{\binom{6}{3}}{4}$

computes $\frac{\binom{2i}{i}}{i+1}$
for $i= 1, 2, 3, 4, 5$ and
expresses the result in
TableForm.

Another useful sequence is that of the Fibonacci numbers named for the Italian merchant and mathematician, Leonardo Fibonacci (1170-1250). These numbers are encountered in many areas as well and first arose in a puzzle about rabbits. One interesting fact about this sequence involves the ratio of successive terms. The limiting value of this ratio is:

$$\text{GoldenRatio} - 1 = \frac{1 + \sqrt{5}}{2} - 1 = \frac{\sqrt{5} - 1}{2} \approx 0.61803.$$

□ Example:

This is demonstrated below. First, a table of the first 16 Fibonacci numbers is created with **Table** and **Fibonacci[n]** which yields the *n*th Fibonacci number. This is called **table1**. Then, a table of the ratios of successive Fibonacci numbers is computed using **table1** to illustrate that the limit is the Golden Mean-1. A numerical approximation of the Golden Mean is also computed for comparison. Note that the first Fibonacci number is found with index, *n* = 0. Thus, the *n*th Fibonacci number is found with **Fibonacci[n-1]**.

```

In[18]:=
  Fibonacci[0]
Out[18]=
  1
In[19]:=
  tablefib=Table[Fibonacci[i],{i,0,15}]
Out[19]=
  {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,
   377, 610, 987}
In[20]:=
  ratio=Table[tablefib[[i]]/tablefib[[i+1]],
    {i,1,15}] // N
Out[20]=
  {1., 0.5, 0.666667, 0.6, 0.625, 0.615385, 0.619048,
   0.617647, 0.618182, 0.617978, 0.618056, 0.618026,
   0.618037, 0.618033, 0.618034}

```

Fibonacci[0] and Fibonacci[1]
are both defined to be zero. The *n*th Fibonacci number is obtained by adding together the (*n*-1)st and (*n*-2)nd Fibonacci numbers.

computes a table of the first sixteen Fibonacci numbers. The resulting table is named **tablefib**.

computes the ratio of successive Fibonacci numbers.

□ Example:

It should be noted that calculation of **Fibonacci[n]** for *n* > 10 is quite slow. Hence, an alternate approach is suggested. A table of the first 250 Fibonacci numbers can easily be found as follows (a shortened table of five lines is requested to save space).

Mathematica is designed to allow it to "remember" computed function values. In general, functions defined in the form

f[x_]:=f[x]=function definition will retain all values computed for later use.

The calculation of the first 250 Fibonacci numbers required approximately 1200 seconds in computation time with `Fibonacci[n]`, but the output was practically instantaneous when using the commands below.

```

In[23]:=
fib[0]=1
fib[1]=1
fib[n_]:=fib[n]=fib[n-1]+fib[n-2]
table=Table[fib[i],{i,0,250}]
Short[table,5]

Out[23]//Short=
{1, 1, 2, 3, 5, <<242>>, 30161<<43>>0501,
 4880197746793002076754294951020699004973\
 287771475874, 7896325826131730509282738\
 943634332893686268675876375}

Time: 12.77 seconds

```

`fib[n_]:=fib[n]=`
tells Mathematica to remember
the values of `fib[n]`
that it computes.

A table containing the values `fib[50]`, `fib[100]`, `fib[150]`, and `fib[200]` is given below.

Since `fib[50]`, `fib[100]`, `fib[150]`, and `fib[200]` were computed above, we note that recalling them is nearly instantaneous. Using the function `Fibonacci[n]` to compute `Fibonacci[50]`, `Fibonacci[100]`, `Fibonacci[150]`, and `Fibonacci[200]` takes considerable time.

```

In[25]:=
Table[fib[50 n],{n,1,4}] // TableForm

Out[25]//TableForm=
12586269025

354224848179261915075

9969216677189303386214405760200

280571172992510140037611932413038677189525

Time: 0.42 seconds

```

Since Mathematica has already
computed `fib[50]`, `fib[100]`,
`fib[150]`, and `fib[200]`,
representing them is almost immediate.
Note that for Mathematica to compute
`Fibonacci[200]`
takes considerable time.

□ **Application:**

Some interesting facts concerning Fibonacci numbers can now be investigated.

Consider the sequence $\{T_n\}_{n=1}^{\infty}$ where $T_n = \frac{F_{5n}}{5F_n}$, F_n being the n th Fibonacci number.

Theorem: If T_n is prime, then n is either prime or a power of 5.

(Another interesting fact about this sequence is that all numbers end in 1.) By computing the following table with the sequence of Fibonacci numbers in `fib[n]` found above, these properties can be investigated.

```

CombinatorialFunctions

In[13]:=
  ratio=Table[ fib[5 i]/(5 fib[i]),{i,1,25}]
Out[13]=
{1, 11, 61, 451, 3001, 20801, 141961, 974611,
 6675901, 45768251, 313671601, 2150012161,
14736206161, 101003973851, 692290189501,
4745031073651, 32522917584361,
222915417520961, 1527884938291801,
10472279325329251, 71778069881360701,
491974211042344811, 3372041404278257761,
23112315627117696001, 158414167964045700001}

Time: 0.85 seconds

```

Since we have already computed fib[i] for i=0, ..., 250, the ratios are quickly computed. Every number in the table ratio is prime.

The **Hofstadter function**, Hof, is recursively defined by

- (i) Hof(1)=Hof(2)=1; and
- (ii) Hof(n)=Hof[n-Hof(n-1)]+Hof(n-Hof(n-2))

Mathematica's definition of the Hofstadter function is contained in the package `CombinatorialFunctions.m` and is denoted by `Hofstadter[n]`.

□ Example:

A table of the first thirty values of the Hofstadter function is given below :

```

CombinatorialFunctions
In[4]:=
  <<CombinatorialFunctions.m
In[5]:=
  table2=Table[Hofstadter[i],{i,1,30}]
Out[5]=
  {1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8,
   8, 8, 10, 9, 10, 11, 11, 12, 12, 12,
   12, 16, 14, 14, 16, 16, 16, 16}
  
```

table2 is a table of Hof(i) for i= 1, 2, ..., 30.

■ CombinatorialSimplification.m

The package **CombinatorialSimplification.m** includes several rules which can be used to simplify certain expressions which *Mathematica* does not automatically simplify. For specific integers m and n , $n!$

computes the factorial of n ; **Binomial** $[m, n]$ computes $\binom{m}{n} = \frac{m!}{(m-n)!n!}$.

Unfortunately, if m and n are not specific integers, *Mathematica* does not automatically simplify the following expressions involving factorials and binomial coefficients:

```

CombinatorialSimplification

In[1]:=
  (n+1)!/n!
Out[1]=
  (1 + n)!
  -----
  n!

In[2]:=
  Binomial[n, k+1]/Binomial[n, k]
Out[2]=
  Binomial[n, 1 + k]
  -----
  Binomial[n, k]

```

For an integer n , $\frac{(n+1)!}{n!} = n+1$.

Nevertheless, *Mathematica* does not complete the simplification.

Binomial $[m, n]$ produces the binomial coefficient

$$\binom{m}{n} = \frac{m!}{(m-n)!n!}$$

□ Example:

Fortunately, these expressions can be simplified with after loading the package

CombinatorialSimplification.m. since **CombinatorialSimplification.m** redefines the built-in combinatorial functions so that *Mathematica* symbolically reduces combinatorial functions:

```

CombinatorialSimplification

In[12]:=
  <<CombinatorialSimplification.m

In[13]:=
  (n+1)!/n!

Out[13]=
  1 + n

In[14]:=
  (2n+1)!/(2n)!

Out[14]=
  1 + 2 n

In[15]:=
  (2n+5)!/(2n)!

Out[15]=
  (1 + 2 n) (2 + 2 n) (3 + 2 n)
  (4 + 2 n) (5 + 2 n)
  
```

After loading the package **CombinatorialSimplification.m** the *Mathematica* assumes that *n* is an integer and correctly simplifies the expressions $\frac{(n+1)!}{n!}$, $\frac{(2n+1)!}{(2n)!}$, and $\frac{(2n+5)!}{(2n)!}$.

Binomial[n,m] is also redefined for the quotients **Binomial[n,k]/Binomial[n,k-1]**:

```

In[21]:=
  Binomial[n,3]/Binomial[n,2]
Out[21]=
  -2 + n
  -----
      3

In[22]:=
  Binomial[n,k+1]/Binomial[n,k]
Out[22]=
  -k + n
  -----
  1 + k

In[23]:=
  Binomial[n,k]/Binomial[n,k+1]
Out[23]=
  1 + k
  -----
  -k + n

In[24]:=
  Binomial[4,k+1]/Binomial[4,k]
Out[24]=
  4 - k
  -----
  1 + k

```

Similarly, Mathematica assumes that k is an integer and simplifies

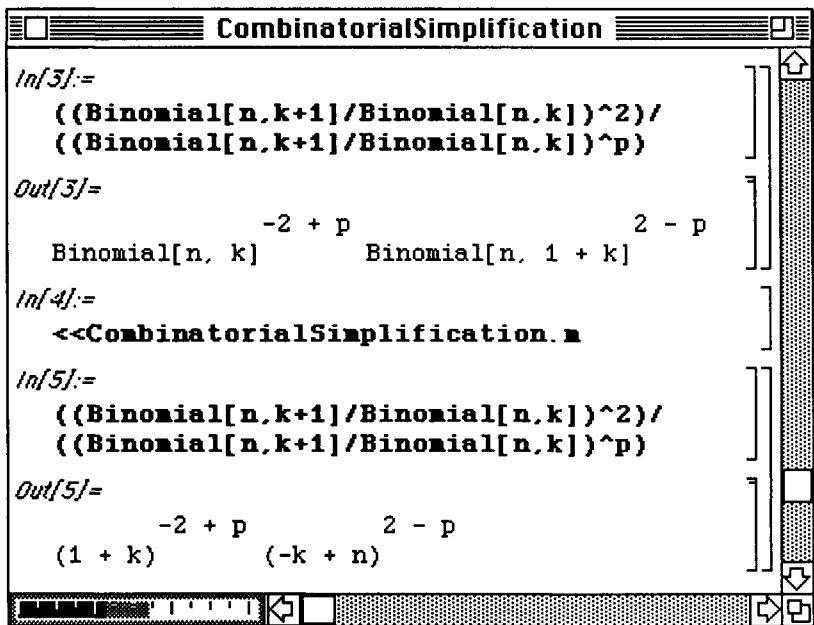
$$\frac{\binom{n}{3}}{\binom{n}{2}}, \frac{\binom{n}{k+1}}{\binom{n}{k}}, \frac{\binom{n}{k}}{\binom{n}{k+1}}, \text{ and } \frac{\binom{4}{k+1}}{\binom{4}{k}}$$

`((Binomial[n, k+1]/Binomial[n, k])^2) / ((Binomial[n, k+1]/Binomial[n, k])^p)` denotes the expression

$$\frac{\left(\frac{\binom{n}{k+1}}{\binom{n}{k}}\right)^2}{\left(\frac{\binom{n}{k+1}}{\binom{n}{k}}\right)^p}$$

Mathematica is unable to simplify this expression until after the package

has been loaded:



Upon closing the `CombinatorialSimplification.m` package, all new definitions are cleared. Hence, any built-in function which was redefined is unaltered by the changes made in the package.

■ Permutations.m

A **permutation** of n distinct elements x_1, x_2, \dots, x_n is an ordering of the n elements.

Permutations.m gives several commands in addition to the built-in commands which are helpful in working with permutations. *Mathematica* already includes the function **Permutation[list]** which gives all possible permutations of the list of n elements **list**. This package includes **PermutationQ[list]** which gives a value of **True** if **list** is a possible permutation of n distinct elements and **False** otherwise. Therefore, if any of the n elements is omitted or repeated in **list**, a value of **False** is given. Several examples that illustrate these ideas are given below. These calculations can be made after loading the package :

The screenshot shows a Mathematica notebook window titled "Permutations". The notebook contains several input-output pairs. To the right of the notebook, there are explanatory text blocks for each input-output pair.

Input	Output	Explanation
<code>In[8]:= perm=Permutations[{1,2,3}]</code>	<code>Out[8]=</code> { {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1} }	perm is a list consisting of all permutations of {1,2,3}.
<code>In[9]:= <<Permutations.m</code>		
<code>In[10]:= PermutationQ[{1,3,2}]</code>	<code>Out[10]= True</code>	PermutationQ[{1,3,2}] is True since {1,3,2} is a permutation of {1,2,3}.
<code>In[11]:= PermutationQ[{3,2,1}]</code>	<code>Out[11]= True</code>	PermutationQ[{3,2,1}] is True since {3,2,1} is a permutation of {1,2,3}.
<code>In[12]:= PermutationQ[{3,2,2}]</code>	<code>Out[12]= False</code>	PermutationQ[{3,2,2}] is False since {3,2,2} is not a permutation of {1,2,3}.
<code>In[13]:= PermutationQ[{3,4,2}]</code>	<code>Out[13]= False</code>	PermutationQ[{3,4,2}] is False since {3,4,2} is not a permutation of {1,2,3,4}.

The package also includes the command `RandomPermutation[n]` which yields one possible permutation of the n elements in the list $\{1, 2, 3, \dots, n\}$. This permutation is selected at random. Notice in the example which follows that two different permutations are given with `RandomPermutation[15]`. Two other commands found in the package are `ToCycles[permutation]` which gives `permutation` as a list of cyclic permutations and `FromCycles[cycles]` which returns `permutation` to its original form.

Notice in the example which follows that two different permutations are given with `RandomPermutation[15]`.

Permutations	
<code>In[30]:=</code> <code>random1=RandomPermutation[15]</code>	random1 <i>is a random permutation of</i> <i>{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}.</i>
<code>Out[30]=</code> {13, 12, 8, 10, 2, 11, 14, 1, 6, 15, 5, 3, 9, 7, 4}	
<code>In[31]:=</code> <code>random2=RandomPermutation[15]</code>	random2 <i>is a random permutation of</i> <i>{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}.</i>
<code>Out[31]=</code> {7, 12, 15, 11, 1, 14, 4, 5, 3, 9, 6, 2, 10, 8, 13}	
<code>In[32]:=</code> <code>cycle1=ToCycles[random1]</code>	ToCycles[random1] <i>writes random1</i> <i>as a product of disjoint cycles.</i>
<code>Out[32]=</code> {{13, 9, 6, 11, 5, 2, 12, 3, 8, 1}, {10, 15, 4}, {14, 7}}	

<pre>In[33]:= original1=FromCycles[cycle1] Out[33]= {13, 12, 8, 10, 2, 11, 14, 1, 6, 15, 5, 3, 9, 7, 4}</pre>	<p>FromCycles[cycle1] writes cycle1 as a permutation.</p>
<pre>In[34]:= cycle2=ToCycles[random2] Out[34]= {{7, 4, 11, 6, 14, 8, 5, 1}, {12, 2}, {15, 13, 10, 9, 3}}</pre>	<p>ToCycles[random2] writes the permutation random2 as a product of disjoint cycles.</p>
<pre>In[35]:= original2=FromCycles[cycle2] Out[35]= {7, 12, 15, 11, 1, 14, 4, 5, 3, 9, 6, 2, 10, 8, 13}</pre>	<p>FromCycles[cycle2] writes cycle2 as a permutation.</p>
<pre>In[36]:= random2=RandomPermutation[7] Out[36]= {3, 4, 5, 1, 2, 7, 6}</pre>	<p>random2 is a random permutation of {1,2,3,4,5,6,7}.</p>

Permutations	
<pre>In[37]:= cycle2=ToCycles[random2] Out[37]= {{3, 5, 2, 4, 1}, {7, 6}}</pre>	<p>ToCycles[random2] writes random2 as a product of disjoint cycles.</p>
<pre>In[38]:= original2=FromCycles[cycle2] Out[38]= {3, 4, 5, 1, 2, 7, 6}</pre>	<p>FromCycles[cycle2] writes cycle2 as a permutation.</p>

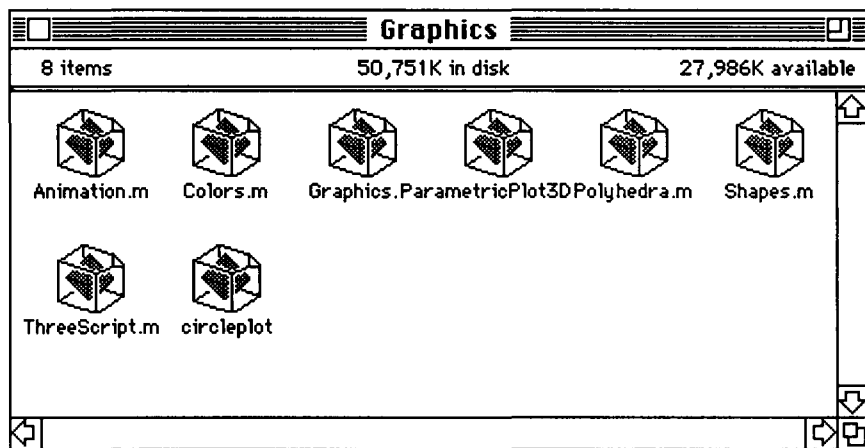
Chapter 8

Some Graphics Packages

- **Chapter 8** introduces several of the graphics packages available with *Mathematica*. Differences between Version 1.2 and Version 2.0 are discussed where appropriate.

■ Graphics

Opening the Version 1.2 **Graphics** folder yields the following window. Each package shown below contains one or more *Mathematica* commands or functions which cannot be used without loading the appropriate package.



The Graphics folder is contained in the folder Packages. This section describes some uses of several of the available packages.

■ 8.1 Graphics.m

Loading **Graphics.m** enables the user to take advantage of several commands which will improve the graphing capabilities previously available. The first command discussed below, **PolarPlot**, allows for the graphing of functions given in polar coordinates (r, θ) . This command should be entered in the following manner: **PolarPlot** [**function** [**var**], {**var**, **var1**, **var2**}, **options**] where **var** represents the angular coordinate θ and **var** varies from **var1** to **var2**. This command produces the graph of the function **r=function** [**var**].

□ Example:

In the first example below, the graph of $r = 1 - 2 \sin \theta$ is given. It is followed by the graph of $r = 1 + 2 \cos \theta$ which is plotted with the **GrayLevel** option. Notice that the graphs are named **polarone** and **polartwo**, respectively, for later use.

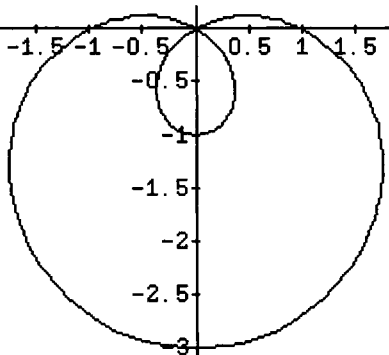
MoreGraphics

```

In[1]:=
<<Graphics.m

In[2]:=
polarone=PolarPlot[1-2Sin[theta],{theta,0,2Pi}]

```



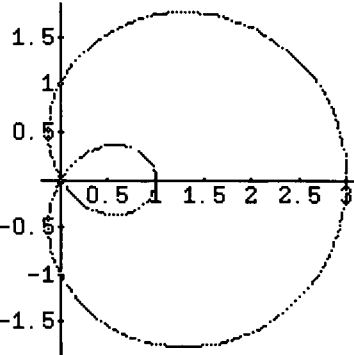
*creates a polar graph of the function $1 - 2\sin(\theta)$, $0 \leq \theta \leq 2\pi$. The resulting graph is named **polarone**.*

```

Out[2]=
-Graphics-

In[3]:=
polartwo=PolarPlot[1+2Cos[theta],{theta,0,2Pi},
PlotStyle->GrayLevel[.3]]

```



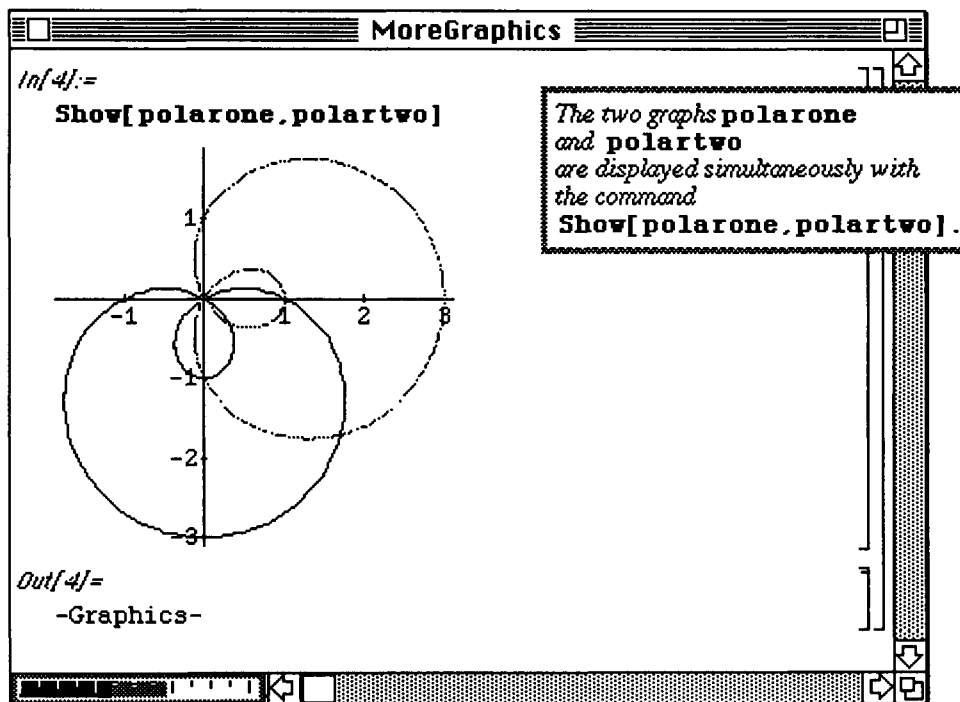
*creates a polar graph of the function $1 + 2\cos(\theta)$, $0 \leq \theta \leq 2\pi$. The resulting graph is named **polartwo**.*

```

Out[3]=
-Graphics-

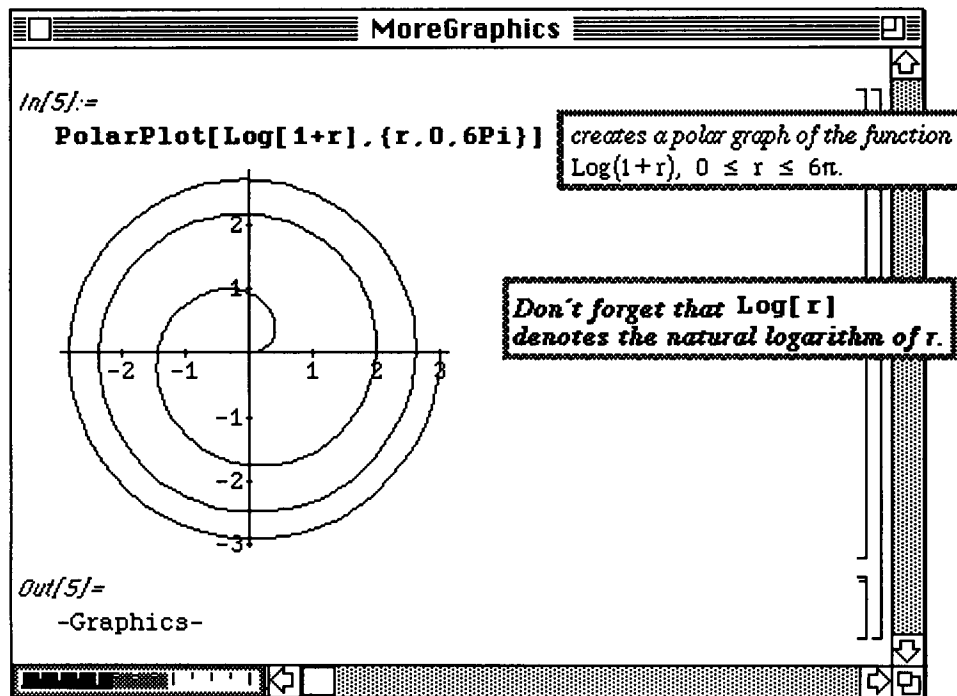
```

Since the two previous graphs were assigned names, they can easily be recalled and graphed together with `Show[polarone, polartwo]`.



□ Example:

The following command produces the "spiral-shaped" graph of $\text{Log}[1+r]$. Note that r represents the angular coordinate.

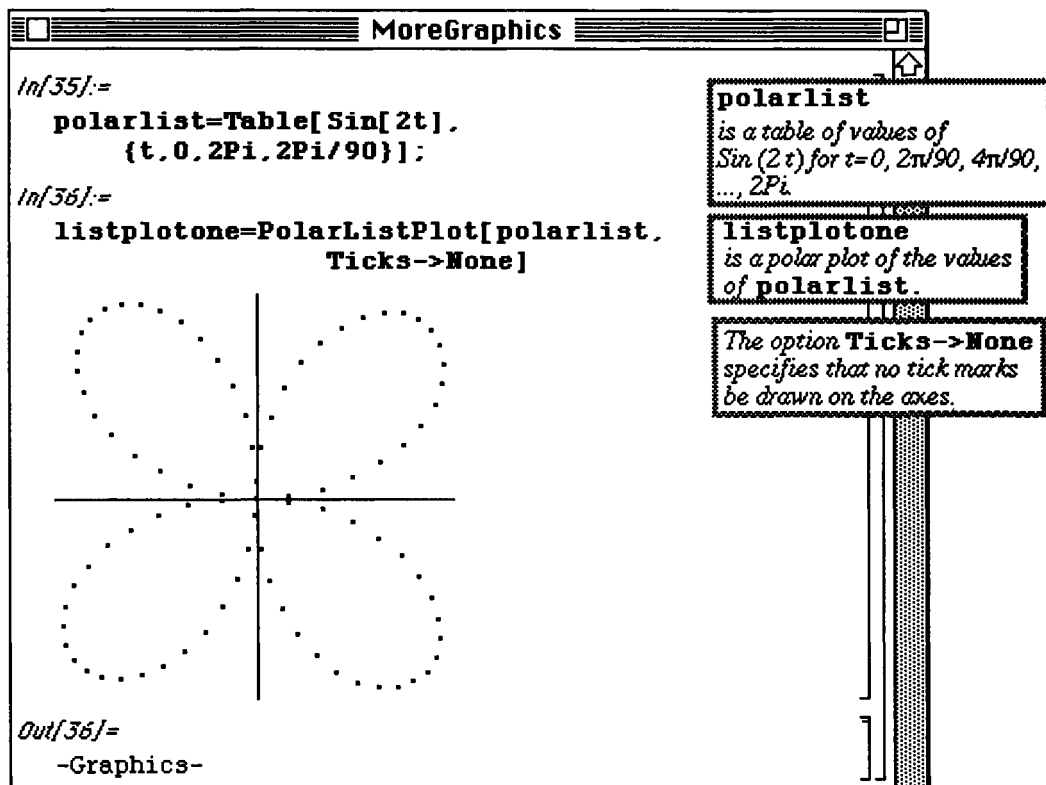


Another useful command in `Graphics.m` is `PolarListPlot`. This enables the user to be able to plot a list of points given in polar coordinates. This command is stated in the following manner:

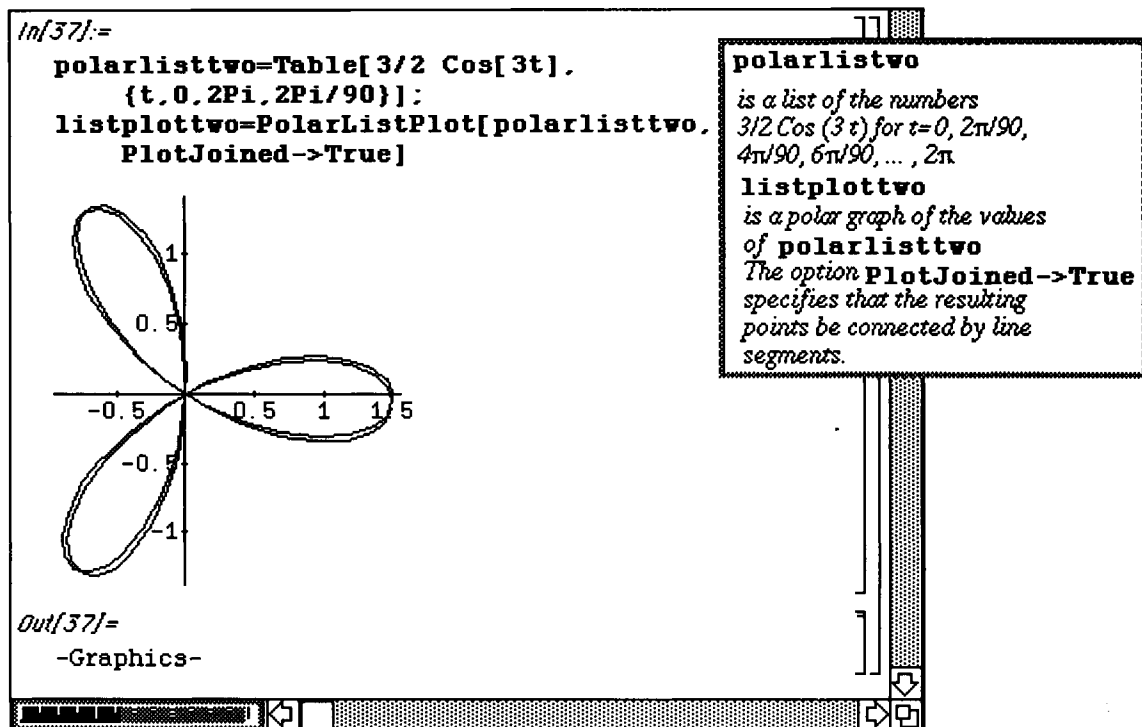
`PolarListPlot[list, options]` where `list` a list of points. A convenient way to enter `list` is in the form of a table.

□ Example:

In the example which follows, a table of values of $\sin(2t)$ is created where t varies from 0 to 2π , using increments of $2\pi/90$. In this case, the variable t represents the angular coordinate. The graph is given below and named **listplotone**.



Next, the graph of the list of values of $3/2 \cos(3t)$ is produced. In the `PolarListPlot` command, the `PlotJoined->True` option is used, so the points are connected. This plot is called `listplottwo`. Notice the overlap in the graph; the three-petal rose is produced for t between 0 and π . Hence, the curve is retraced for t between π and 2π .



The previous plots can be displayed simultaneously with **Show**.

The screenshot shows a Mathematica window titled "MoreGraphics". The input field contains the command `Show[listplotone, listplottwo, Ticks->Automatic]`. The output is a plot of two curves: a solid line and a dotted line. The plot has x-axis ticks at -0.5, 0.5, 1, and 1.5, and y-axis ticks at -1, -0.5, 0.5, and 1. A text box on the right explains that the `Show` command is used to display multiple plots together, and the `Ticks->Automatic` option allows Mathematica to choose suitable tick marks. The output field shows `Out[38]=` followed by `-Graphics-`.

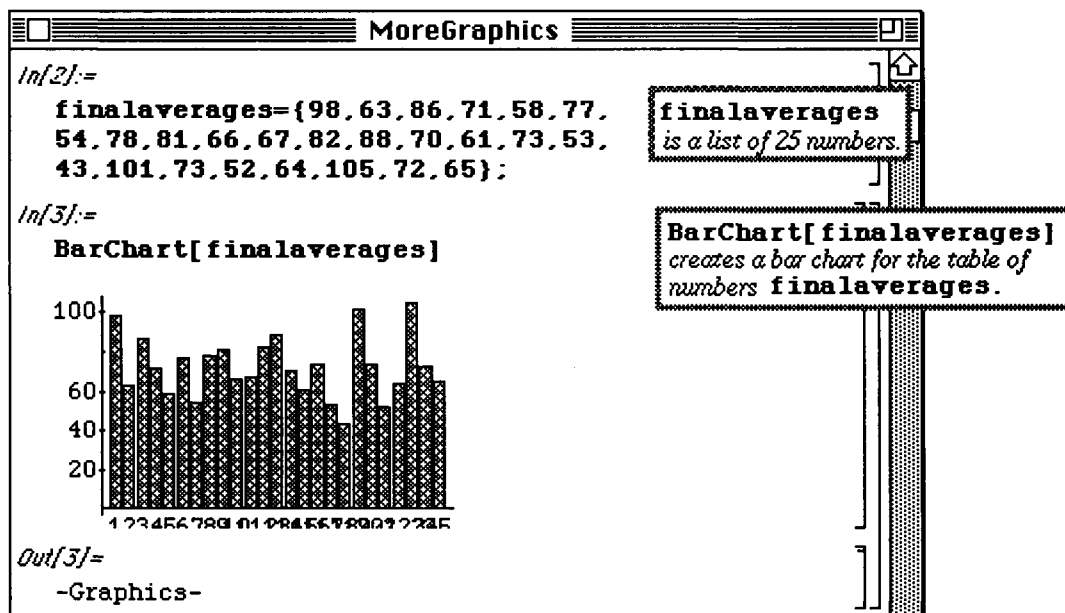
```
In[38]:= Show[listplotone, listplottwo, Ticks->Automatic]
```

The two graphs listplotone and listplottwo are shown simultaneously with the Show command. The option Ticks->Automatic allows Mathematica to "choose" suitable tick marks.

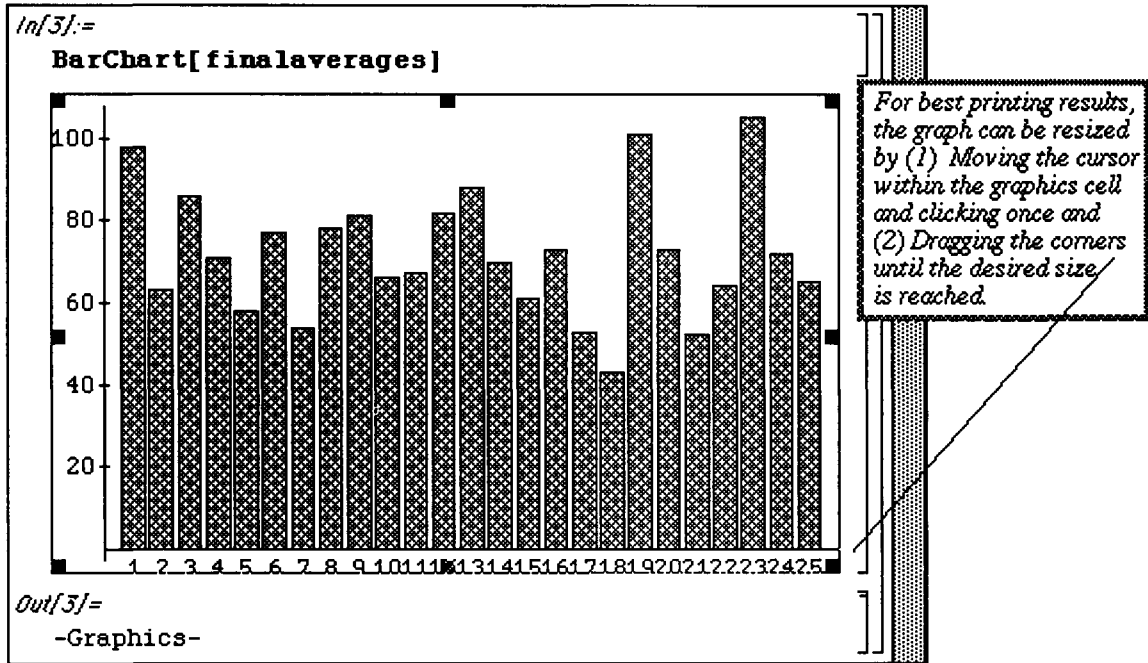
```
Out[38]=  
-Graphics-
```

Bar graphs can be drawn using `Graphics.m` with `BarChart[list]`. For each number in `list`, *Mathematica* draws a rectangle of that height. These rectangles are drawn in order from left to right. The position of the element is given beneath each rectangle. These numbers are quite small in the following graph, but resizing the graph, which will be described next, is possible.

□ Example:



Resizing is accomplished by clicking once anywhere inside the graphics cell. This encloses the graph in a box as shown below. If the cursor is moved to the lower right-hand corner and dragged, the graph can be enlarged to the desired size. Notice below that the numbers beneath the graph are now readable.



Before producing `BarChart[list]`, the elements of `list` can first be sorted. `Sort[list]` sorts the elements of `list` from smallest to largest. In the following example, the grades found in `finalaverages` are sorted and named `sort`. A shortened 2-line output of `sort` is given with `Short[sort,2]`, and the bar graph corresponding to the sorted list is produced with `BarChart[sort]`. The options of `BarChart` are listed below along with the default values. To use an option with `BarChart` simply enter the command as `BarChart[list, options]`.

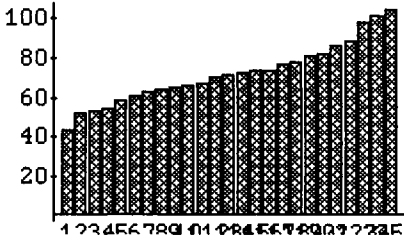
MoreGraphics
□

```

In[4]:=
  sort=Sort[finalaverages];
  Print[Short[sort,2]]
  BarChart[sort]

{43, 52, 53, 54, 58, 61, 63, 64, 65, 66, 67, 70,
  <<7>>, 82, 86, 88, 98, 101, 105}
  
```

The command `sort=Sort[finalaverages]` puts the list `finalaverages` in standard form. `Print[Short[sort,2]]` prints an abbreviated form of `sort`.



Finally, `BarChart[sort]` produces a bar chart corresponding to the list of numbers `sort`. Notice that `finalaverages` and `sort` consist of the same numbers; just in a different order.

```

Out[4]=
  -Graphics-

In[5]:=
  Options[BarChart]

Out[5]=
  {BarStyle -> GrayLevel[0.6], BarEdges -> True,
  BarEdgeStyle ->
  {GrayLevel[0], Thickness[0.002]},
  BarSpacing -> 0.2, BarOrientation -> Vertical}
  
```

`Options[BarChart]` displays the options and default settings of the command `BarChart`.

◀
▶

Pie charts are also possible with *Mathematica* by making use of the `PieChart[list]` command found in `Graphics.m`. This can be done in two ways. In the first example below, a pie chart is created from the list of numbers, `percents`. `PieChart[percents]` produces a pie chart in which each segment of the pie represents a number in `percents`. The segments are numbered to correspond to the position in `percents`. Notice that the sum of the numbers in `percents` is 1. However, a pie chart can be created which depicts both a quantity and a description. The list `description` is given below. Note that each element of `description` contains a number and a description. These represent portions of a governmental budget.

□ Example:

MoreGraphics
⏏

```

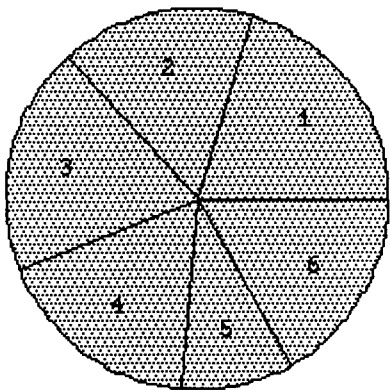
In[6]:=
percents={0.203,0.165,0.192,
0.176,0.095,0.169};

In[7]:=
description=
{{.203,defense},{.165,transportation},
{.192,services},{.176,education},
{.095,debt},{.169,other}}

Out[7]=
{{0.203,defense},{0.165,transportation},
{0.192,services},{0.176,education},
{0.095,debt},{0.169,other}}

In[8]:=
PieChart[percents]

```



`percents`
is a table of six numbers.

`description`
is a table of six pairs. For each pair, the first is a number, the second is a description of the number.

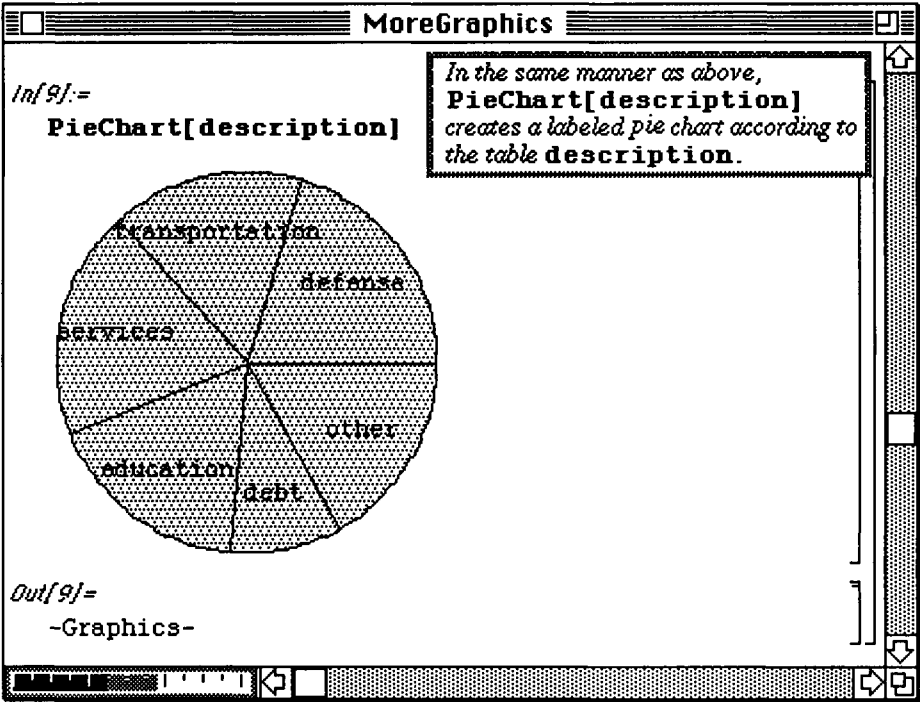
`PieChart[percents]`
creates a pie chart corresponding to the table `percents`.

```

Out[8]=
-Graphics-

```

Hence, a labeled pie chart is given with `PieChart [description]`. This chart demonstrates the percentage of the budget that is allotted to each area.



■ 8.2 Polyhedra.m

Pictures of polyhedra can be produced with **Polyhedra.m**. Many geometrical properties of polyhedra are stored in this package, so some pictures can be obtained by specifying a desired polyhedra with **Show[Polyhedron[Shape]]**. Since stored polyhedra include the icosahedron, dodecahedron, octahedron, cube, and tetrahedron. **Shape** is one of the following: **Icosahedron**, **Dodecahedron**, **Octahedron**, **Cube**, or **Tetrahedron**. If unspecified, the center is taken to be (0,0,0). A cube centered at the origin is produced in the first example below.

Several polyhedra can be shown simultaneously and, thus, complicated three-dimensional objects can be constructed. However, another command which involves more options must first be introduced.

Three-dimensional graphics objects are created but not displayed with

Graphics3D[Polyhedron[{x0,y0,z0},scale]] where **Polyhedron** is the desired polyhedron from the list of stored polyhedra: **Icosahedron**, **Dodecahedron**, **Octahedron**, **Cube**, and **Tetrahedron**. {x0,y0,z0} represents the center, and **scale** adjusts the size. The default value of **scale** is 1, so **scale > 1** produces a larger polyhedron and **scale < 1**, a smaller one. The second command below creates and stores (as **figure1**) the graphics of a dodecahedron centered at the origin using **scale = 1/2**. Since **Show** is not used, the picture is not shown.

```

In[11]:=
<<Polyhedra.m
In[12]:=
Show[Polyhedron[Cube]]
Out[12]=
-Graphics3D-
In[13]:=
figure1=Graphics3D[
  Dodecahedron[{0,0,0},1/2]]
Out[13]=
-Graphics3D-

```

contains definitions of the polyhedra icosahedron, dodecahedron, octahedron, cube, and tetrahedron.

shows a cube with center (0,0,0)

a dodecahedron with center (0,0,0) and scale 1/2.

□ Example:

Next, the graphics of an octahedron centered at $\{\text{Cos}[\text{Pi}/3], \text{Sin}[\text{Pi}/3], 0\}$ and $\text{scale} = 1/3$ is created and stored as **figure2**. Also, a tetrahedron with center $\{\text{Cos}[2\text{Pi}/3], \text{Sin}[2\text{Pi}/3], 1/3\}$ and $\text{scale} = 1/4$ is stored as **figure3**. Since the graphics of each polyhedra was named, they can be shown simultaneously with `Show[figure1,figure2,figure3]`.

The screenshot shows a Mathematica notebook window titled "ThreeDGraphics". The input area contains the following code:

```

In[14]:=
  figure2=Graphics3D[
    Octahedron[ {Cos[Pi/3], Sin[Pi/3], 0}, 1/3]]
  figure3=Graphics3D[
    Tetrahedron[ {Cos[ 2Pi/3], Sin[ 2Pi/3],
                  1/3}, 1/4]]

Out[14]=
  -Graphics3D-

In[15]:=
  Show[ figure1, figure2, figure3]

Out[15]=
  -Graphics3D-

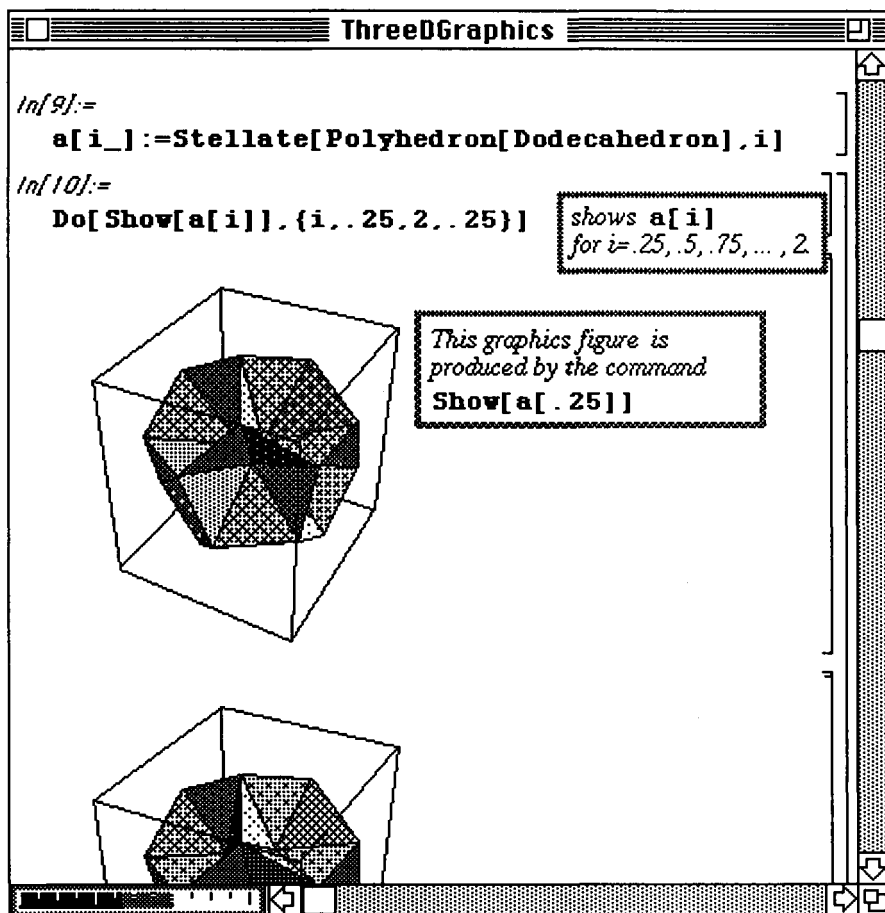
```

The output area displays a 3D plot of an octahedron and a tetrahedron inside a wireframe box. The octahedron is centered at $(\cos(\pi/3), \sin(\pi/3), 0)$ with a scale of $1/3$. The tetrahedron is centered at $(\cos(2\pi/3), \sin(2\pi/3), 1/3)$ with a scale of $1/4$.

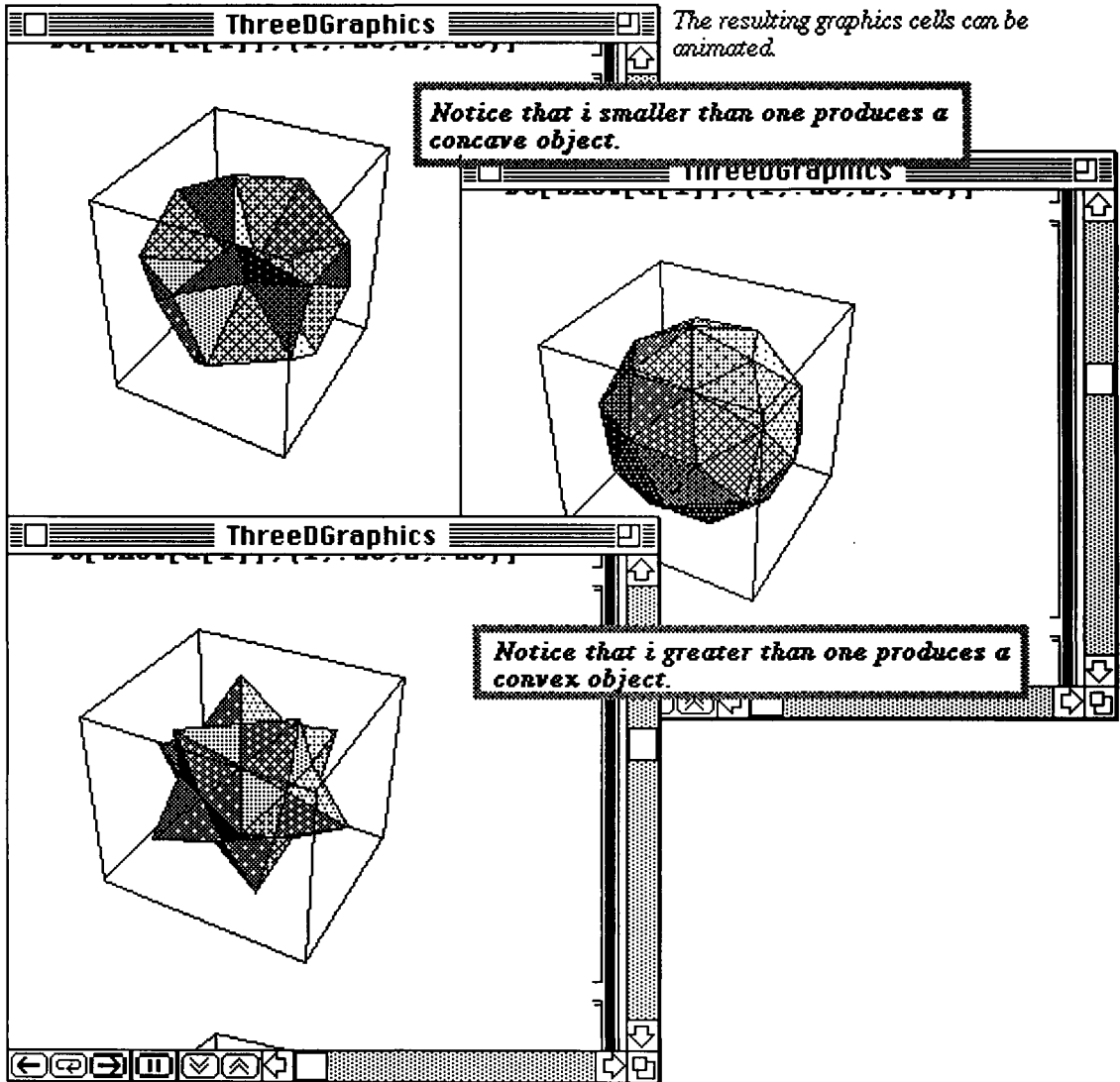
Two callout boxes provide additional information:

- A box on the right states: *figure2 is an octahedron with center $(\text{Cos}(\pi/3), \text{Sin}(\pi/3), 0)$ and scale $1/3$; figure3 is a tetrahedron with center $(\text{Cos}(2\pi/3), \text{Sin}(2\pi/3), 1/3)$ and scale $1/4$.*
- A box below it states: *shows all three polyhedra simultaneously.*

Another command available in `Polyhedra.m` is `Stellate[Polyhedron[Shape], ratio]` where `Shape` is again one of the following: `Icosahedron`, `Dodecahedron`, `Octahedron`, `Cube`, or `Tetrahedron`. This takes the symbolic representation of the polyhedron and represents it as a stellated polyhedron. (Each face is replaced by a stellate.) A function `a[i]` is defined below as `Stellate[Polyhedron[Dodecahedron], i]`. This function is then used in a `Do` loop to produce the graphics of stellated dodecahedra for values of `i` from `i=.25` to `i=2` using increments of `.25`. The first graph with `i=.25` is shown below.



Notice how the pictures change with **ratio**. If **ratio** < 1 , the object is concave. If **ratio** > 1 , the object is convex. Shown below are three pictures. In the first, **ratio** = $.25 < 1$, so the object produced is concave. In the third, **ratio** = $2 > 1$, so a convex object is given. Both can be compared to the middle picture which is simply a dodecahedron (**ratio** = 1). The graphics obtained with this **Do** loop can be animated to observe the changes which take place as **ratio** changes.



■ 8.3 Shapes.m

Shapes.m contains commands which produce the graphics of many shapes commonly used in mathematics. As with the all graphics objects, different shapes may be combined and shown simultaneously to create more complicated objects.

□ Example:

Illustrated first below is **MoebiusStrip**[*outerradius*, *innerradius*, *n*] where *innerradius* and *outerradius* are the inner and outer radii, respectively, and the Moebius strip is approximated using $2n$ polygons. (**MoebiusStrip** actually produces a list of polygons which are displayed with **Show**.) In the example below, the graphics are produced for a Moebius Strip with inner radius 2 and outer radius 4. The graph uses 60 polygons and is called **msone**. The list of polygons created with **MoebiusStrip** is visualized with **Show**[**msone**].

```

In[22]:=
  msone=MoebiusStrip[4,2,30]
Out[22]=
  -Graphics3D-
In[23]:=
  Show[msone]

```

msone
is a *Graphics3D* object consisting of 60 polygons. When visualized, these polygons approximate a Moebius Strip with inner radius 2 and outer radius 4.

Show[msone]
produces a visualization of **msone**.

□ Example:

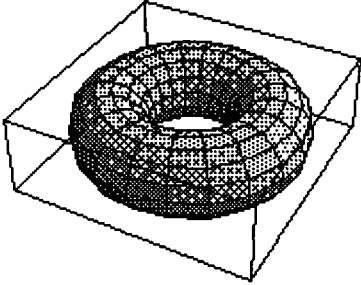
Another shape which can be approximated in this package is that of a torus. This is accomplished with `Torus[outerradius, innerradius, m, n]` where m times n polygons are used to approximate the shape of the torus. A torus of inner radius .5 and outer radius 1 is approximated with 300 polygons and called `torusone`. The approximation is then shown with `Show[torusone]`.

☐
☐
ThreeDGraphics

```

In[14]:=
  torusone=Torus[1, .5, 20, 15]
Out[14]=
  -Graphics3D-
In[15]:=
  Show[torusone]

```



```

Out[15]=
  -Graphics3D-

```

↑

torusone
is a *Graphics3D* object consisting of $20 \times 15 = 300$ polygons which approximate a torus with inner radius .5 and outer radius one when visualized.

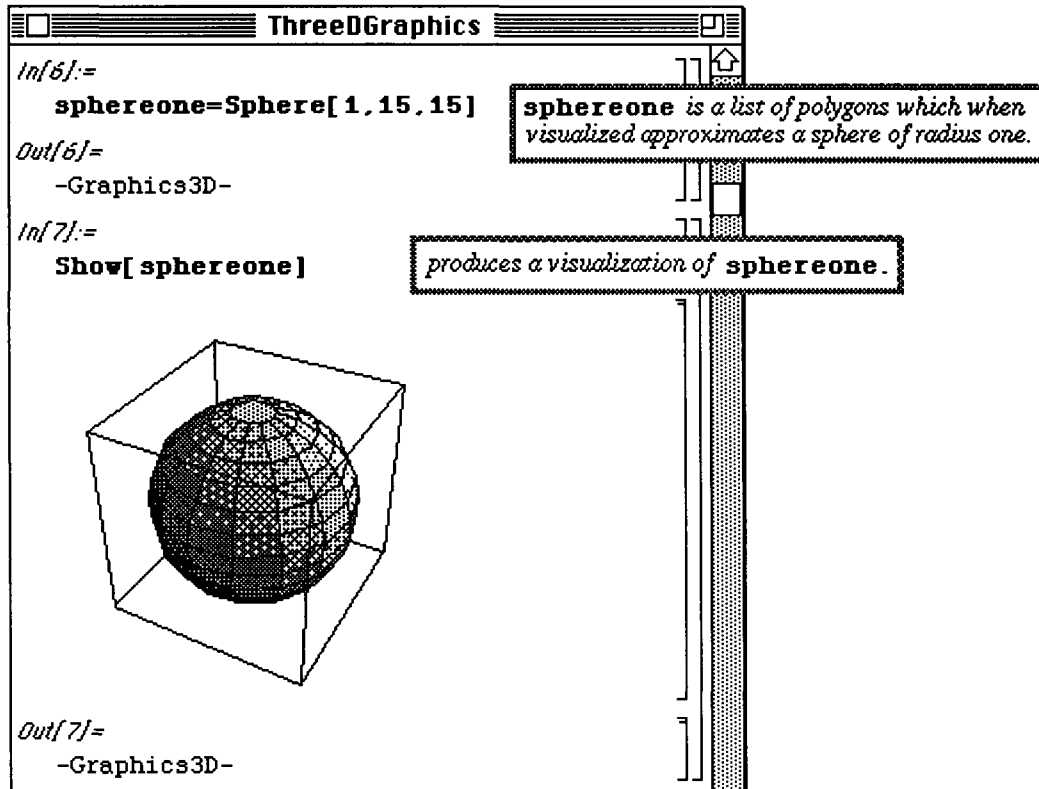
Show[torusone]
produces a visualization of **torusone**.

↓

←
→
☐

□ Example:

The command **Sphere**[*r*, *m*, *n*] produces an approximation of a sphere of radius = *r* using *m* times *n* polygons. The approximation of a sphere of radius 1 is obtained below using 225 polygons.



Several other commands are available for visualizing the lists of polygons produced by the commands found in **Shapes.m**. The command **WireFrame**[*polygonlist*] replaces each polygon in *polygonlist* by closed lines, so the shape resembles that of a wire frame when visualized.

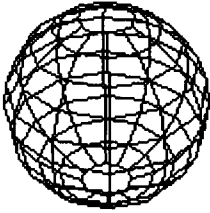
□ Example:

In the example below, a list of 144 polygons to approximate a sphere of radius 2 is obtained with **Sphere[2,12,12]**. This list is called **spheretwo**. **WireFrame** is then applied to this list of polygons, and the list of closed lines **wiretwo** which results is visualized with **Show[wiretwo,Boxed->False]**. (The **Show** option, **Boxed->False**, causes no box to be drawn around the sphere.)

```

In[16]:=
  spheretwo=Sphere[2,12,12]
Out[16]=
  -Graphics3D-
In[17]:=
  wiretwo=WireFrame[spheretwo]
Out[17]=
  -Graphics3D-
In[18]:=
  Show[wiretwo,Boxed->False]

```



The image shows a 3D wireframe sphere, which is a sphere constructed from a grid of lines. The sphere is centered in the lower-left quadrant of the notebook's graphics area. The notebook interface includes a command input area at the top, a list of inputs and outputs, and a 3D graphics window at the bottom. The sphere is rendered with black lines on a white background.

spheretwo is a list of polygons which when visualized approximates a sphere of radius two.

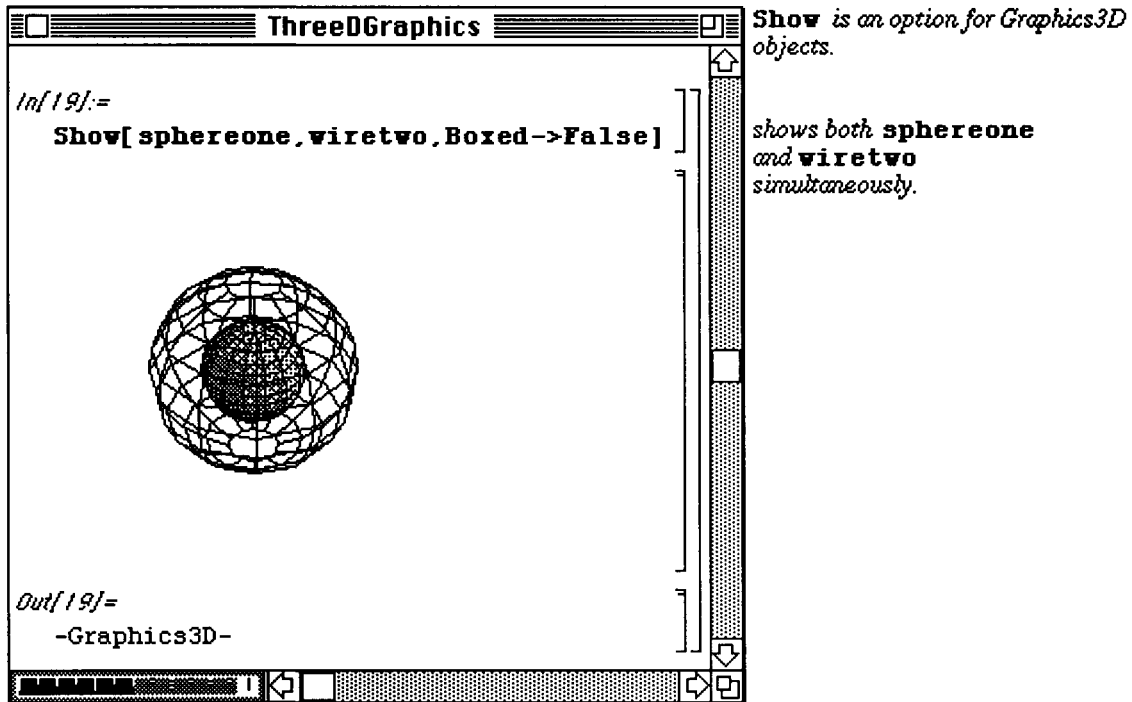
replaces each polygon of spheretwo by closed lines.

produces a visualization of wiretwo; the option Boxed->False specifies that sphere not be enclosed by a box.

Shapes can be viewed simultaneously using the commands previously discussed.

□ Example:

The following example shows how the approximations of the sphere of radius 1, **sphereone**, and the wire frame form of the sphere of radius 2, **wiretwo**, are shown together.



Below, the graphs of two Moebius strips are shown simultaneously by first computing the lists of approximating polygons in **msone** and **mstwo**. **Show** is then used to visualize these objects. Two other useful commands in **Shapes.m** are illustrated in the second example.

RotateShape [**shape**, **xrotate**, **yrotate**, **zrotate**] causes **shape** to be rotated **xrotate** units about the x-axis, **yrotate** units about the y-axis, and **zrotate** units about the z-axis.

In the example below, **msone** is rotated $\pi/2$ units about the y-axis.

The other command introduced is **TranslateShape** [**shape**, {**x0**, **y0**, **z0**}] which translates **shape** **x0** units along the x-axis, **y0** units along the y-axis, and **z0** units along the z-axis.

□ Example:

The Moebius strip obtained through rotation in `msone` is then translated 2 units along the x-axis and called `mstwo`. The two are then shown simultaneously with `mstwo` being the Moebius strip to the right of `msone`.

```

MoreShapes
In[1]:=
<<Shapes.m
In[32]:=
msone=MoebiusStrip[1,1/3,30]
mstwo=MoebiusStrip[2,1/3,30]
Show[msone,mstwo]

Out[32]=
-Graphics3D-
In[33]:=
msthree=RotateShape[msone,0,Pi/2,0]
msfour=TranslateShape[msthree,{2,0,0}]
Show[msthree,msfour]

Out[33]=
-Graphics3D-

```

`msone` and `mstwo` are Moebius strips centered around the z-axis. Notice that `Show` is an option for `Graphics3D` objects.

`msthree` is `msone` rotated $\pi/2$ radians about the y-axis. `msfour` is `msthree` translated 2 units along the x-axis.

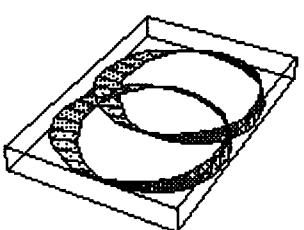
The Moebius strip **mstwo** given previously is then rotated $\pi/2$ radians about both the x- and z-axes. This is named **msfive**. Then, **msfive** is translated (-2) units along the y-axis to obtain **mssix**. **msfive** and **mssix** are then visualized below with **Show** where **mssix** is the Moebius strip to the rear of **msfive**. Several **Graphics3D** objects can be viewed simultaneously as shown below.

MoreShapes

```

In[34]:=
msfive=RotateShape[mstwo,Pi/2,0,Pi/2]
mssix=TranslateShape[msfive,{0,-2,0}]
Show[mssix,msfive]

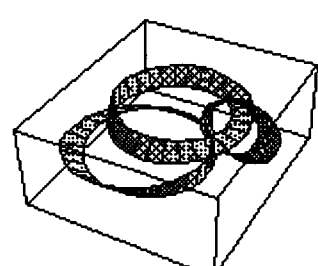
```



```

Out[34]=
-Graphics3D-
In[35]:=
Show[mstwo,msfour,mssix]

```



```

Out[35]=
-Graphics3D-

```

Similarly, **msfive** is **mstwo** rotated $\pi/2$ radians about the x-axis and the z-axis. **mssix** is **msfive** translated -2 units along the y-axis.

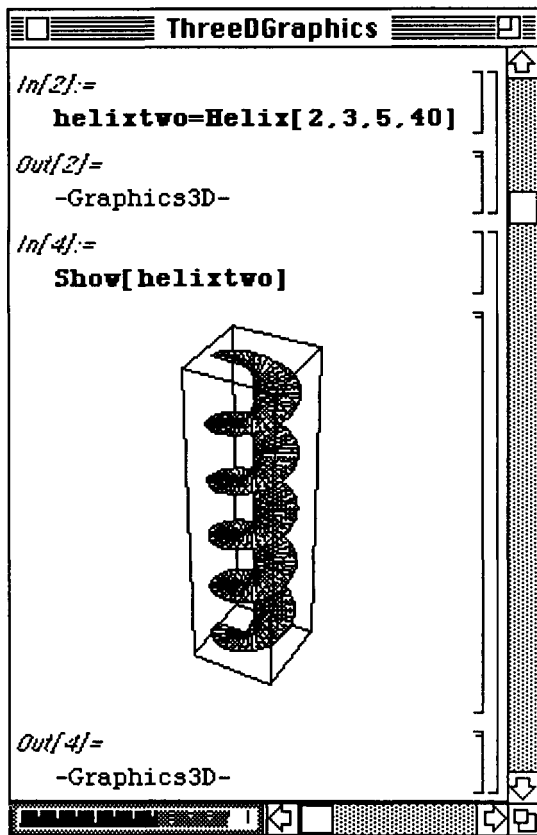
Since **Show** is an option for **Graphics3D**, several **Graphics3D** objects may be displayed simultaneously.

*displays **mstwo**, **msfour**, and **mssix** simultaneously.*

Yet another shape that can be graphed in **Shapes.m** is **Helix[r,h,m,n]** which approximates a helix with half height **h** and **m** turns using $m * n$ (where $n = 20\pi$) polygons.

□ Example:

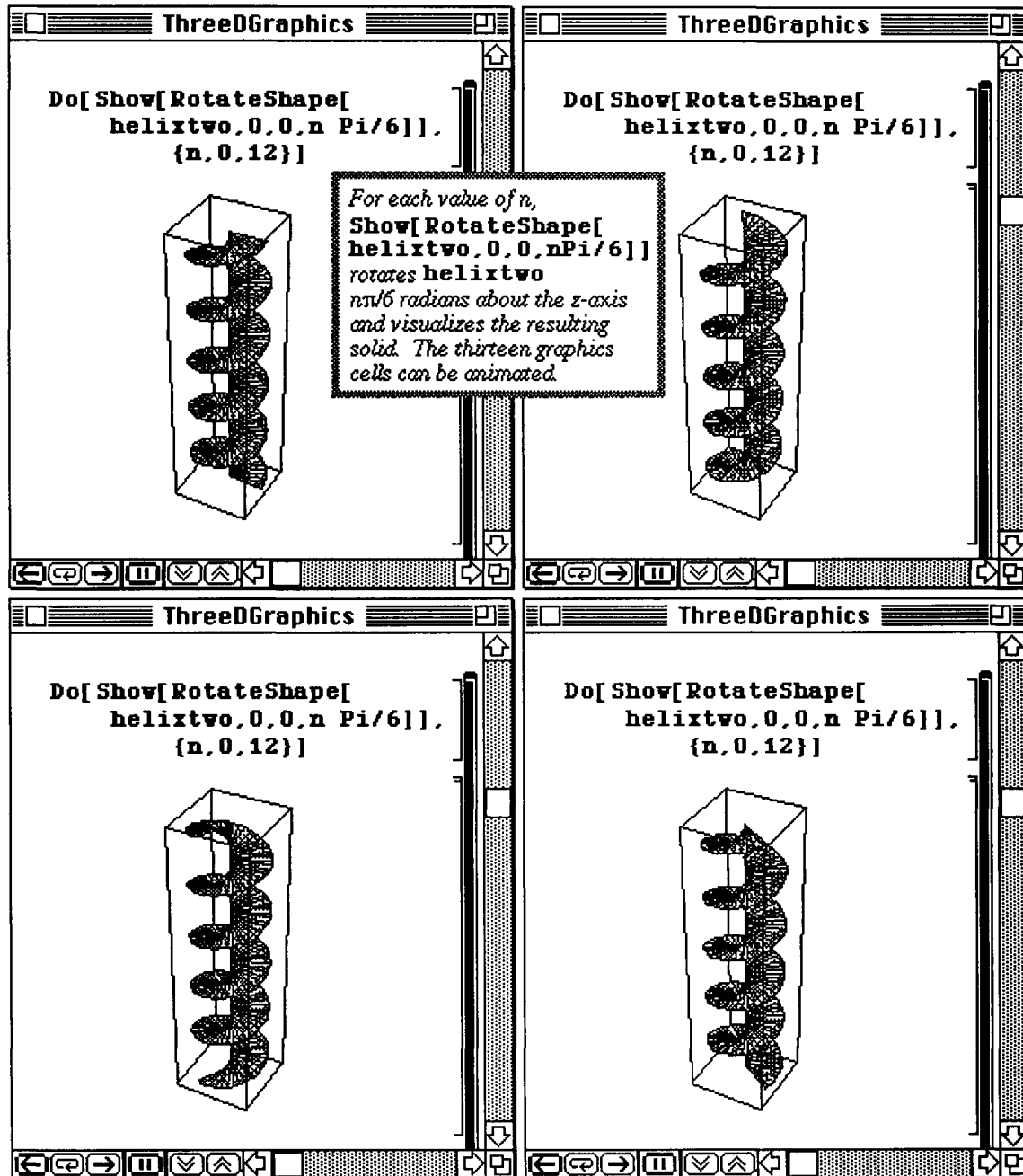
Shown below is a helix of half height 3 with 5 turns. The list of polygons which approximate the helix is found in **helixtwo**. Hence, **Show[helixtwo]** displays the helix.



helixtwo is a list of $40 \times 5 = 200$ polygons. When the polygons are visualized, the result approximates a helix with 5 turns and half-height 3.

Show[helixtwo] produces a visualization of **helixtwo**.

A `Do` loop which shows the rotation of `helixtwo` $n \cdot \pi/2$ radians (where n varies from $n = 0$ to $n = 12$) about the z -axis is defined below. This loop produces 13 graphics cells which can be animated to view the rotation of the helix about the z -axis. Several of these cells are shown below.



□ Example:

In the following example, we define a function `surface[n]` which simultaneously graphs the tori obtained with `torusj[1, 1/2, {0, 9/5(j-1), 0}]` for $j=1$ to $j=n$. The result allows us to visualize a surface of genus n .

The command `torusj`, given in the first line below, is defined in terms of `Torus` and may seem redundant. However, defining it in this manner enables *Mathematica* to remember the previous torus as it proceeds through the loop contained within the `Block`. Hence, `torusj[j-1]` does not have to be recomputed in order to find `torusj[j]`. After the loop is completed for $j=n$, the table of tori found in `s` is displayed with `Show`. The surface obtained with $n=4$ is shown below.

o In Version 2.0, `Block`, although still recognized, has been replaced by the command `Module`.

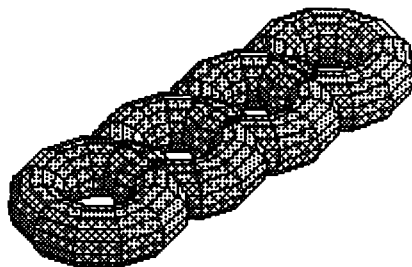
```

torusj[a_., b_., c_., d_] := torusj[a, b, c, d] = Torus[a, b, c, d]
surface[n_] := Block[ {torus, s},
    torus[j_] := TranslateShape[torusj[1, 1/2, 12, 15],
        {0, 9/5 (j-1), 0}];
    s = Table[torus[j], {j, 1, n}];
    Show[s, ViewPoint -> {-4.000, 4.000, 4.000},
        Boxed -> False];
]

```

`surface[4]`

By translating tori, we are able to visualize a surface of genus n . In this case, we are able to visualize a surface of genus 4.



■ 8.4 ParametricPlot3D.m

ParametricPlot3D.m provides the capabilities to produce three-dimensional graphs of functions which depend on two parameters. These equations exist in many different forms. The example below illustrates how the surface of revolution generated by a function of one variable revolved about the y -axis can be generated with **ParametricPlot3D**. The command is entered in the following way:

ParametricPlot3D[$\{x[u, v], y[u, v], z[u, v]\}, \{u, u0, u1\}, \{v, v0, v1\}, options]$ where $x, y,$ and z are defined in terms of the parameters u and v . The limits on the parameters are $\{u0, u1\}$ and $\{v0, v1\}$.

- In Version 2.0, **ParametricPlot3D**[$\{x[t], y[t], z[t]\}, \{t, t0, t1\}, options]$ where $x, y,$ and z are defined in terms of the parameter t graphs the vector-valued function $\{x[t], y[t], z[t]\}$ for $t0 \leq t \leq t1$.

In addition to the command **ParametricPlot3D**, Versions 1.2 and 2.0 of the package

ParametricPlot3D.m contain the command

SphericalPlot[$r[theta, phi],$

$\{theta, theta0, theta1\}, \{phi, phi0, phi1\}, options]$ which is used to create a graph of $r[theta, phi]$ using spherical coordinates; and

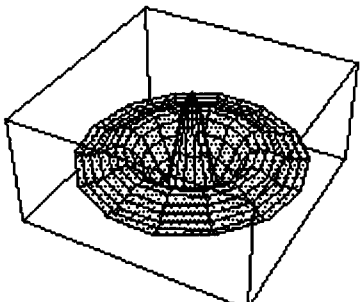
- the Version 2.0 edition of **ParametricPlot3D.m** contains the command

CylindricalPlot[$z[r, theta], \{z, z0, z1\}, \{theta, theta0, theta1\}, options]$ which is used to graph $z[r, theta]$ using cylindrical coordinates.

Since the surface of revolution generated by revolving $y = f(x) = \frac{\cos(\pi x)}{x^3 + 1}$ about the y -axis

is radially symmetric (z does not depend on the angular coordinate), the surface is visualized by representing x and y in polar coordinates and replacing x in the function $\cos[2 \text{ Pi } x] / (x+1)$ with u to form the equation for z in terms of the parameters.

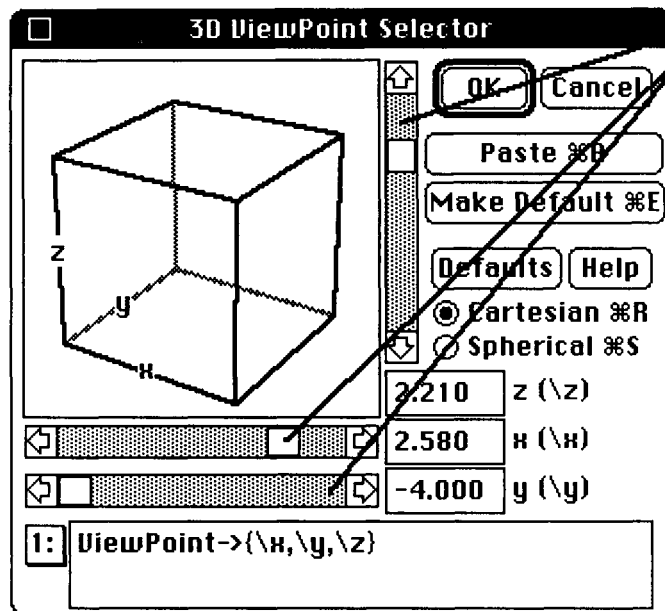
```
In[18]:=
ParametricPlot3D[{u Cos[v], u Sin[v], Cos[Pi u]/(u^3+1)},
{u, 0, Pi}, {v, 0, 2Pi}, Shading->False, BoxRatios->{1, 1, 1/2}]
```



In this case, the result corresponds to a solid of revolution.

```
Out[18]=
-Graphics3D-
```

In order to change the perspective from which three-dimensional graphics are viewed, the **3D ViewPoint Selector** which is shown below can be used. The window seen below is obtained by selecting **Prepare Input** under **Action** in the *Mathematica* Menu. The sub-menu found under **Prepare Input** contains this **3D ViewPoint Selector**. The numbers in the three boxes on the right represent the coordinates of the viewpoint. The x and y coordinates are changed by dragging the boxes beneath the graphics window. Similarly, the z coordinate is increased and decreased by dragging the box which is along the right side of the graphics window. Once a desirable viewpoint is located, it can be pasted into the **ParametricPlot3D** command by clicking the **Paste** button.



Different viewing orientations may be obtained by using the Mouse and cursor to drag these boxes.

Note that if Spherical coordinates may be displayed instead of Cartesian coordinates.

In the next example, all three components $\{x, y, z\}$ depend on two parameters, x and t . Again, the x and y coordinates are defined as functions of the polar coordinates, x and t , where t represents the angular coordinate. The third coordinate is defined as the function $f[x, t]$. The graph of this function of two parameters is plotted for values of x from $x=0$ to $x=2$ using increments of $2/25$ and for $t=0$ to $t=2\pi$ with $2\pi/25$ increments. Several options are illustrated in this command. The graph is not enclosed in a box, the axes are given automatically, the ratios of the $x, y,$ and z coordinates are 1-1-1, and the viewpoint is selected to be the point $\{0.390, -4.000, 3.360\}$.

Another command included in the Version 1.2 edition of **ParametricPlot3D.m** is **SpaceCurve** which allows curves that depend on one variable to be plotted in three dimensions. This command is entered as **SpaceCurve** $\{\{x, y, z\}, \{t, t_0, t_1\}, \text{options}\}$ where the coordinates $x, y,$ and z depend on the parameter t and t varies from $t=t_0$ to $t=t_1$. The options for **SpaceCurve** are the same as those for **ParametricPlot3D**.

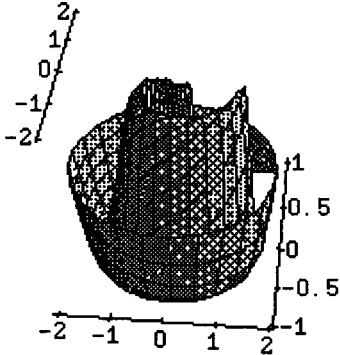
In the second example below, a spiral shaped curve is plotted. All three coordinates increase as t increases, so the points along the curve move away from the origin. The spiraling is due to the sine and cosine terms in the x and y coordinates.

- o In Version 2.0, the command **SpaceCurve** has been replaced by the command **ParametricPlot3D**. If using Version 2.0, **ParametricPlot3D** $\{\{x[t], y[t], z[t]\}, \{t, t_0, t_1\}\}$ yields the same result as the command **SpaceCurve** $\{\{x[t], y[t], z[t]\}, \{t, t_0, t_1\}\}$ from earlier versions.

ParametricPlots

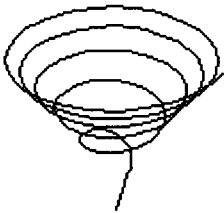
```
In[2]:=
Clear[f,x,y]
f[r_,t_]=Sin[(r-2) t]
x[r_,t_]=r Cos[t]
y[r_,t_]=r Sin[t]
ParametricPlot3D[{x[r,t],y[r,t],f[r,t]},
  {r,0,2,2/25},{t,0,2Pi,2Pi/25},Boxed->False,
  Axes->Automatic,BoxRatios->{1,1,1},
  ViewPoint->{0.390,-4.000,3.360}]
```

Once the desired orientation has been reached using the 3DViewPoint Selector, it can be inserted into document by clicking Paste.



The command **ParametricPlot3D** allows one to graph points that depend on two parameters.
The command **SpaceCurve** allows one to graph points that depend on one parameter.

```
Out[2]=
-Graphics3D-
SpaceCurve[{Sqrt[t] Cos[t],Sqrt[t] Sin[t],t^(1/3)},
  {t,0,10Pi,10Pi/150},
  Boxed->False,BoxRatios->{1,1,1}]
```



-Graphics3D-

□ **Example:**

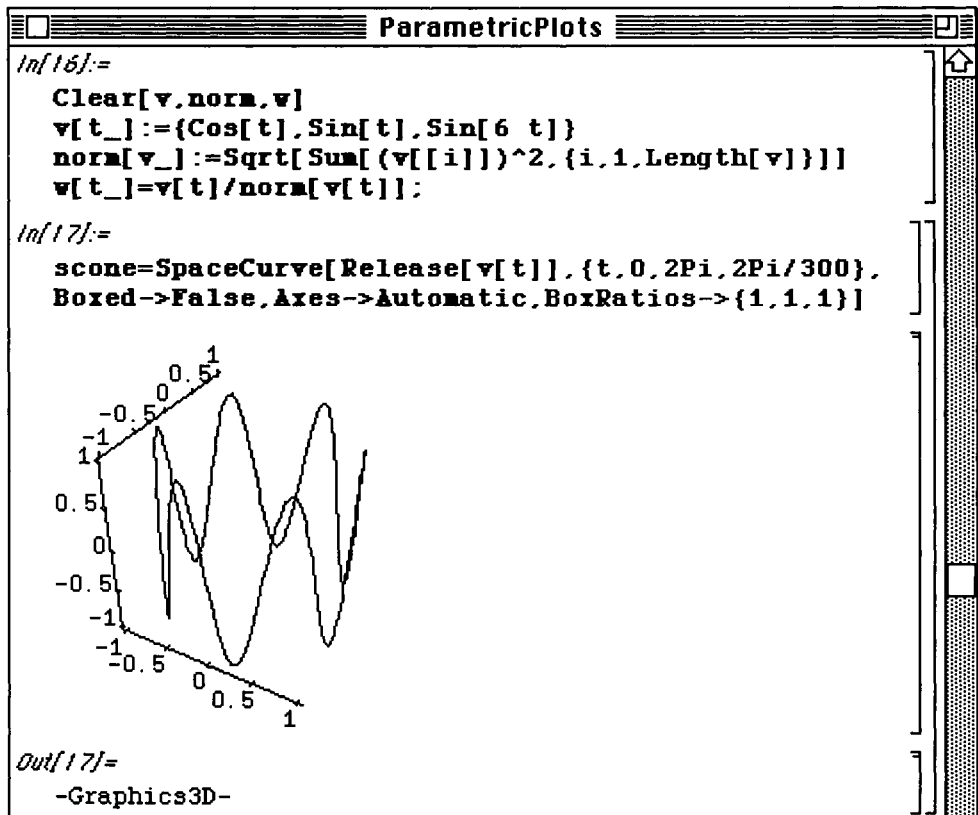
The following example shows how functions which have been previously defined can be graphed with **SpaceCurve**. A function $\mathbf{v}[t]$ is defined in terms of the parameter t . Then, a function **norm** $[t]$ is given. This function is simply the square root of the sum of the squares of the components of $\mathbf{v}[t]$. Since \mathbf{v} is a list, **Length** $[\mathbf{v}]$ represents the number of components of \mathbf{v} which in this case is 3.

In this case, defining **norm** $[\mathbf{v}__] := \text{Sqrt}[\mathbf{v} \cdot \mathbf{v}]$ produces the same result.

Finally, a function $\mathbf{w}[t]$ is defined as $\mathbf{v}[t]/\text{norm}[t]$.

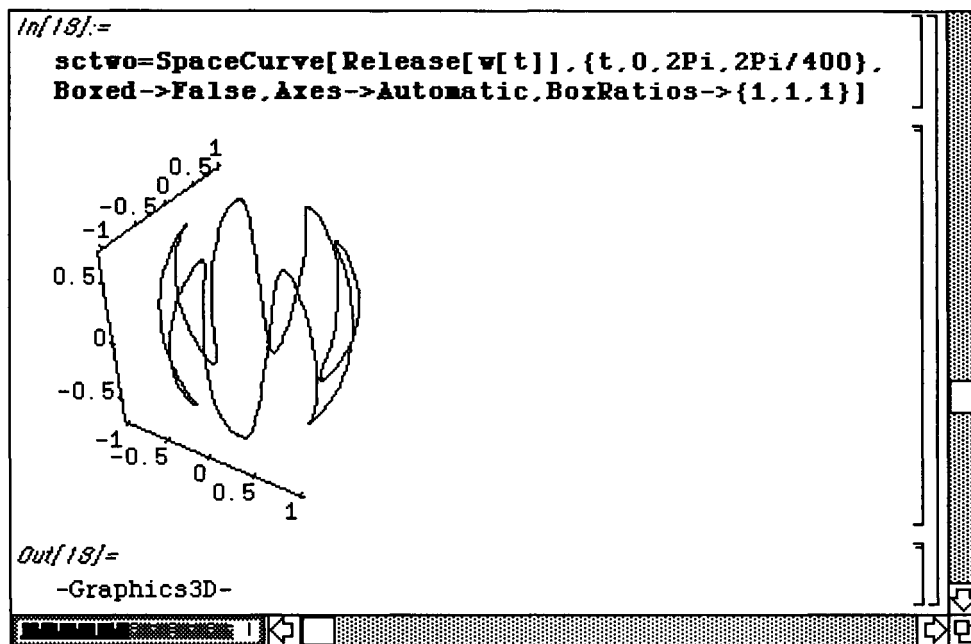
The graph of \mathbf{v} is plotted with **SpaceCurve** and called **scone**. Notice that **Release** $[\mathbf{v}[t]]$ must be entered in this command in order for the components of $\mathbf{v}[t]$ to be evaluated at the various values of t . The curve is plotted for $t=0$ to $t=2\pi$ using small increment sizes of $2\pi/300$.

- If using Version 2.0, use **ParametricPlot3D** instead of **SpaceCurve**; use **Evaluate** instead of **Release**.



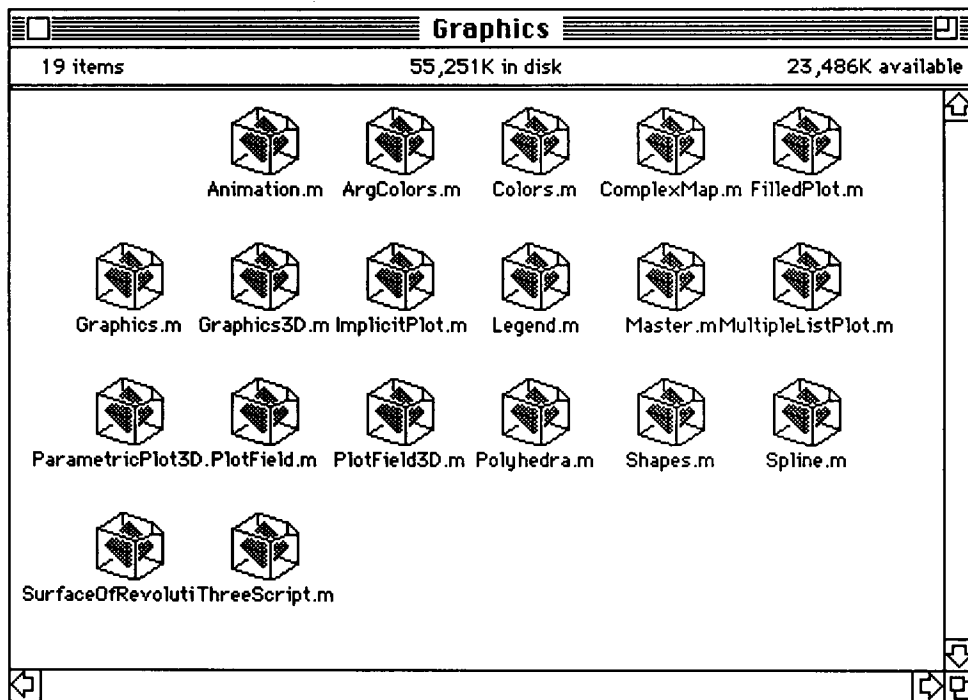
The graph of $w[t]$ is then plotted with a similar command. Again, `Release[w[t]]` must be used in order to plot this function of one parameter.

- o If using Version 2.0, be sure to use `Evaluate` instead of `Release`.



● Version 2.0 Graphics

The following window shows the contents of the Graphics folder in Version 2.0 of *Mathematica*. Notice that several new packages are included. In this section, we discuss *ImplicitPlot.m*,

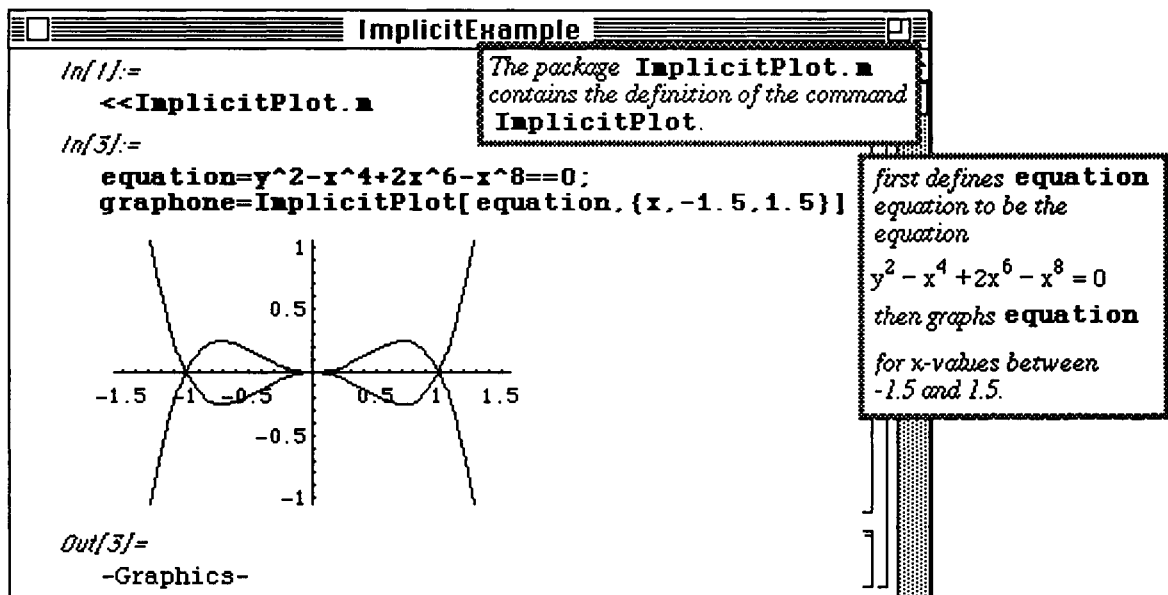


● 8.5 ImplicitPlot.m

Plotting an implicit equation was demonstrated earlier using the commands available in Version 1.2. This involved a rather lengthy procedure in which the equation was graphed in pieces. Fortunately, this task is not required in Version 2.0 which contains the **Graphics** package **ImplicitPlot.m**. This package includes the command **ImplicitPlot[equation, {x, xmin, xmax}]** which graphs the implicit equation, **equation**, from **x = xmin** to **x = xmax**. The set of y-values displayed may be specified by entering the command **ImplicitPlot[equation, {x, xmin, xmax}, {y, ymin, ymax}]**.

○ Example:

After loading **ImplicitPlot.m**, this command is demonstrated with the same equation that was plotted earlier with Version 1.2 in **Chapter 4** on page 237:

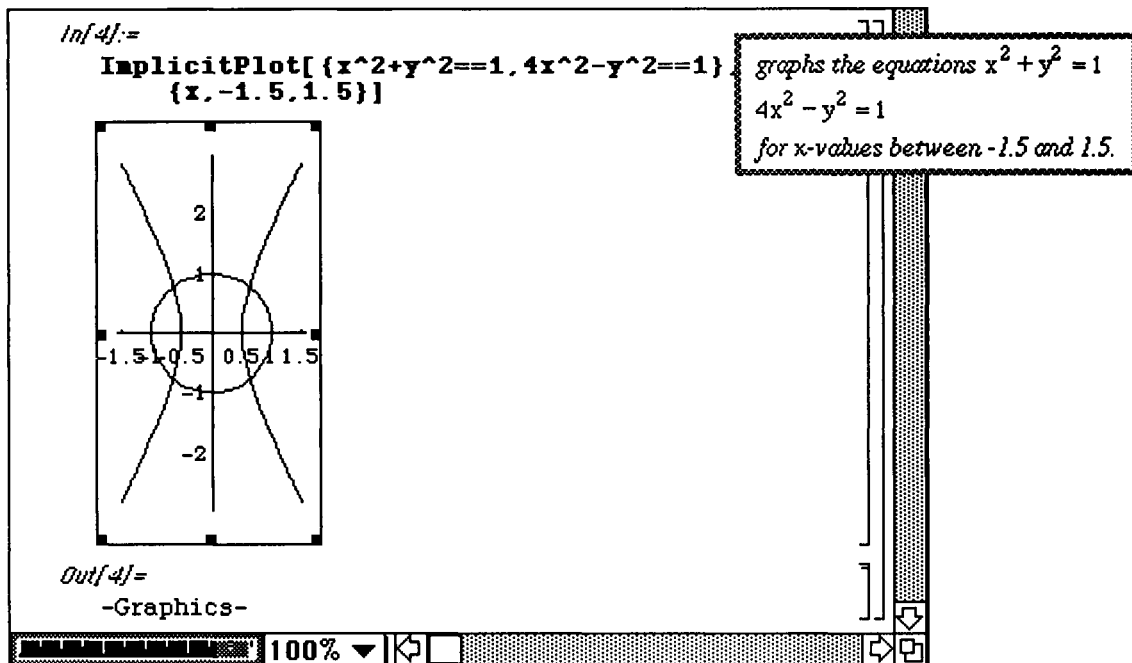


○ Example:

Implicit equations can be plotted simultaneously, as with the command `Plot`, with

`ImplicitPlot[{eq1, eq2, ..., eqn}, {x, xmin, xmax}]` and

`ImplicitPlot[{eq1, eq2, ..., eqn}, {x, xmin, xmax}, {ymin, ymax}]`. This is shown below. Recall that a double equals sign must be used with each equation.



□ Example:

Conic sections can also be plotted with `ImplicitPlot`. A table of conic equations of the form `conic=a x^2+b y^2==1` is produced below.

The values of `a` are found in `alist={-2, -1, 1, 2}` while those for `b` are in `blist={1, 2}`. The eight equations found in `conicequations` result when `conic` is evaluated at the values in `alist` and `blist`. Note that `conicequations` is a list of pairs of elements.

Hence, the conic equations can be extracted from this list with `conicequations[[i, j]]`, where `i=1,2,3,4` and `j=1,2`.

A function which yields the graphics of each equation is defined as

```
graph[i,j]=ImplicitPlot[conicequations[[i,j]],{x,-2,2},  
  DisplayFunction->Identity].
```

The option **DisplayFunction**->**Identity** causes the plot to be suppressed. Then, the table of graphics for each equation in **conicequations** is produced with

```
graphics=Table[graph[i,j],{i,1,4},{j,1,2}].
```

ImplicitExample

```
In[9]:=
Clear[conic]
conic=a x^2+b y^2==1;
alist={-2,-1,1,2};
blist={1,2};
conicequations=Table[conic /.
  a->alist[[i]] /. b->blist[[j]],
  {i,1,4},{j,1,2}]

General::spell1:
Possible spelling error: new symbol name
"blist" is similar to existing symbol
"alist".

Out[9]=
{{-2 x2 + y2 == 1, -2 x2 + 2 y2 == 1},
 {-x2 + y2 == 1, -x2 + 2 y2 == 1},
 {x2 + y2 == 1, x2 + 2 y2 == 1},
 {2 x2 + y2 == 1, 2 x2 + 2 y2 == 1}}
```

conicequations
is a table of equations obtained by substituting values from the list {-2,-1,1,2} for a and values from the list {1,2} for b in the general equation
 $ax^2 + by^2 = 1$.

Version 2 issues warnings if you define objects to have names similar to existing objects.

```
In[11]:=
graph[i_,j_]:=ImplicitPlot[
  conicequations[[i,j]],
  {x,-2,2},DisplayFunction->Identity]
graphics=Table[graph[i,j],{i,1,4},{j,1,2}]

General::spell1:
Possible spelling error: new symbol name
"graphics" is similar to existing symbol
"Graphics".

Out[11]=
{{-Graphics-, -Graphics-},
 {-Graphics-, -Graphics-},
 {-Graphics-, -Graphics-},
 {-Graphics-, -Graphics-}}
```

graph[i,j]
the jth member of the ith member of the list **conicequations**.
The resulting graphics object is suppressed with the option
DisplayFunction->**Identity**.
graphics is a table of the graphics objects **graph**[i,j] for i=1,2,3,4 and j=1,2.

The graphics contained in `graphics` can be visualized with `Show[GraphicsArray[graphics]]`. `GraphicsArray[graphics]` sets up similar rectangular areas to display each graph in `graphics`.

ImplicitExample

In[12]:=
Show[GraphicsArray[graphics]]

The resulting eight graphics cells can be visualized in an array of graphics cells using the command **Show[GraphicsArray[graphics]]**.

Notice that it is easy to identify each graph:

- $-2x^2 + 2y^2 = 1$
- $-x^2 + 2y^2 = 1$
- $x^2 + 2y^2 = 1$
- $2x^2 + 2y^2 = 1$

Out[12]=
 -GraphicsArray-

100%

● 8.6 PlotField.m

The package `PlotField.m` contains the commands `PlotVectorField` and `PlotGradientField` which are useful in many areas of physics and engineering.

`PlotVectorField[vector[x,y], {x,xmin,xmax}, {y,ymin,ymax}]` graphs the vector field given by the vector-valued function, `vector[x,y]`. This is illustrated below with the vector field `f[x,y]`.

○ Example:

PlotfieldExample

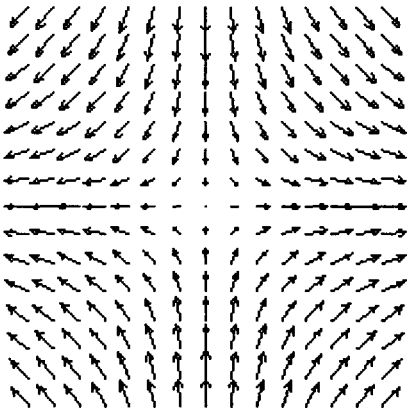
```

In[1]:=
<<PlotField.m

In[10]:=
f[x_,y_]={x/Sqrt[x^2+y^2+1],
-y/Sqrt[x^2+y^2+1]};
vfone=PlotVectorField[f[x,y],{x,-2,2}
{y,-2,2}]

```

The package `PlotField.m` contains the commands `PlotVectorField` and `PlotGradientField`.



After defining the vector field

$$f(x,y) = \left\{ \frac{x}{\sqrt{x^2+y^2+1}}, \frac{-y}{\sqrt{x^2+y^2+1}} \right\}$$

`PlotVectorField[f[x,y], {x,-2,2}, {y,-2,2}]` graphs the vector field `f` on the square `[-2,2] x [-2,2]`.

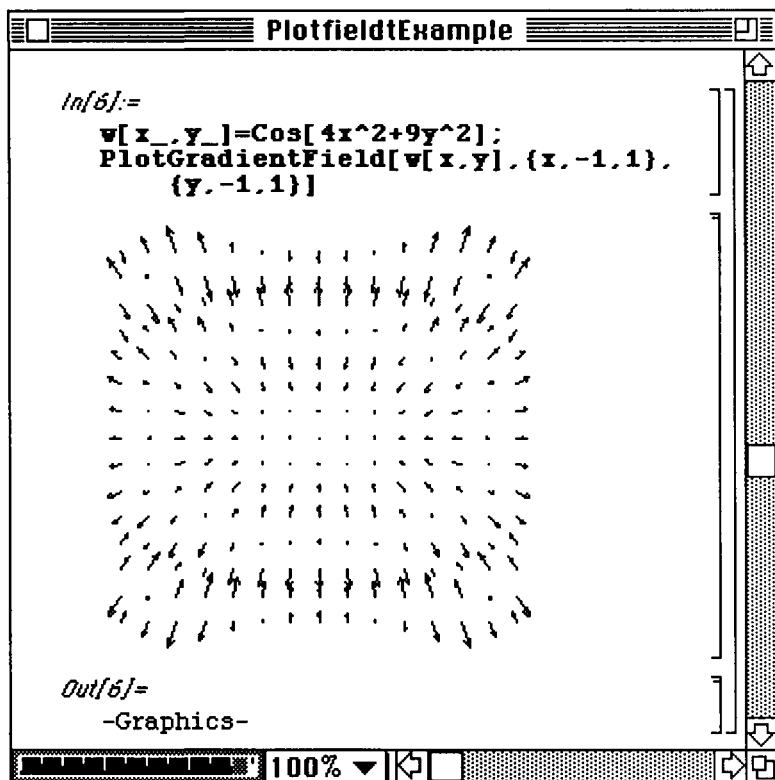
```

Out[10]=
-Graphics-

```

The command `PlotGradientField[function[x,y], {x,xmin,xmax}, {y,ymin,ymax}]` graphs the gradient field of the function, `function[x,y]`. This is done by first computing the gradient of `function[x,y]` (which yields a vector field) and then plotting the gradient. This is shown below with the function `w[x,y]=Cos[4x^2+8y^2]`.

o Example:



After defining

$w(x,y) = \cos(4x^2 + 9y^2)$,
`PlotGradientField[
w[x,y], {x,-1,1},
{y,-1,1}]`

computes the gradient of w
and then graphs the resulting
vector field on the rectangle
 $[-1,1] \times [-1,1]$.

● 8.7 PlotField3D.m

Vector fields can be plotted in three dimensions as well. The commands needed to plot these fields are found in the **PlotField3D.m** package. The syntax for the **PlotGradientField3D** and **PlotVectorField3D** commands are similar to those used in the two-dimensional case in the previous section with the additional z-component.

○ Example:

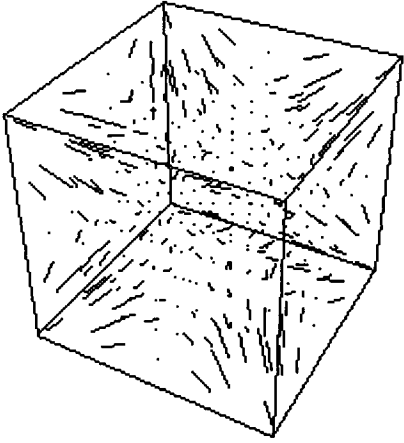
PlotGradientField3D is shown below with the three dimensional function, $\text{Cos}[x\ y\ z]$.

Plotfield3DtExample
⌵

```

In[1]:=
<<PlotField3D.m
pgf=PlotGradientField3D[
  Cos[x y z].{x,-Pi,Pi},
  {y,-Pi,Pi},{z,-Pi,Pi}]

```



-Graphics3D-

The commands **PlotGradientField3D** and **PlotVectorField3D** are contained in the package **PlotField3D.m**

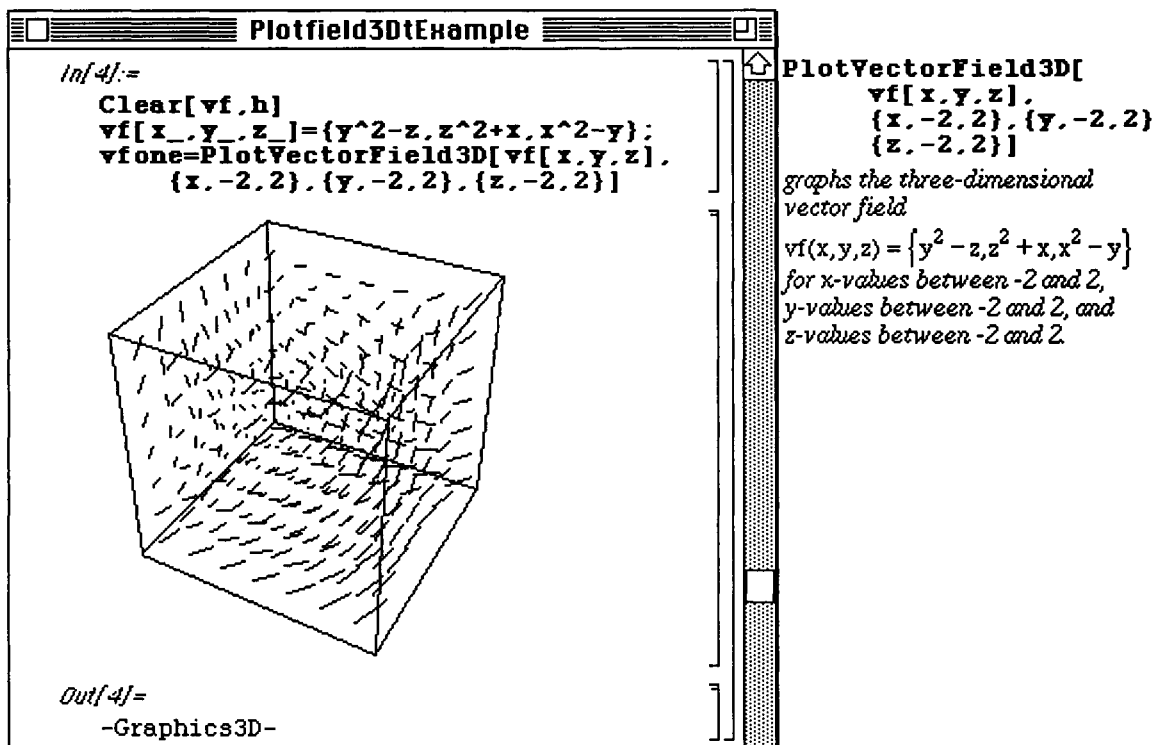
first computes the gradient of $\text{Cos}(x\ y\ z)$ and then graphs the resulting vector field for x-values between $-\pi$ and π , y-values between $-\pi$ and π , and z-values between $-\pi$ and π .

Don't forget to include the space between the x, y, and z to denote the product of x, y, and z.

100% ▾
⏪ ⏩

□ Example:

The vector field, $\mathbf{vf}[x, y, z] = \{y^2 - z, z^2 + x, x^2 - y\}$, is plotted below using the command `PlotVectorField3D`. This graph is named `vfone` for later use.



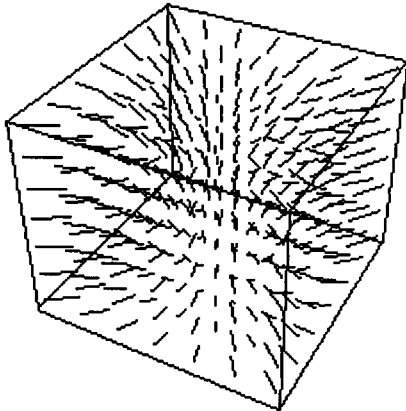
□ Example:

The function $h[x, y, z] = x^2 + y^2 + z - 4$ is defined below and the gradient field for $h[x, y, z]$, called **vgone**, is graphed with **PlotGradientField3D**. Notice that since the definition of **h** and the **PlotGradientField3D** command are enclosed in the same input cell and no semi-colon follows the definition of **h**, the formula for **h** is given as part of the output.

```
In[6]:=
h[x_, y_, z_] = x^2 + y^2 + z - 4
vgone = PlotGradientField3D[h[x, y, z],
  {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]
```

```
Out[5]=
-4 + x^2 + y^2 + z
```

General::spell1:
Possible spelling error: new symbol
name "vgone"
is similar to existing symbol
"vfone".



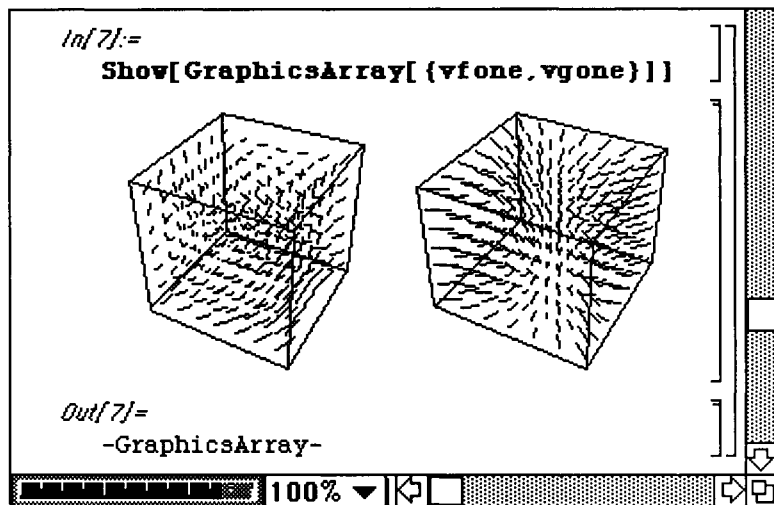
```
Out[4]=
-Graphics3D-
```

```
PlotGradientField3D[
  h[x, y, z], {x, -2, 2},
  {y, -2, 2}, {z, -2, 2}]
```

*first computed the gradient of
 $h(x, y, z) = x^2 + y^2 + z - 4$ and then
graphs the resulting vector field.*

*Notice that Version 2 displays
the definition of h since a semi-
colon was NOT included at the
command.
Version 2 warns of a possible
spelling error.*

The three-dimensional graphs, `vfone` and `vgone`, obtained earlier can be viewed together with `Show[GraphicsArray[{vfone, vgone}]]`.



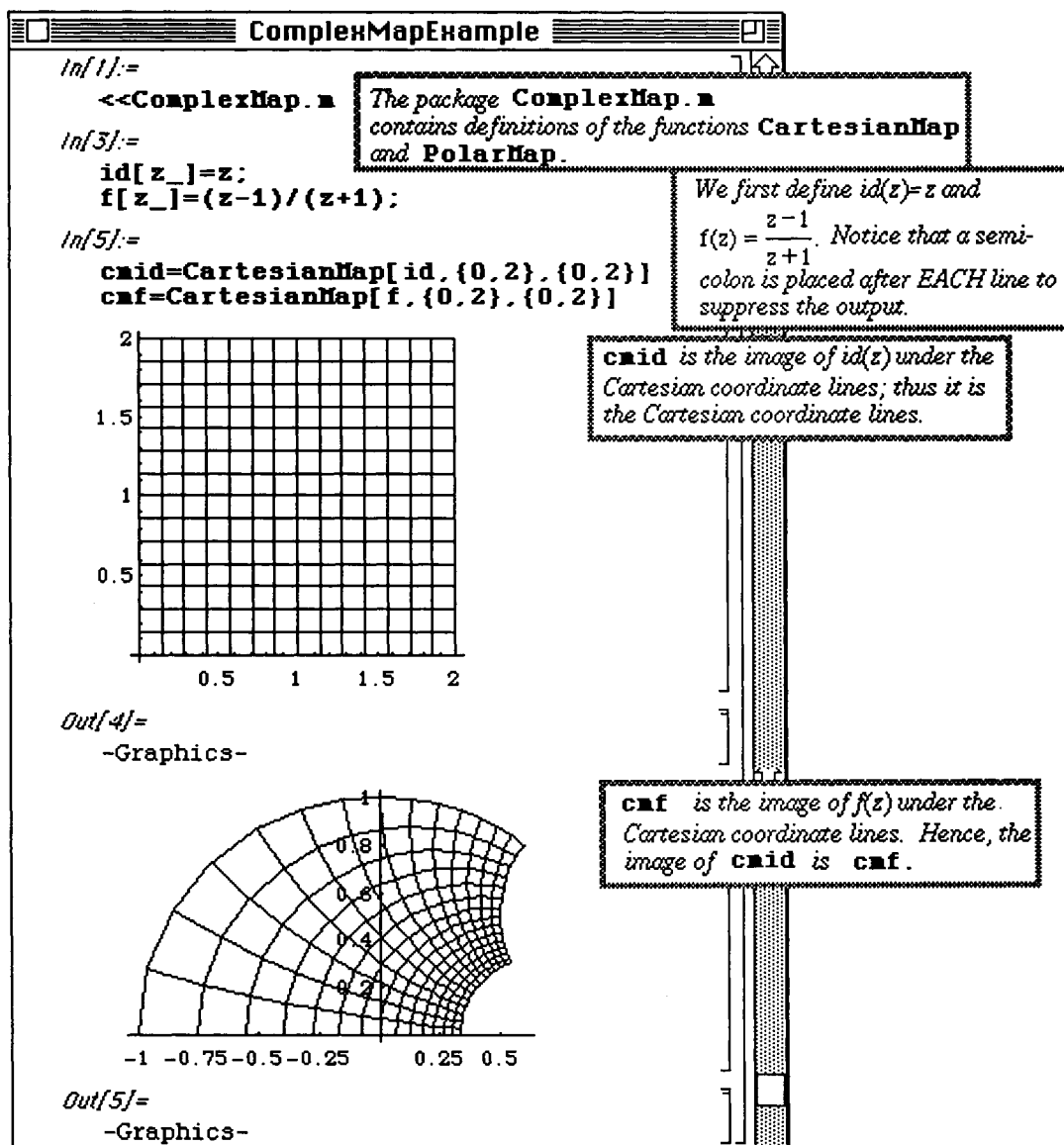
*both `vfone`
and `vgone`
can be displayed in a
single graphics cell with
the command
`Show[GraphicsArray
[{vfone, vgone}]]`.*

● 8.8 ComplexMap.m

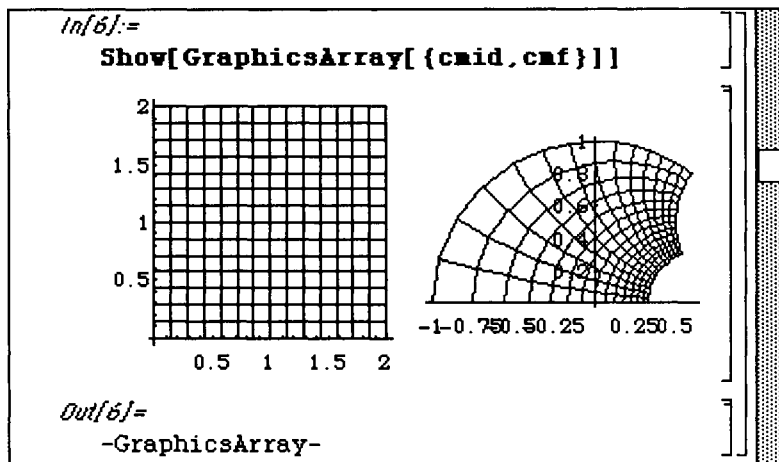
A problem of interest in complex analysis is finding the image of a complex-valued function $f(z)$. The package `ComplexMap.m` provides several commands which are useful in solving problems of this type.

`CartesianMap[f[z], {{x0, x1}, {y0, y1}}` gives the image of $f[z]$ using Cartesian coordinate grid lines over the rectangular region $\{x0, x1\} \times \{y0, y1\}$.

This is illustrated below with the functions $id[z]=z$ and $f[z]=(z-1)/(z+1)$. Since $id[z]$ is the identity map, each point in the domain is mapped to itself. Hence, the Cartesian grid, called `cmid`, is unchanged upon application of $id[z]$. (This region can therefore be viewed as the domain of $f[z]$.) The second graph, `cmf`, illustrates the effects that $f[z]$ has on the points in `cmid`.



The two graphics objects, **cmid** and **cmf**, can be viewed in a single graphics cell with **Show[GraphicsArray[{cmid, cmf}]]**. This gives the usual manner in which the domain and image of a function are illustrated.

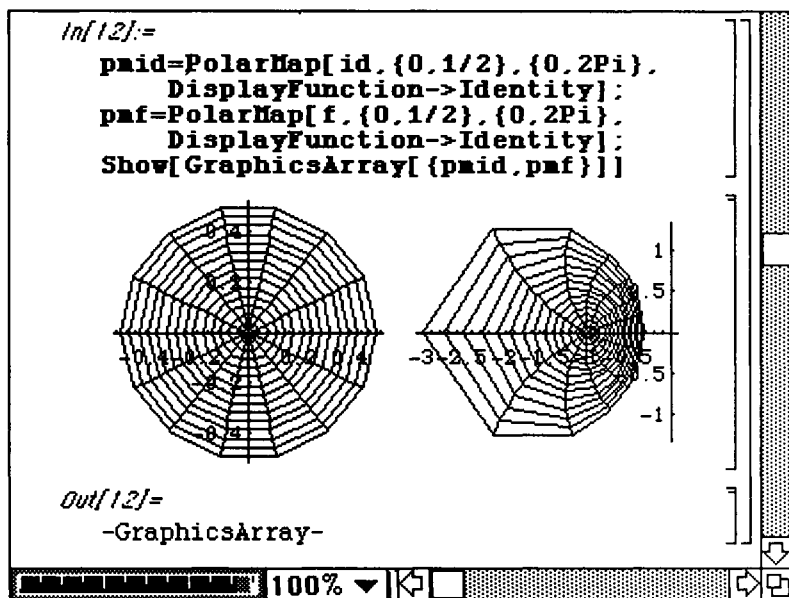


Show[GraphicsArray[{cmid, cmf}]]
*shows the two graphics objects
cmid and **cmf**
in a single graphics cell.*

In addition to Cartesian coordinates, polar coordinates can also be used. This is done with **PolarMap[f[z], {r0, r1}, {theta0, theta1}]** which produces the image of **f[z]** over the circular region **R** bounded by limits placed on the polar coordinates **r** and **θ**:

R : r0 ≤ r ≤ r1, theta0 ≤ θ ≤ theta1.

The following problem is worked in a method similar to that of the previous problem involving Cartesian coordinates. However, many of the graphs are suppressed by using the `DisplayFunction->Identity` option. Again, the identity map, $\text{id}[z]=z$, is used to produce the polar grid, called `pmid`, to be viewed as the domain of the function $f[z]$. The image of f , named `pmf`, is then determined with `PolarMap` and the two are displayed with `Show[GraphicsArray[{pmid, pmf}]]`.



In the same manner as above, `pmid` consists of the Polar coordinate lines and `pmf` is the image of the Polar coordinate lines. Remember that the option `DisplayFunction->Identity` suppresses the output of the resulting graphics.

The following example illustrates several ideas. First, the built-in *Mathematica* function `Identity` can be used to produce the domain grid for a function as opposed to defining $\text{id}[z]=z$ which was done in the previous examples. Also, `GraphicsArray` can be used to plot graphics cells in a desired order. For example, the domain and image of a function can be displayed consecutively.

This is done below for the functions $w[z]=(1-\text{Cos}[z])/(1+\text{Cos}[z])$ and $m[z]=(z-2)/(2z-1)$. The domain and image of w are called `cmid` and `cmw`, respectively, while those of m are named `pmid` and `pmm`. These graphics objects are shown in the appropriate order with the command `Show[GraphicsArray[{{cmid, cmw}, {pmid, pmm}}, AspectRatio->1]`. (Notice the grouping of {domain, image} within `GraphicsArray`.)

ComplexMapExample

```

In[17]:=
w[z_]=(1-Cos[z])/(1+Cos[z]);
m[z_]=(z-2)/(2z-1);

In[22]:=
cmid=CartesianMap[Identity,
  {0,Pi/2,Pi/20},{0,2,2/10},
  DisplayFunction->Identity];
cmw=CartesianMap[w,{0,Pi/2,Pi/20},{0,2,2/10},
  DisplayFunction->Identity];
pmid=PolarMap[Identity,
  {0,1,1/10},{0,2Pi},
  DisplayFunction->Identity];
pmm=PolarMap[m,
  {0,1,1/10},{0,2Pi},
  DisplayFunction->Identity];
Show[GraphicsArray[{{cmid,cmw},
  {pmid,pmm}},AspectRatio->1]]

```

Power::infy: Infinite expression - encountered.

1
0

Identity[x] is a built-in Mathematica function.

Even though Mathematica encounters an undefined expression, the final result is satisfactory.

corresponds to the Cartesian coordinate lines and the image of w(z) under the Cartesian coordinate lines.

corresponds to the Polar coordinate lines and the image of m(z) under the Polar coordinate lines.

Out[22]=
-GraphicsArray-

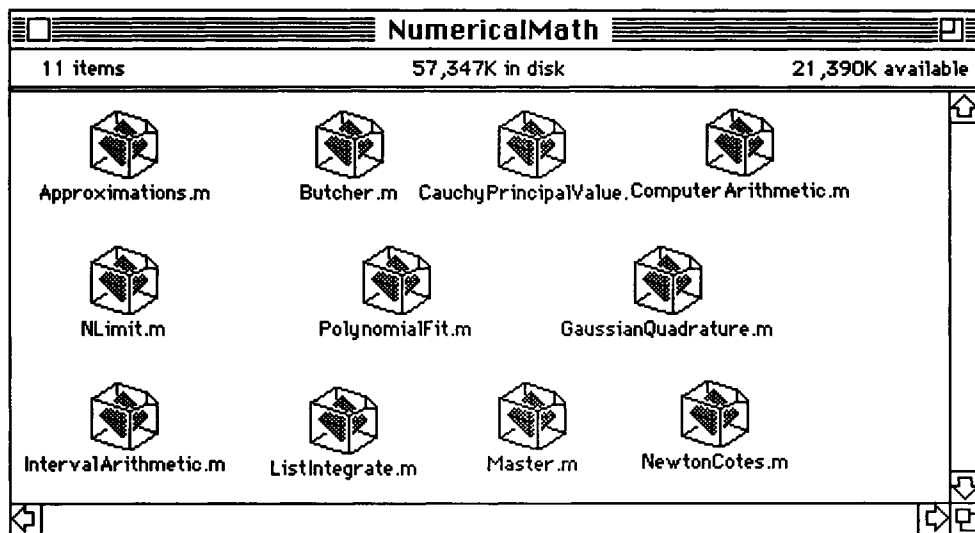
100%

Chapter 9 Some Special Packages

Chapter 9 discusses some of the more specialized packages available with *Mathematica*.

● Numerical Math

The packages within the **Numerical Math** folder in Version 2.0 are shown below:



● 9.1 Approximations.m

- o Although the examples done here were completed with Version 2.0, Version 1.2 also contains the package **Approximations.m** in the **Numerical Math** folder.

The package **Approximations.m** contains useful commands for the approximation of functions with rational functions. The first command discussed is that of

RationalInterpolation[function, {x,m,n}, {x,x0,x1}, options] which gives the interpolating rational function $P(x,y)/Q(x,y)$ on the interval from x_0 to x_1 where the the degree of $P(x,y)$ is m and that of $Q(x,y)$ is n .

- o Example:

This command is illustrated below with **Sqrt**[1-4x²]. The interpolating rational function **rint1** is found and then this approximation is compared to the original function by investigating the error function, **Abs**[**rint1** - **Sqrt**[1-4x²]].

RationalInterpolation

```

In[14]:=
<<Approximations.m
In[15]:=
rint1=RationalInterpolation[Sqrt[1-4 x^2],
  {x,2,2},{x,-.5,.5}]
Out[15]=
      -21
(1. + 8.3759224159064725 10 x -
      2
3.577708763999663515 x ) /
      -20
(1 + 2.710505431213761085 10 x -
      2
1.689164944001345945 x )
In[16]:=
approx[x_]=Chop[rint1]
Out[16]=
      2
1. - 3.577708763999663515 x
-----
      2
1 - 1.689164944001345945 x

```

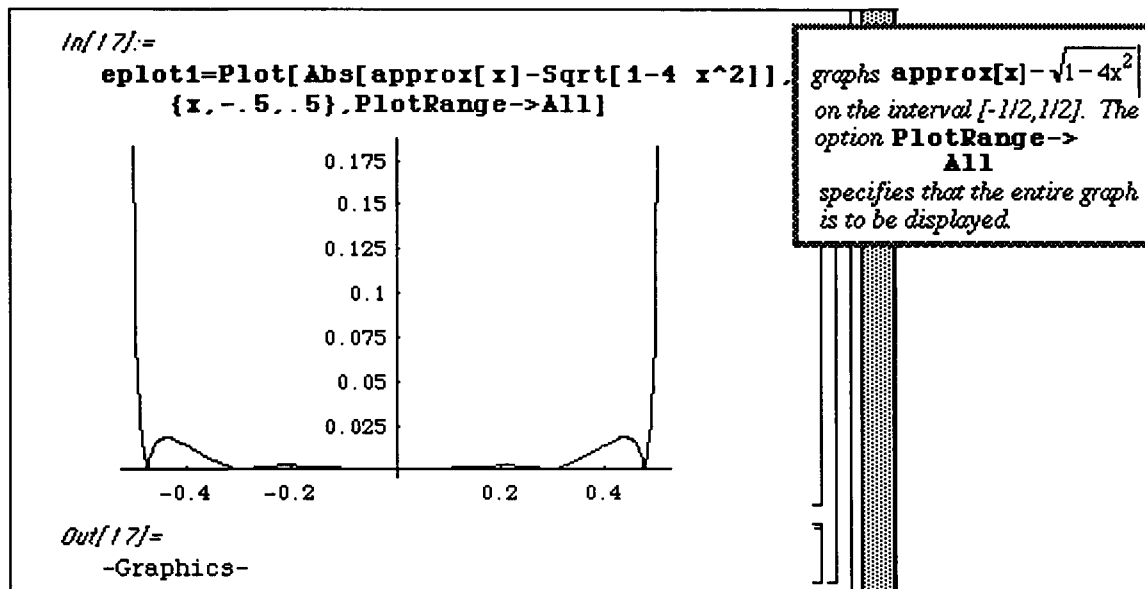
The command **RationalInterpolation** is contained in the package **Approximations.m**

rint1 is a rational approximation of $\sqrt{1-4x^2}$ on the interval $[-1/2, 1/2]$. Both the numerator and denominator have degree 2.

Notice that the coefficient of x in both the numerator and denominator is "small".

Chop[**rint1**] replaces the "small" numbers by zero.

This error, `eplot1`, is graphed below.



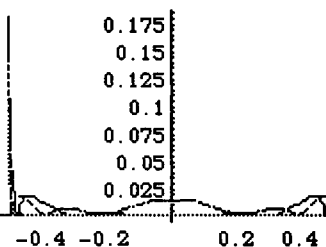
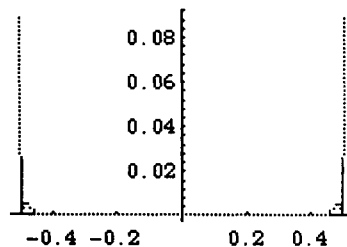
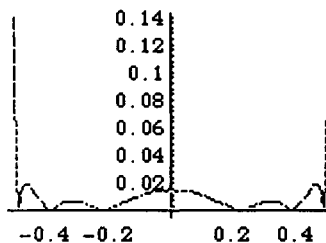
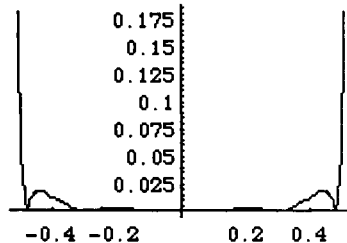
Several other interpolating rational functions are computed below. `rint2` is a function of order 3 in both the numerator and denominator. Similarly, `rint3` is of order 4 in both the numerator and denominator. The output of these functions is suppressed. Finally, the error for each rational interpolating function is graphed in order to compare accuracy. Clearly, `rint3` yields the best approximation of the three.

```

In[36]:=
  rint2=RationalInterpolation[Sqrt[1-4 x^2].{x,3,3},
    {x,-.5,.5}];
  rint3=RationalInterpolation[Sqrt[1-4 x^2].{x,4,4},
    {x,-.5,.5}];

In[44]:=
  eplot2=Plot[Abs[rint2-Sqrt[1-4 x^2]].{x,-.5,.5},
    PlotStyle->GrayLevel[.2],
    PlotRange->All,
    DisplayFunction->Identity];
  eplot3=Plot[Abs[rint3-Sqrt[1-4 x^2]].{x,-.5,.5},
    PlotStyle->GrayLevel[.3],
    PlotRange->All,
    DisplayFunction->Identity];
  all=Show[eplot1,eplot2,eplot3,
    DisplayFunction->Identity];
  Show[GraphicsArray[{{eplot1,eplot2},{eplot3,all}}]]

```



rint2 and rint3 are degree 3 and degree 4, rational approximations of $\sqrt{1-4x^2}$.

In each case, the resulting error is graphed and then all three error graphs are shown simultaneously. Finally, all four graphs are shown in a graphics array.

```

Out[44]=
  -GraphicsArray-

```

Also located in the `Approximations.m` package is the command

`MiniMaxApproximation[f[x], {x, {x0, x1}, m, k}]` which improves on the approximation found with `RationalInterpolation[f[x], {x, m, k}, {x, x0, x1}]`. Although the syntax differs, both of these commands yield a rational function of order `m` in the numerator and `k` in the denominator which approximates `f[x]` over the interval `{x0, x1}`. Another difference in the commands occurs in the output. While `RationalInterpolation` gives only the approximating rational function, `MiniMaxApproximation` gives some additional information of the form `{abscissalist, {approx, maxerror}}` where `abscissalist` is a list of abscissa at which the maximum error occurs, `approx` is the desired rational approximation, and `maxerror` is the value of the minimax error.

○ Example:

These two commands are investigated below with `f[x]=Exp[x]` on the interval `{-1, 1}` using a rational function of order 2 in the numerator and 1 in the denominator. First, the approximating rational function is found with `RationalInterpolation` and named `rint1`. Note that this command yields only the approximating function as output. Next, an approximation is found with `MiniMaxApproximation`. These results (which are given in the form mentioned earlier) are named `mmax1`.

```

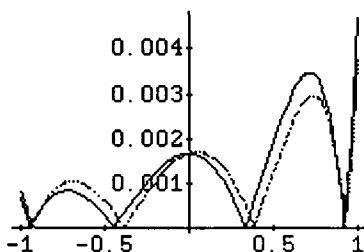
MiniMaxApprox
In[1]:=
<<Approximations.m
In[2]:=
rint1=RationalInterpolation[
  Exp[x], {x, 2, 1}, {x, -1, 1}]
Out[2]=
          2
1.00168 + 0.678005 x + 0.164056 x
-----
          1 - 0.322666 x
In[3]:=
mmax1=MiniMaxApproximation[
  Exp[x], {x, {-1, 1}, 2, 1}]
Out[3]=
{{-1., -0.743577, -0.067892, 0.679364,
  1.}, {
          2
1.00167 + 0.673354 x + 0.16033 x
-----
          1 - 0.325977 x
-0.00172826}}
```

`rint1` is a rational approximation of e^x on the interval $[-1, 1]$. The degree of the numerator is 2; the degree of the denominator is 1.

The command `MiniMaxApproximation` not only computes a rational approximation but also gives a list of numbers for which the maximum error occurs. Hence, in this case, the maximum error occurs when x is $-1, -0.743577, -0.067892, 0.679364$, and 1 .

In order to work with the approximating function obtained with `MiniMaxApproximation`, the technique of extracting an element from a list must be used. Since `mmax1` is a list of two parts, the second of which contains the approximating function, `mmax1[[2,1]]` yields the desired rational function from the list. This function is called `apx1`. After extracting the approximating function `apx1`, the error for both approximations is investigated. This is done by observing the error function for each. The function `rint1error` which represents the error of the `RationalInterpolation` approximation is given by `Abs[rint1-Exp[x]]`. Likewise, the error of the `MiniMaxApproximation`, called `mmax1error`, is given by `apx1-Exp[x]`. These error functions are plotted simultaneously below. Notice how the critical points of the `mmax1error` curve (graphed in the lighter print) correspond to the values in `abscissalist`.

```
In[18]:=
errorrint=Plot[Abs[Exp[x]-rint1],
{x,-1,1},PlotRange->All,
PlotStyle->GrayLevel[.3],
DisplayFunction->Identity];
errormmax=Plot[Abs[Exp[x]-mmax1[[2,1]]],
{x,-1,1},PlotRange->All,
DisplayFunction->Identity];
Show[errorrint,errormmax,
DisplayFunction->$DisplayFunction]
```



```
Out[18]=
-Graphics-
```

`errorrint` is a graph of the error between `Exp[x]` and `rint1`. Remember that the option `DisplayFunction->Identity` specifies that the resulting graphics object not be displayed. Similarly, `errormmax` is a graph of the error between `Exp[x]` and `mmax[[2,1]]`.

Finally, both graphs are shown simultaneously.

The option `DisplayFunction->$DisplayFunction` indicates that the resulting graphics objects are to be displayed. The lighter graph represents `errorrint`.

● 9.2 GaussianQuadrature.m

- The package **GaussianQuadrature.m** is contained only in the Version 2.0 **Numerical Math** and is not included with Version 1.2.

Numerical integration by Gaussian quadrature is based on the Lagrange interpolation formula

$$p(x) = \sum_{i=1}^n f(x_i) \ell_i(x) \quad \text{where} \quad \ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^n \left(\frac{x - x_j}{x_i - x_j} \right).$$

If this formula provides a good approximation of f , then the integral of p yields a good approximation to the integral of f . Therefore,

$$\int_a^b f(x) dx \approx \int_a^b p(x) dx = \sum_{i=1}^n f(x_i) \int_a^b \ell_i(x) dx = \sum_{i=1}^n A_i f(x_i)$$

where A_i represents the weights and x_i the nodes for $i = 1, 2, \dots, n$.

The *Mathematica* command **GaussianQuadratureWeights[n, a, b]** which is located in the **GaussianQuadrature.m** package determines the values of these weights and nodes. The output is given in the form of ordered pairs where the first entry in each pair gives the node while the second entry represents the corresponding weight. Several examples of this command are given below for different values of n using the same interval from -2 to 2. Note that the calculation of the weights and nodes is independent of the function f .

- **Example:**

```

In[1]:=
<<GaussianQuadrature.m

In[2]:=
GaussianQuadratureWeights[2, -2, 2]

Out[2]=
{{-1.1547, 2.}, {1.1547, 2.}}

In[3]:=
GaussianQuadratureWeights[3, -2, 2]

Out[3]=
{{-1.54919, 1.11111}, {0., 1.77778}, {1.54919, 1.11111}}

In[4]:=
GaussianQuadratureWeights[4, -2, 2]

Out[4]=
{{-1.72227, 0.69571}, {-0.679962, 1.30429}, {0.679962, 1.30429},
{1.72227, 0.69571}}

```

Since the output appears in the form of a list, the weights and nodes can be extracted from the output. This is illustrated below by assigning the name of `gqw` to the expression which results from the command `GaussianQuadratureWeights[3, -1, 1]`. Hence, `gqw[[1, 1]]` gives the first node on the interval $[-1, 1]$, and `gqw[[1, 2]]` gives the weight which corresponds to this node. Therefore, the integral of a function $f(x)$ from -1 to 1 can be approximated with the Gaussian quadrature formula given earlier using the values obtained with `gqw`. This is done in `gqint` below. Since no function is specified, the general integral formula results.

○ Example:

In the next command, however, the function $f[x]=\text{Exp}[-(\text{Cos}[x])^2]$ is defined. Thus, a numerical approximation of the integral of f from $x = -1$ to $x = 1$ is given.

<pre> In[5]:= gqw=GaussianQuadratureWeights[3, -1, 1] Out[5]= {{-0.774597, 0.555556}, {0., 0.888889}, {0.774597, 0.555556}} In[6]:= gqw[[1, 1]] Out[6]= -0.774597 In[7]:= gqw[[1, 2]] Out[7]= 0.555556 In[8]:= gqint=Sum[gqw[[i, 2]] f[gqw[[i, 1]]], {i, 1, 3}] Out[8]= 0.555556 f[-0.774597] + 0.888889 f[0.] + 0.555556 f[0.774597] In[10]:= f[x_]= Exp[-(Cos[x])^2]; gqint1=Sum[gqw[[i, 2]] f[gqw[[i, 1]]], {i, 1, 3}] Out[10]= 0.993687 </pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p><code>gqw</code> is a nested list of three elements. Each element of <code>gqw</code> is a list with two elements where the first element represents the node and the second element represents the weight.</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p><code>gqw[[1]]</code> would yield the first element of <code>gqw</code> which is <code>{-0.774597, 0.555556}</code>.</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p>After defining $f(x) = e^{-\text{Cos}^2(x)}$, we use the weights and nodes from above to estimate $\int_{-1}^1 f(x) dx$.</p> </div>
---	--

A function which evaluates the Gaussian quadrature for any value of n is defined below in `gauss[n]`. A table of approximations of the integral of f given above is then created for $n = 2$ to $n = 10$ and placed in `TableForm`. This procedure is useful in comparing the approximations obtained with Gaussian quadrature and can be repeated for other integrals.

```

In[1]:=
  gauss[n_]:=
  Module[{k=n},
    weights=N[GaussianQuadratureWeights[k,-1,1]];
    quad=Sum[weights[[i,2]] f[weights[[i,1]]],
             {i,1,k}]
  ]

```

```

In[2]:=
  Table[{i,N[gauss[i]]},{i,2,10}]/TableForm

```

```

Out[2]//TableForm=

```

```

  2    0.991091
  3    0.993687
  4    0.991297
  5    0.991294
  6    0.991308
  7    0.991308
  8    0.991308
  9    0.991308
 10    0.991308

```

We can create a table of approximations of $\int_{-1}^1 f(x) dx$ for various values of n .

Gaussian Quadrature procedures are useful when the exact value of an integral cannot be computed. This is the case

with $\int_{-1}^1 e^{-\cos^2(x)}$ as shown below. However, in this case, our results can be verified

with **NIntegrate**:

```

In[2]:=
f[x_]:=Exp[-(Cos[x])^2];
Integrate[f[x],{x,-1,1}]

Syntax::bktwrn:
Warning: "f (b+a x^dg)" should probably be "f [b+a x^dg]".
(line 169 of "Integrate`mainalgorithm`")

Syntax::bktwrn:
Warning: "f (Denominator[a]^Abs[b])" should probably be
"f [Denominator[a]^Abs[b]]".
(line 1227 of "Integrate`mainalgorithm`")

Out[2]:=
Integrate[E-Cos[x]2, {x, -1, 1}]

In[3]:=
NIntegrate[f[x],{x,-1,1}]

Out[3]:=
0.991308

```

Mathematica cannot compute the exact value of
 $\int_{-1}^1 f(x) dx$
*although a numerical approximation which agrees with the above can be calculated using **NIntegrate**.*

100% ▾

● 9.3 NLimit.m

- **NLimit.m** is included with Version 2.0 but not earlier versions.

The **NLimit.m** package contains which are useful in the calculation of limits and derivatives. These are **NLimit[f[x], x->x0]** and **ND[f[x], x, x0]**. **NLimit[f[x], x->x0]** computes the numerical limit of $\lim_{x \rightarrow x_0} f[x]$.

The value of **x0** can be either **Infinity** or **-Infinity**. However, a limit is not given with **NLimit** if the limit is **Infinity** or **-Infinity**. This command may be used when the built-in function **Limit[f[x], x->x0]** fails.

- **Example:**

Compute $\lim_{x \rightarrow \infty} \frac{e^x}{x!}$.

Illustrated below is the calculation of the limit as **x** approaches **Infinity** of the function **Exp[x]/(x!)**. First, the limit is attempted using **Limit**. Since this is unsuccessful, a second attempt is made with **NLimit** to yield a limit of 0.

```

In[1]:=
<<NLimit.m

In[3]:=
Limit[Exp[x]/(x!), x->Infinity]
Series::esss:
Essential singularity encountered
      1
in Gamma[- + <<2>>].
      x

General::stop:
Further output of Series::esss
will be suppressed during this
calculation.

Out[3]=
      x
      E
Limit[---, x -> Infinity]
      x!

In[4]:=
NLimit[Exp[x]/(x!), x->Infinity]

Out[4]=
0.

```

*Mathematica is unable to compute the limit of $\frac{e^x}{x!}$ with the command **Limit**.*

*In this case, **NLimit** can be used to compute the value of $\lim_{x \rightarrow \infty} \frac{e^x}{x!}$.*

Similarly, **Limit** is unsuccessful in computing the limit as x approaches **Infinity** of the function $(x^5)/\text{Exp}[x]$. However, **NLimit** is used to obtain a numerical approximation of this limit. Note that the result is quite close to 0. The built-in *Mathematica* command **Chop[expression]** replaces all approximate real numbers in **expression** which are less than $10^{(-10)}$ in magnitude with the number 0. Hence, **Chop[%]** yields the correct limit.

<pre>In[6]:= Limit[(x^5)/Exp[x], x->Infinity]</pre>]]	<p>Similarly, even though <i>Mathematica</i> cannot compute</p>
<pre>Out[6]= 5 x Limit[---, x -> Infinity] x E</pre>]]	<p>Lim $\frac{x^5}{e^x}$ $x \rightarrow \infty$ using the command Limit,</p>
<pre>In[7]:= NLimit[(x^5)/Exp[x], x->Infinity]</pre>]]	<p>the command NLimit can be used to approximate the limit.</p>
<pre>Out[7]= -19 1.35525 10</pre>]]	
<pre>In[8]:= Chop[%]</pre>]]	<p>Chop[%] produces zero since</p>
<pre>Out[8]= 0</pre>]]	<p>$1.35525 \cdot 10^{-19} < 10^{-10}$.</p>

o Example:

Special care must be taken when dealing with limits which achieve the value of **Infinity** or **-Infinity**. **NLimit** cannot calculate limits of this type ! In these cases, **NLimit** may yield an incorrect limit or no limit at all. In the first example which follows, **NLimit** gives a value which obviously is not the limit of the given function. The limit of this function is **Infinity** as substantiated with the graph which follows as well as with well-known properties of functions of this type. In the second example, **NLimit** does not compute the limit as **x** approaches **Infinity** of the function x^2 . This limit is clearly **Infinity**.

NumericalLimits

```

NLimit[Exp[x]/(x^5), x->Infinity]
2.71828
Plot[Exp[x]/(x^5), {x, 80, 100}]

```

Here **NLimit** yields an incorrect value since $\lim_{x \rightarrow \infty} \frac{e^x}{x^5} = \infty$.

```

NLimit[x^2, x->Infinity]
2
NLimit[x, x -> Infinity]

```

NLimit is unable to compute $\lim_{x \rightarrow \infty} x^2 = \infty$.

When dealing with certain functions, the calculation of a numerical derivative may be necessary. The command **ND[f[x], x, x0]** gives the numerical approximation of $f'(x_0)$. This method is based on Richardson extrapolation and does not yield a formula for the derivative. It simply gives an approximation of the derivative at the value $x = x_0$.

○ **Example:**

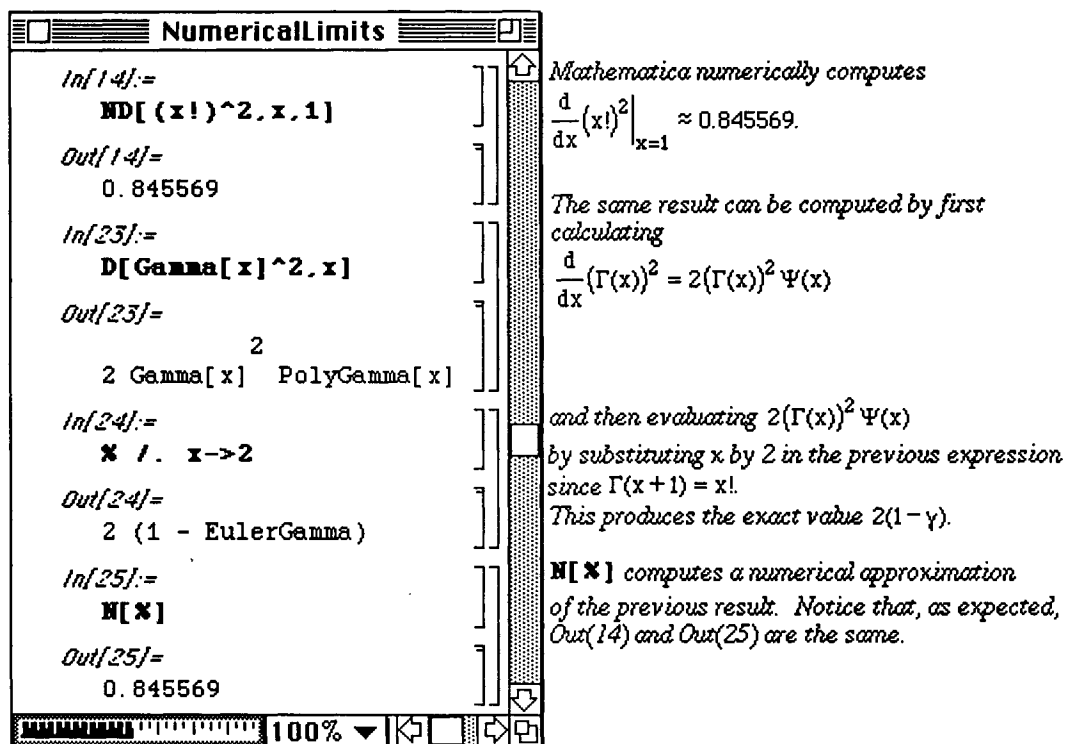
The derivative of $(x!)^2$ at $x = 1$ is approximated below. Next, an attempt to illustrate the accuracy of `ND[f[x], x, x0]` is performed with the function `Tan[x]`.

The function $(x!)^2$ is the same as the function $(\Gamma(x))^2$ where $\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$

$$= \lim_{n \rightarrow \infty} \frac{n! n^x}{x(x+1)\dots(x+n)}$$

is the Gamma function which is given by the command `Gamma[a]`.

$\Psi(x) = \Gamma'(x)$ is called the Digamma Function and is given by the command `PolyGamma[x]`.



NumericalLimits

Mathematica numerically computes

$$\frac{d}{dx} (x!)^2 \Big|_{x=1} \approx 0.845569.$$

The same result can be computed by first calculating

$$\frac{d}{dx} (\Gamma(x))^2 = 2(\Gamma(x))^2 \Psi(x)$$

and then evaluating $2(\Gamma(x))^2 \Psi(x)$ by substituting x by 2 in the previous expression since $\Gamma(x+1) = x!$. This produces the exact value $2(1-\gamma)$.

N[x] computes a numerical approximation of the previous result. Notice that, as expected, `Out(14)` and `Out(25)` are the same.

Since the derivative of `Tan[x]` is known to be `(Sec[x])^2`, the numerical values obtained using `ND` are compared to the values of `(Sec[x])^2` for $x = -1.5$ to $x = 1.5$. A table of numerical approximations of the derivative of `Tan(x)` at values of x in the interval $[-1.5, 1.5]$ using `ND` are given in `dtable`.

This table of ordered pairs is created in order to plot the approximated derivative. These points are plotted simultaneously with $(\text{Sec}[x])^2$ to show the accuracy of this approximation.

```

In[4]:=
dtable=Table[{t,ND[Tan[x],x,t]},
             {t,-1.5,1.5,.1}];
Short[dtable,3]

Out[4]:=Short=
{{-1.5, 197.742}, {-1.4, 34.5964},
 {-1.3, 13.974}, <<26>>, {1.4, 27.5755},
 {1.5, 1556.58}}

In[10]:=
plot1=ListPlot[dtable,
               DisplayFunction->Identity];
plot2=Plot[(Sec[x])^2,{x,-1.5,1.5},
           DisplayFunction->Identity,
           PlotStyle->GrayLevel[.4]];
Show[plot1,plot2,
     DisplayFunction->${DisplayFunction}]

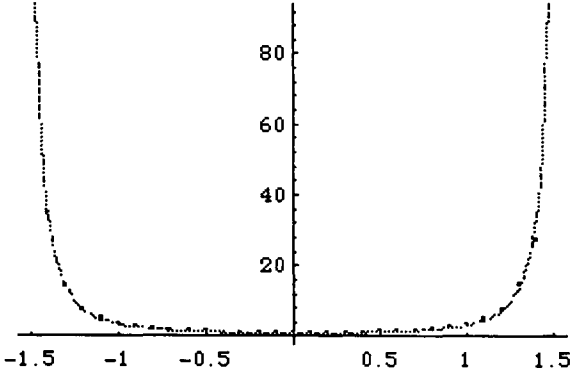
```

dtable is a table consisting of the ordered pairs $\{t, \text{ND}[\text{Tan}[x], x, t]\}$ for $t = -1.5, -1.4, -1.3, \dots, 1.4, 1.5$.

The command **Short[dtable, 3]** prints an abbreviated form of **dtable** on no more than three lines.

The command **ListPlot** is used to plot the table of points **dtable**. Remember that with Version 2.0, a semi-colon must be placed after each command for which the output is to be suppressed.

The **Show** command is used to display both graphs simultaneously.



```

Out[10]=
-Graphics-

```

□ Displaying Points with Versions 1.2 and 2.0:

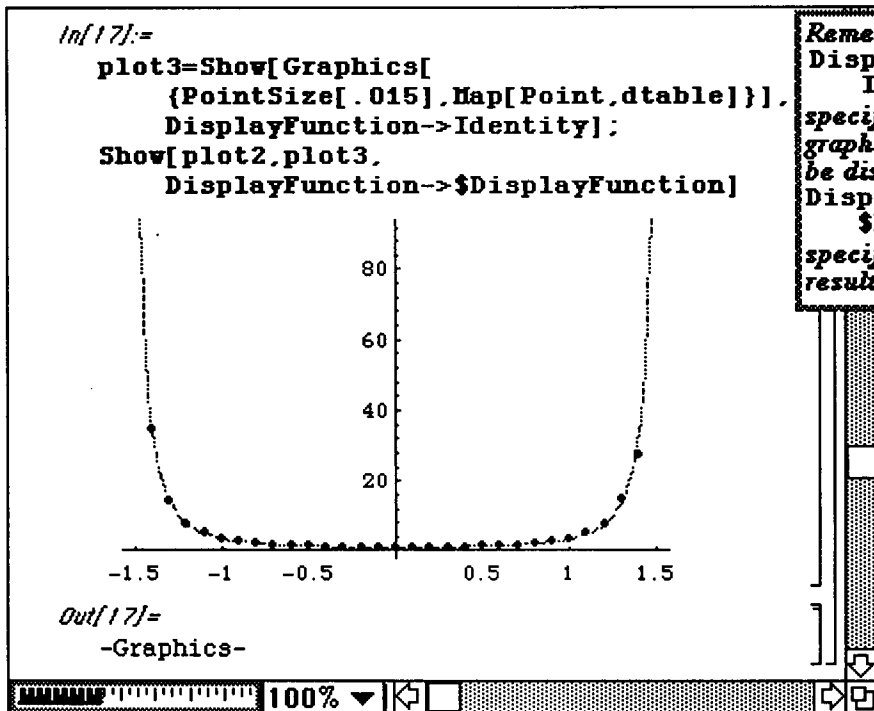
Since the points are difficult to see in the previous plot, the command **PointSize[n]** is used to increase the size of the points. In order to increase their size, the ordered-pairs must be specified as **points**. Given an ordered pair $\{a, b\}$, **Point[{a,b}]** specifies the graphics object which represents a point with coordinates $\{a, b\}$. The graphics object may then be displayed with **Show[Point[{a,b}]]**. Therefore, the command

```

Map[Point, dtable] produces the list of points
{Point[{-1.5, 197.742}], Point[{-1.4, 34.5964}], ...,
 Point[{1.5, 1556.58}]}.

```

Given a list of points `orderedpairs`, `Show[Graphics[PointSize[n], orderedpairs]]` displays `orderedpairs` according to the size given by `PointSize[n]`. Consequently, `plot3` represents the points created by `Map[Point, dtable]` displayed in size `.015`. The resulting graph is not displayed because the option `DisplayFunction->Identity` is included. Instead both `plot2` and `plot3` are displayed together by including the option `DisplayFunction->$DisplayFunction`.



Remember that the option `DisplayFunction->Identity` specifies that the resulting graphics object is not to be displayed; the option `DisplayFunction->$DisplayFunction` specifies to show the resulting graphics object.

● 9.4 PolynomialFit.m

Version 2.0 includes the package `PolynomialFit.m` which offers the command `PolynomialFit[dataList, n]` that can be used to approximate a list of data in `dataList` with a polynomial of degree `n`. The evaluation of the approximating polynomial is based on Chebyshev polynomials and the formula for this polynomial is not given as output as it was with the built-in `Fit` command. Instead, the command `PolynomialFit[dataList, n]` yields the true function, `FittingPolynomial[<>, n]` which can be used to investigate the accuracy of the approximation. (Note that `dataList` is not a list of ordered pairs.)

A list of data is given below in `values`. After loading this package, this data is approximated with a polynomial of degree 2. This polynomial is named `approx1` and is found with `PolynomialFit[values, 2]`. The true function `approx1` can be plotted as shown in `plot1`. (Note that square brackets must be used with `approx1` in the `Plot` command.) It can also be evaluated for any value of `x` as illustrated below with `approx1[4.5]`.

PolynomialApprox
⏏

```

In[42]:=
  <<PolynomialFit.m

In[43]:=
  values={1.2, -.3, .5, 2.3, 1.7, .5};

In[44]:=
  approx1=PolynomialFit[values, 2]

Out[44]=
  FittingPolynomial[<>, 2]

In[45]:=
  plot1=Plot[approx1[x], {x, 0, 6}]

Out[45]=
  -Graphics-

In[46]:=
  approx1[4.5]

Out[46]=
  1.24652

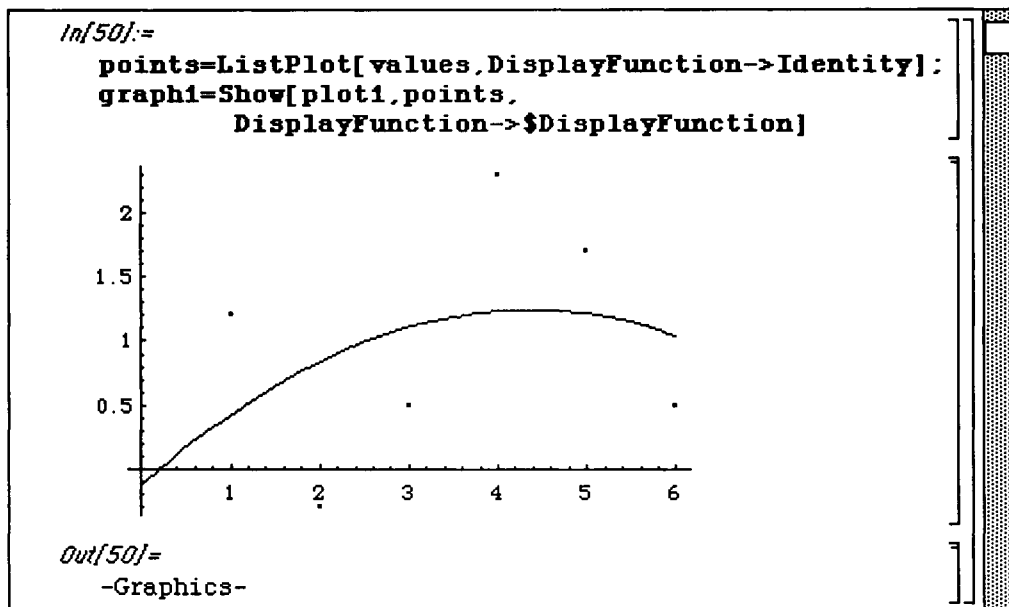
```

computes `approx1[4.5]`.
 The same result would be obtained by entering
`approx1[x] /. x -> 4.5`.

50%
⏏

Defines `values`
 to be the list consisting of
`{1.2, -.3, .5, 2.3, 1.7, .5}`. The
 semi-colon placed at the end
 of the command suppresses
 the resulting output.
`approx1` is a function;
 thus,
`Plot[approx1[x],
 {x, 0, 6}]`
 graphs the second degree
 polynomial for values of `x`
 between 0 and 6.

In order to investigate how well the approximating polynomial `approx1` fits the data in `values`, the two are plotted simultaneously. Recall that the `ListPlot` option `DisplayFunction->Identity` causes the graph of the data points in `points` to be suppressed initially. Then, `DisplayFunction->$DisplayFunction` is used in the `Show` command to display `plot1` and `points`.



Since the second order polynomial determined above does not yield an accurate approximation, the polynomial of order 4 is computed below. This polynomial is called `approx2` and is found with `PolynomialFit[values, 4]`. In a manner similar to that used above, the data and the approximating polynomial are plotted together.

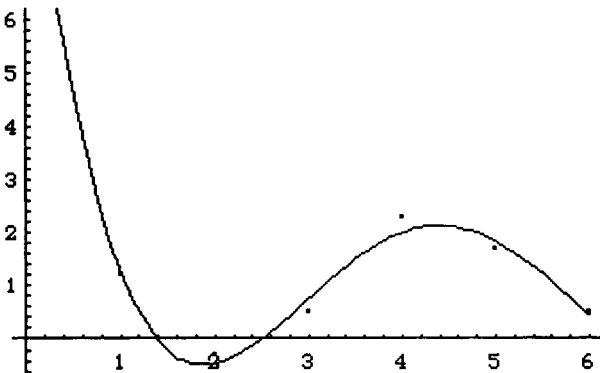
```
In[51]:=
  approx2=PolynomialFit[values, 4]
```

approximates the list values with a polynomial of degree 4.

```
Out[51]=
  FittingPolynomial[<>, 4]
```

```
In[55]:=
  plot2=Plot[approx2[x], {x, 0, 6},
    DisplayFunction->Identity];
  graph2=Show[plot2, points,
    DisplayFunction->$DisplayFunction]
```

The resulting polynomial `approx2[x]` is graphed on the interval `{0,6}` and shown simultaneously with the graph points.



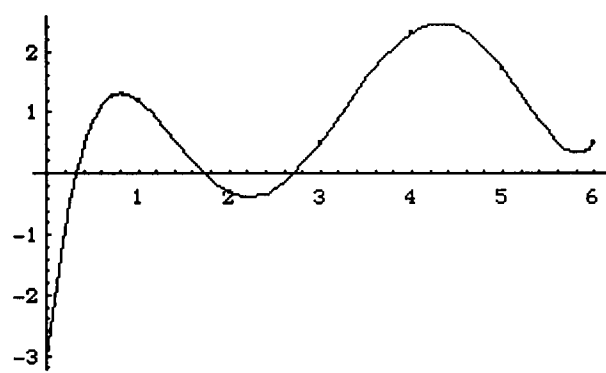
```
Out[55]=
  -Graphics-
```

In an attempt to improve on the approximation, the sixth order polynomial, **approx3**, is found and plotted below. This appears to be an accurate approximation of the data after plotting all three simultaneously.

```

In[58]:=
approx3=PolynomialFit[values,6];
plot3=Plot[approx3[x].{x,0,6}.
DisplayFunction->Identity];
graph3=Show[plot3,points,
DisplayFunction->${DisplayFunction}]

```



```

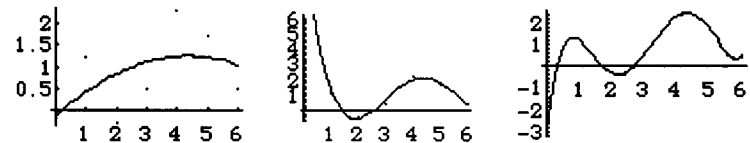
Out[58]=
-Graphics-

```

```

In[59]:=
Show[GraphicsArray[{graph1,graph2,graph3}]]

```



```

Out[59]=
-GraphicsArray-

```

*In the same manner as above, **approx3** is a polynomial of degree 6 that approximates the table of numbers **values**.*

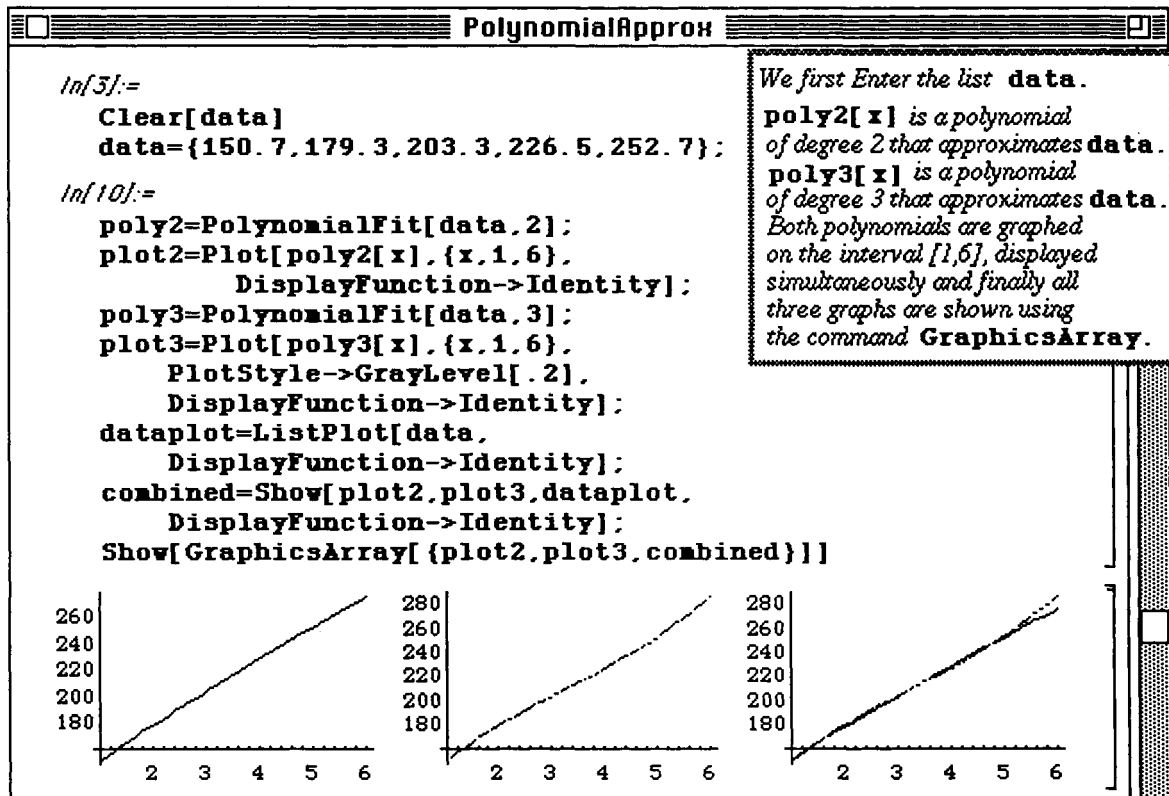
All three graphs of approximations and points are shown to illustrate that the higher the degree of the fit, the better it is.

o **Example:**

The determination of an approximating polynomial has many applications. For instance, this method can be used on existing data to estimate future values such as population. Consider the following census data:

Year	Population (million)
1950	150.7
1960	179.3
1970	203.3
1980	226.5
1990	252.7

This data are entered as **data** below. Polynomials of order 2 and 3 are calculated and plotted below along with the data points. Both polynomials, **poly2** and **poly3**, appear to be accurate approximations. Since **ListPlot** is used to plot the data points, one unit along the x-axis represents 10 years.



`poly2[6]` and `poly3[6]` yield the approximate population in the year 2000 based on each polynomial. These values are given below.

```

In[11]:=
  poly2[6]

Out[11]=
  275.06

In[12]:=
  poly3[6]

Out[12]=
  285.7

```

`poly2[6]`
*evaluates `poly2[x]`
 for $x=6$. The same result could
 be obtained by entering
`poly2[x] /. x->6`*

`poly3[6]`
*evaluates `poly3[x]`
 for $x=6$. The same result could
 be obtained by entering
`poly3[x] /. x->6`*

■ 9.5 RungeKutta.m

- o In Version 1.2, **RungeKutta.m** is contained in the **Numerical Math** folder; in Version 2.0 **RungeKutta.m** is contained in the **Programming Examples** folder.

The Runge-Kutta method is used to numerically approximate the solution of the initial value problem

$$y' = f(x, y)$$

$$y(x_0) = y_0,$$

$$\text{at } x_1 = x_0 + h, x_2 = x_1 + h, \dots$$

This method can be used with systems of equations as well as a single equation. The steps towards solving ordinary differential equations of this form by the Runge-Kutta method are given in the package **Runge-Kutta.m** in the Numerical Methods folder located in **Packages**. It should be noted that the package be used to solve those differential equations which are classified as nonstiff. Note the following definition of a stiff differential equation.

The differential equation $y' = f(x, y)$ is stiff if the ratio of the largest eigenvalue to the smallest eigenvalue of the matrix $\frac{\partial f}{\partial y}(x, y)$ is very large.

The syntax for the command is given as follows:

```
RungeKutta [ListOfODEexpressions, ListOfVariables,  
             ListOfInitialValues,  
             Finalx, ErrorTolerance, (options...) ]
```

where **ListOfODEexpressions** is made up of the components of $f(x, y)$, **ListOfVariables** is the list of both independent and dependent variables, **ListOfInitialValues** is the initial vector (including the initial value of the independent variable x), **Finalx** is the value of x at which a solution is sought, and **ErrorTolerance** is the local error tolerance which must be met at each integration. In giving **ListOfODEexpressions**, **ListOfVariables**, and **ListOfInitialValues**, the variables must be entered in the same order in which the system of differential equations is stated with the independent variable listed first.

The **RungeKutta** options are given below along with their default values :

```
WorkingPrecision->Precision[N[1] ],  
MaximumStepSize->Finalx-Initialx,  
InitialStepSize->MaximumStepSize,  
ProgressTrace->False .
```

ProgressTrace->**True** causes the value of x , the step-size, and the local error to be given as the calculation is performed.

□ Example:

The following example illustrates how the package is used to solve the following initial value problem:

$$\begin{cases} y' = 2x + y \\ y(0) = 1 \end{cases}$$

Notice that the differential equation must be entered in the form of a list even though there is only one equation. **ErrorTolerance** is selected to be 10^{-5} and the solution, y , is found at values of x between 0 and 1. Since a value of **False** is assumed by **ProgressTrace**, no progress report is given. The elements given in the output are of the form $\{x, y\}$ where y is the approximate solution to the differential equation at x .

```

In[20]:=
  <<RungeKutta.m
In[21]:=
  sol1=RungeKutta[{2 x +y},{x,y},{0,1},1,10^(-5),
    WorkingPrecision->20, InitialStepSize->.1,
    MaximumStepSize->0.2, ProgressTrace->False]
Out[21]=
  {{0., 1.}, {0.1, 1.115512755}, {0.3, 1.4495764739790405334},
    {0.5, 1.946163936143421276}, {0.7, 2.6412583489668909842},
    {0.9, 3.5788097018513151164}, {1., 4.154845894401927461}}

```

If **True** is used with **ProgressTrace**, the following output is given:

```

In[23]:=
  RungeKutta[{2 x + y}, {x, y}, {0.1}, 1, 10^(-5),
    WorkingPrecision->20, InitialStepSize->.1,
    MaximumStepSize->0.2, ProgressTrace->True]
RungeKutta::progress x = 0., step = 0.1, local error =
      -8
  2.08760499561 10
RungeKutta::progress x = 0.1, step = 0.2, local error =
      -7
  5.452752959092 10
RungeKutta::progress x = 0.3, step = 0.2, local error =
      -7
  4.960625430711 10
RungeKutta::progress x = 0.5, step = 0.2, local error =
      -7
  4.464407969928 10
RungeKutta::progress x = 0.7, step = 0.2, local error =
      -7
  4.024343595717 10
RungeKutta::progress x = 0.9, step = 0.1, local error =
      -8
  1.37858326798 10
Out[23]=
  {{0., 1.}, {0.1, 1.115512755}, {0.3, 1.4495764739790405334},
    {0.5, 1.946163936143421276}, {0.7, 2.6412583489668909842},
    {0.9, 3.5788097018513151164}, {1., 4.154845894401927461}}

```

□ Example:

As stated earlier, the **RungeKutta** command can also be used to solve systems of first-order ordinary differential equations. Consider the following initial value problem :

$$\begin{cases} y' = z \\ z' = 2xz - 4y \\ y(0) = 1, z(0) = 0 \end{cases}$$

Here there are two equations which are entered as the list $\{z, 2xz - 4y\}$ for **ListOfODEexpressions** in the **RungeKutta** command. The variables must, therefore, be given as the list $\{x, y, z\}$. The initial conditions are given for y and z at $x = 0$, so **ListOfInitialValues** is the list $\{0, 1, 0\}$ which represents the initial values for the variables $\{x, y, z\}$, respectively. As in the previous example, the local tolerance is taken to be 10^{-5} and the solution is approximated for values of x between 0 and 1. Each element in the output consists of the value of x and the approximate values of y and z found with this procedure.

```
In[24]:=
sol2=RungeKutta[ {z, 2 x z-4 y}, {x,y,z}, {0,1,0}, 1, 10^-5,
WorkingPrecision -> 20, InitialStepSize -> 0.1,
MaximumStepSize -> 0.2, ProgressTrace -> False ]

Out[24]=
{{0., 1., 0.}, {0.1, 0.9799999817919399506, -0.4000000190929186766},
{0.2652985946997730703, 0.8592329761222134417,
-1.0611951756237893048}, {0.4353952520609771713,
0.6208612134185668668, -1.7415828078592385648},
{0.6258389127874684977, 0.2166499887699848187,
-2.5033597842530430796}, {0.8258389127874684977,
-G.3640218792002295708, -3.3033631282921385819},
{1., -1.0000032370017912819, -4.0000094146543892189}}
```

If less precision is desired, a smaller value can be used for **WorkingPrecision**. The following command shows how this change affects the results obtained with the previous command. (A value of **10** is used below as compared to **20** in the previous example.)

```
In[25]:=
  RungeKutta[ {z, 2 x z-4 y}, {x,y,z}, {0.1,0}, 1, 10^-5,
    WorkingPrecision -> 10, InitialStepSize -> 0.1,
    MaximumStepSize -> 0.2, ProgressTrace -> False ]

Out[25]=
  {{0., 1., 0.}, {0.1, 0.98, -0.4}, {0.265299, 0.859233, -1.0612},
    {0.435395, 0.620861, -1.74158}, {0.625839, 0.21665, -2.50336},
    {0.825839, -0.364022, -3.30336}, {1., -1., -4.00001}}
```

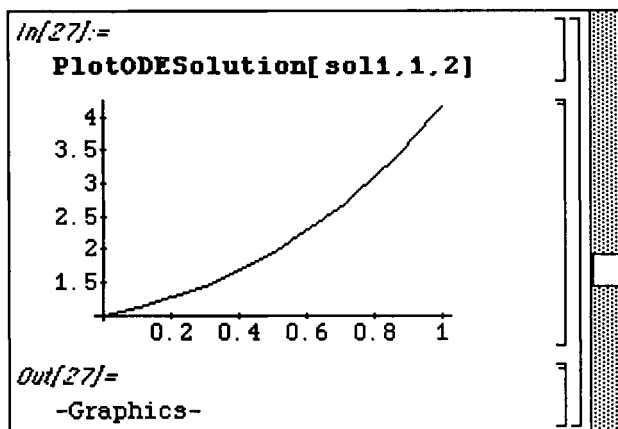
Solutions found using **RungeKutta** can easily be plotted in two dimensions with the command **PlotODESolution[solutionlist,m,n,options]** where **m** and **n** are the components of the solution vector to be plotted.

In the previous problem, the solution was represented as the list $\{x, y, z\}$. Hence, a graph of the **y** and **z** values at each **x** can be generated with **PlotODESolution[%,2,3]**. The values of **2** and **3** in this command represent **y** and **z**, respectively. The solution list to the previous problem was conveniently named **sol12** for use with **PlotODESolution**.

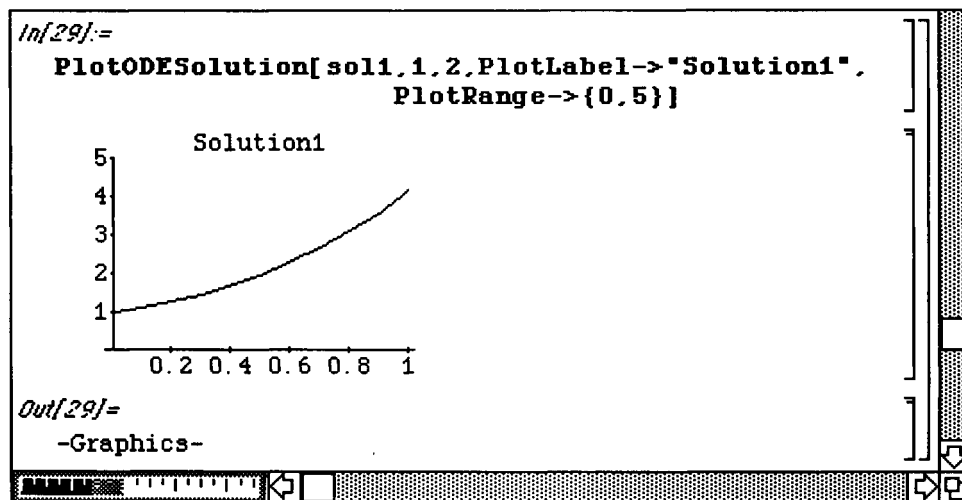
```
In[26]:=
  PlotODESolution[sol12,2,3]

Out[26]=
  -Graphics-
```

Solutions to a single ordinary differential equation can also be plotted with **PlotODESolution**. Since each element in the solution list has only two components, x and y , the command is entered in the following way to produce a graph of the approximate solution y .

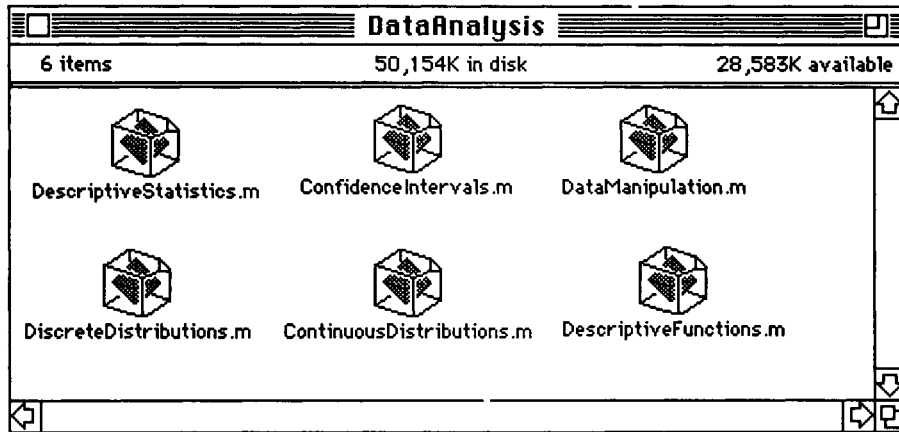


As indicated earlier, **ListPlot** options can be included in the **PlotODESolution** command. Several options are illustrated in the following command.



■ Version 1.2 Data Analysis

- o The Version 1.2 **Data Analysis** folder depicted below contains several packages useful in the area of statistics. In Version 2.0, the **Data Analysis** folder is replaced by the **Statistics** folder.



■ 9.6 Continuous Distributions.m and DescriptiveStatistics.m

The *Mathematica* packages **ContinuousDistributions.m** and **DescriptiveStatistics.m** contain many useful commands which can be used to solve problems in statistics.

- o In Version 2.0, **ContinuousDistributions.m** and **DescriptiveStatistics.m** are contained in the **Statistics** folder.

These packages are first loaded below in order that several examples can be shown.

ContinuousDistributions.m includes the following distributions: **BetaDistribution**, **CauchyDistribution**, **ChiDistribution**, **ChiSquareDistribution**, **ExponentialDistribution**, **ExtremeValueDistribution**, **FRatioDistribution**, **GammaDistribution**, **NormalDistribution**, **HalfNormalDistribution**, **LaplaceDistribution**, **LogNormalDistribution**, **LogisticDistribution**, **RayleighDistribution**, **StudentTDistribution**, **UniformDistribution**, and **WeibullDistribution**.

The mean value of a continuous distribution with probability density f is given by $\mu = \int_{-\infty}^{+\infty} xf(x)dx$.

The variance of a distribution is given by $\sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 f(x)dx$; the standard deviation is given

$$\text{by } \sigma = +\sqrt{\sigma^2} = +\sqrt{\int_{-\infty}^{+\infty} (x-\mu)^2 f(x)dx}.$$

The **DescriptiveStatistics.m** package includes several functions which can be applied to these distributions. For example, **Density[Distribution, t]** gives the density function which corresponds to **Distribution**. **Mean[Distribution]** gives the mean of **Distribution** and **Variance[Distribution]** gives the variance of **Distribution**. These values are commonly known for the Normal distribution and are given below using these newly introduced commands.

□ Example:

The package **ContinuousDistributions.m** contains definitions of many familiar continuous distributions and the package **DescriptiveStatistics.m** contains commands to extract data from various lists.

Density[CauchyDistribution[a,b],t] computes the density (as a function of t) of the Cauchy distribution.

NormalDistribution[mu,sigma] computes the mean of the normal distribution.

Variance[NormalDistribution[mu,sigma]] computes the variance of the normal distribution.

```

In[17]:=
<<ContinuousDistributions.m
In[18]:=
<<DescriptiveStatistics.m
In[19]:=
Density[CauchyDistribution[a,b],t]
Out[19]=
      1
-----
      2
      (-a + t)
Pi b (1 + -----)
      2
      b

In[20]:=
Mean[NormalDistribution[mu,sigma]]
Out[20]=
mu

In[21]:=
Variance[NormalDistribution[mu,sigma]]
Out[21]=
2
sigma

```

The following example illustrates how a set of data is analyzed with *Mathematica*. The command **Table[Random[NormalDistribution[mu,sigma]],{n}]** generates a random list of n numbers which approximates a normal distribution of mean μ and standard deviation σ .

□ Example:

In this example, a list of 175 numbers which approximate the normal distribution of mean 75 and standard deviation 10 is produced and stored in **table1**. The values in **table1** are rounded to integers with **Map[Floor,table1]** where **Floor[x]** gives the greatest integer not larger than x . **Map** applies **Floor** to each value in **table1**. This new table is called **table2**. The integer values in **table2** are then sorted with **Sort[table2]** and named **table3**. A shortened list of these sorted integers is given with **Short[table3,3]**.

Since the command **Random** is used, each time these commands are executed, different results are obtained.

The mean of **table3** is computed by adding the integers in **table3** and dividing by the number of integers found in **table3**. This is accomplished in the single command

Apply[Plus, table3]/Length[table3]//N.

Of course, the mean can also be computed with **Mean[table3]** which is illustrated below.

```

StatsExamples

In[22]:=
  table1=Table[Random[
    NormalDistribution[75,10]],{175}];
  table2=Map[Floor,table1]
  table3=Sort[table2]
  Short[table3,3]

Out[22]:=Short=
  {49, 49, 53, 54, 54, 56, 57, 58, 59, 60,
   60, 60, 61, 61, 61, <<153>>, 88, 89, 90,
   91, 93, 93, 97}

In[23]:=
  Apply[Plus,table3]/
  Length[table3] // N

Out[23]=
  73.6629

In[24]:=
  Mean[table3]

Out[24]=
  12891
  -----
  175

```

table1 is a table of 175 "random" numbers that approximate a normal distribution of mean 75 and variance 100.
Map[Floor, table1] computes **Floor[table1[[i]]]** for each $i=1, 2, \dots, 175$ and names the resulting list **table2**.
Sort[table2] sorts **table2** according to the standard increasing order and names the resulting list **table3**.
Short[table3, 3] displays an abbreviated form of **table3** on no more than three lines.

computes a numerical value of the mean of **table3**.

computes an exact value of the mean of **table3**.

In order to compare the mean of `table3` obtained by the two methods, a numerical approximation of `Mean[table3]` is requested to show that the same result is achieved by each approach. `Median[table3]` determines the median of `table3` and `Quartiles[table3]//N` determines the numerical approximation of the quartiles of `table3`. `MeanDeviation[table3]//N` gives a numerical value of the mean deviation and `N[MedianDeviation[table3]]` gives a numerical value for the median deviation of `table3`.

```

In[25]:=
Mean[table3] // N
Out[25]=
73.6629
In[26]:=
Median[table3]
Out[26]=
74
In[27]:=
Quartiles[table3] // N
Out[27]=
{67., 81.}
In[28]:=
MeanDeviation[table3] // N
Out[28]=
7.64193
In[29]:=
N[MedianDeviation[table3]]
Out[29]=
7.

```

computes a numerical approximation of the mean of `table3`.

computes the median of `table3`

computes numerical values of the quartiles of `table3`.

computes a numerical value of the mean deviation of `table3`.

computes a numerical value of the median deviation of `table3`.

A summary of some of the information concerning **table3** is obtained with the command **DispersionReport[table3]//N**. These numerical values are shown below:

```
In[30]:=
  DispersionReport[table3] // N
Out[30]=
  {Variance -> 87.2063,
   StandardDeviation -> 9.33843,
   SampleRange -> 48.,
   MeanDeviation -> 7.64193,
   MedianDeviation -> 7.,
   QuartileDeviation -> 7.}
```

computes numerical approximations of the variance, standard deviation, sample range, mean deviation, median deviation, and quartile deviation of table3.

□ Example:

In the following example, the function **countlist[table]** is defined. This function counts the number of times each distinct member of the list **table** appears in **table**. First the variables **j** and **list** are defined to be local to the function **countlist**. Second, **table** is sorted with **Sort[table]** and this sorted list is named **list**. Then, **Union[list]** removes all duplicates from **list** to obtain a new list, **list2**. Finally, **Count[list,list2[[j]]]** gives the number of elements in **list** that match each element in the list of distinct elements, **list2[[j]]**. In other words, the number of times each element in **list** appears is given. This number is divided by **Length[list]** (the total number of elements in **list**) to yield the portion of list that each element comprises. This function is illustrated below with **table3** given earlier. The output lists the distinct elements of **table3** along with a numerical value which indicates the portion of **table3** that each element constitutes. Hence, if {**number**, **p**} is an element of **table3**, then the probability of choosing **number** from the list of numbers in **table3** is **p**.

Also defined below is the function `between[list, {a,b}]` which counts the number of elements of `list` which fall on the interval from `a` to `b`, including the endpoints. This is done with the command `Sum[Count[list, j], {j, a, b}]` where `Count[list, j]` gives the number of elements of `list` which match `j` (i.e., the number of times `j` appears in `list`). This is done for each value of `j` from `j = a` to `j = b` and then the sum of these numbers is taken to yield the total number of values between `a` and `b`.

- o In Version 2.0, the command `Block` has been replaced by the command `Module`. However, `Block` is evaluated correctly when using Version 2.0.

```

In[31]:=
  countlist[table_]:=
    Block[{j,list},
      list=Sort[table];
      list2=Union[list];
      Table[{list2[[j]],
        N[Count[list,list2[[j]]]/
          Length[list]]},
        {j,1,Length[list2]}]
    ]

In[32]:=
  between[list_,{a_,b_}]:=
    Sum[Count[list,j],{j,a,b}]

In[33]:=
  table4=countlist[table3]
  Short[table4,4]

Out[33]//Short=
  {{49, 0.0114286}, {53, 0.00571429},
   {54, 0.0114286}, {56, 0.00571429},
  <<35>>, {93, 0.0114286}, {97, 0.00571429}}

```

Be sure that square brackets are nested correctly. Notice that a semi-colon is placed at the end of each command, except for the last.

computes countlist[table3] and names the resulting list table4. Short[table4,4] displays an abbreviated form of table4 on no more than four lines.

The sum of the numerical values given in `table4` should equal 1. This is verified below with `Apply[Plus,table4][[2]]`. The command `Apply[Plus,table4]` adds the corresponding components of the members of `table4` and is, therefore, of the form of each element of `table4`. Hence, the sum of the second components is given with `Apply[Plus,table4][[2]]`.

A table is then compiled which gives an interval breakdown of the values found in **table3**. This is accomplished by choosing several intervals, {45,55}, {55,65},...,{95,105}, and using the function **between**. The output is given in the form of a table as shown below.

```

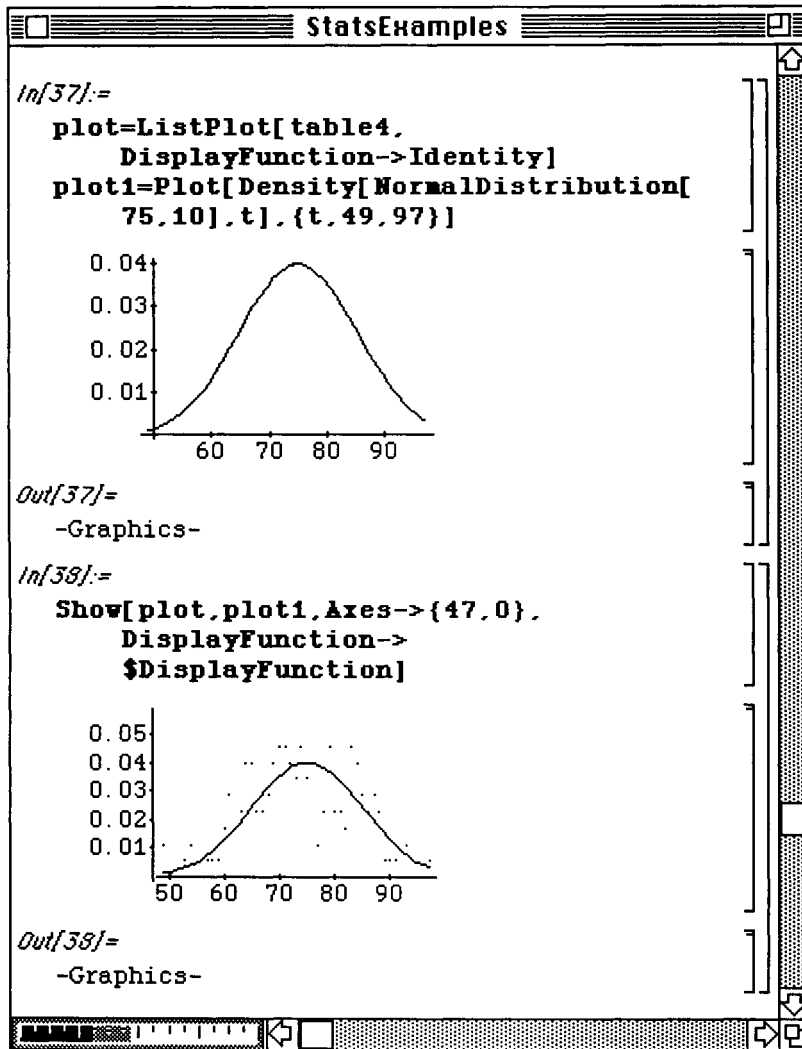
In[34]:=
  Apply[Plus, table4][[2]]
Out[34]=
  1.
In[35]:=
  Table[{45+10 i, 55+10i,
    between[table3, {45+10 i, 55+10i}]},
    {i, 0, 5}]/TableForm
Out[35]//TableForm=
  45  55  5
  55  65  33
  65  75  70
  75  85  61
  85  95  23
  95  105  1

```

Hence, there are five elements of table3 between 45 and 55; 33 elements between 55 and 65; 70 elements between 65 and 75; 61 elements between 75 and 85; 23 elements between 85 and 95; and 1 element between 95 and 105.

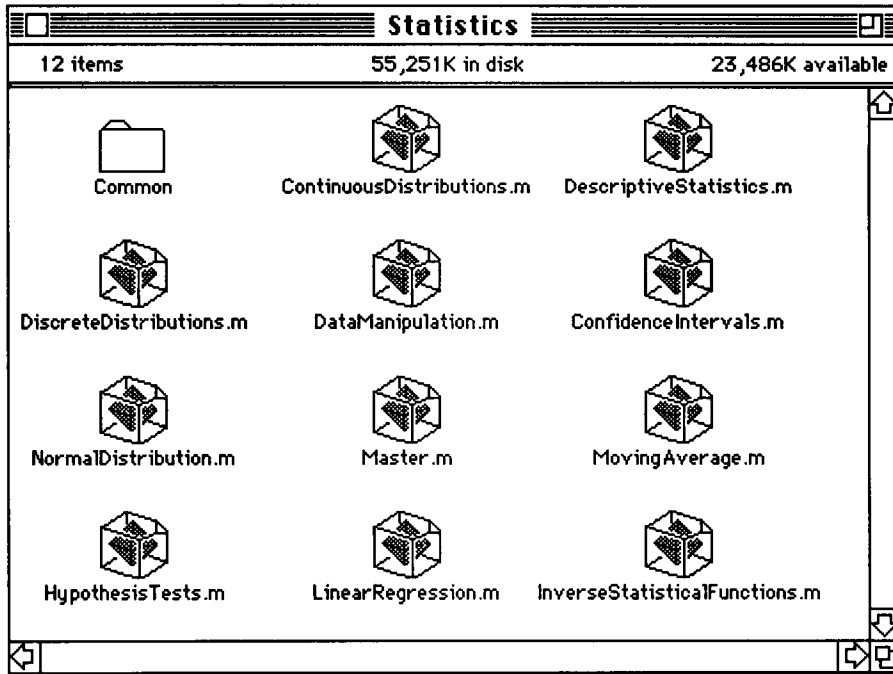
The elements of `table4` can be graphed along with the normal distribution with mean 75 and variance 10 to show how well these values approximate the density function of this distribution. Since each member of `table4` is an ordered pair, the elements in `table4` are plotted with `ListPlot` below. However, the graph is not shown initially, because the option `DisplayFunction->Identity` is used. The density function of the normal distribution is plotted for values of `t` between 49 and 97. This plot is named `plot1`.

The second graph below shows `plot` and `plot1` simultaneously. The option `DisplayFunction->${DisplayFunction}` allows `plot` to be displayed. The `Show` command also specifies that the axes meet at the point `{47,0}`.



● Version 2.0 Statistics

In Version 2.0, the **Data Analysis** folder is replaced by the following **Statistics** folder. Some of the added packages are discussed below.



*In Version 2.0, the folder **DataAnalysis** has been replaced by the folder **Statistics**. Notice that a considerable number of packages have been added.*

● 9.7 HypothesisTests.m

The Version 2.0 **HypothesisTests.m** package contains useful hypothesis test commands for solving problems in statistics. The first command shown below is **MeanTest[list, mu0, options]** which is used to test the null hypothesis that the population mean equals **mu0**. Each command in this package yields the appropriate P-value. Recall that the P-value of a hypothesis test is equal to the smallest significance level at which the null hypothesis can be rejected. Hence, if the P-value is less than or equal to the specified significance level, then the null hypothesis is rejected. Otherwise, the null hypothesis is not rejected.

○ Example:

Consider the data collected by the U.S. Energy Information Administration on residential energy expenditures. According to this agency, the mean residential energy expenditure of all American families was \$1123 in 1985. The expenditures of 15 upper-level families is given in **energy**. The hypothesis that the population mean equals \$1123 is tested with this data (with **mu0 = 1123**). The command **MeanTest[energy, 1123]** gives the one-sided P-value. Assuming a significance level of .05, the P-value obtained is compared to $.5(.05) = .025$. Since $.0014959 < .025$, the null hypothesis is rejected.

A helpful option of all of the tests to be discussed in this section is that of **FullReport->True**. This option is illustrated with the same problem to reveal the sample mean, the test statistic, the number of degrees of freedom (when applicable), and type of test statistic used. These commands use the test statistics based on the normal distribution, the Student's t-distribution, the chi-square distribution, or the F-distribution. The **FullReport** shows that in this case, the Student's t-distribution was employed.

```

In[13]:=
  <<HypothesisTests.m

In[14]:=
  energy={1254, 1615, 1711, 1350, 1521, 1293,
          1227, 908, 1205, 1154, 1231, 1351,
          1790, 1369, 1185};

In[15]:=
  MeanTest[energy, 1123]

Out[15]=
  OneSidedPValue -> 0.0014959

In[16]:=
  MeanTest[energy, 1123,
           FullReport->True]

Out[16]=
  {FullReport ->      Mean      TestStat  DF,
   _____
                15      3.58403   14

  StudentTDistribution,

  OneSidedPValue -> 0.0014959}

```

The semi-colon is placed at the end of the definition of **energy** in order to suppress the resulting output.

More information can be obtained with the command **MeanTest** by using the option **FullReport->True**.

Another option is `KnownVariance->var0`. When this option is used, the normal test statistic is used to obtain the P-value as expected. This is also shown below.

```

In[17]:=
  MeanTest[energy, 1123,
    FullReport->True, KnownVariance->53361]

Out[17]=
{FullReport ->      Mean      TestStat,
  20164
  -----
  15          3.70979

  NormalDistribution,

  OneSidedPValue -> 0.000103715}

```

o Example:

Consider another set of data which gives the daily intake of calcium (in milligrams) for 35 people with an income below the poverty level. This is given in `calcium` below.

```

calcium={879,1096,701,986,828,1077,703,
  555,422,997,473,702,508,530,
  513,720,944,673,574,707,864,
  1199,743,1325,655,1043,599,1008,
  705,180,287,542,893,1052,473};

```

A sample list indicating the daily intake of calcium for 35 people with income below the poverty level.

The recommended daily allowance (RDA) is 800 milligrams. A nutritionist states that the average person with an income below the poverty level gets less than the RDA of 800 mg. This claim is tested below with **MeanTest**. This is a one-sided test. Suppose that the significance level is again .05. Since the P-value which results (.124765) is greater than .05, the null hypothesis (that the calcium intake of people of poverty level incomes is equal to 800 mg) is not rejected. Another option available to **MeanTest** is **KnownStandardDeviation->sigma**. Suppose that the standard deviation is assumed to be 262. When the sample standard deviation equals the known standard deviation, *Mathematica* uses the normal distribution with **MeanTest**. This is indicated in the **FullReport**.

```

In[19]:=
MeanTest[calcium, 800,
          FullReport->True]
Out[19]=
{FullReport ->      Mean      TestStat    DF,
  26156
  -----
           35      -1.17153    34

  StudentTDistribution, OneSidedPValue -> 0.124765}

In[20]:=
MeanTest[calcium, 800,
          FullReport->True,
          KnownStandardDeviation -> 262]
Out[20]=
{FullReport ->      Mean      TestStat,
  26156
  -----
           35      -1.18967

  NormalDistribution, OneSidedPValue -> 0.117089}

```

Mathematica can also be used to conduct hypothesis tests for the means of two normal populations with equal standard deviations using independent samples. This is accomplished with the command **MeanDifferenceTest[list1, list2, diff0, options]** where the two populations are given in **list1** and **list2**. The value **diff0** is compared to **Mean[list1]-Mean[list2]**.

o Example:

The data given in **below** represents the daily protein intake in grams for 10 people with incomes below the poverty level while **above** represents that of 15 people with incomes above the poverty level. The null hypothesis is that the below-poverty mean is not less than the above-poverty mean. Hence, the alternative hypothesis is that the below-poverty mean is less than the above-poverty mean. If the means of these two populations are equal, the difference of the respective means equals zero. Hence, $\text{diff} = 0$ in the command below. The P-value given is .00934066 which is less than a significance level of .05. Thus, the null hypothesis is rejected. In the `FullReport` which follows, the test statistic used in the determination of this P-value is given.

```

In[22]:=
  below={51.4,76.7,73.7,66.2,65.5,
         49.7,65.8,62.1,75.8,62.0,
         72.0,55.0,79.7,65.4,73.3};
  above={86.0,59.7,68.6,98.6,87.7,
         69.0,80.2,78.1,69.8,77.2};

In[23]:=
  MeanDifferenceTest[below,above,0]

Out[23]=
  OneSidedPValue -> 0.00934066

In[24]:=
  MeanDifferenceTest[below,above,0,
                    FullReport->True]

Out[24]=
  {FullReport ->      MeanDiff   TestStat   DF
                    -11.2033   -2.60755   16.5403

  StudentTDistribution,

  OneSidedPValue -> 0.00934066}

```

o **Example:**

Consider the data below which is used to compare the lifetimes of two brands of water heaters. The lifetimes (in years) of one brand is given in the list **heater1** while that of the other brand is given in **heater2**. In this problem, the null hypothesis is that the mean of the lifetimes of **heater1** equals that of **heater2**. Hence, this is a two-sided test. Assume a significance level of .05. The P-value for this test is determined from the Student's t-distribution. Since this value is greater than .025, the null hypothesis is not rejected. Hence, the data do not provide sufficient information to conclude that the two brands of water heaters have different mean lifetimes.

```

In[26]:=
  heater1={6.9,7.2,7.6,7.3,6.6,5.7,
           7.8,6.2,5.5,7.4,8.2,6.9};
  heater2={8.7,8.6,11.2,7.0,6.1,6.1,
           8.7,7.5,6.3,6.7,7.7,7.0,
           7.8,7.5,10.7};

In[27]:=
  MeanDifferenceTest[heater1,heater2,0,
                    FullReport->True]

Out[27]=
{FullReport ->      MeanDiff      TestStat      DF
                  -0.898333      -1.94567      22.2285

  StudentTDistribution,

  OneSidedPValue -> 0.0322241}

```

In addition to hypothesis tests on the mean of a population as was illustrated with **MeanTest**, *Mathematica* is able to consider hypothesis tests for a population standard deviation (or variance). This is done with **VarianceTest**[*list*, *var0*, *options*]. In problems of this type, the null hypothesis is that the variation of the population in *list* equals *var0*. This command uses the same options as the previously used commands.

○ Example:

An analysis is conducted on the data in `diameter` which gives the diameter in millimeters of 20 bolts produced by a hardware manufacturer. It has been determined that an acceptable standard deviation for bolt diameters is .09 millimeters. Therefore, `var0 = (.09)^2` in the `VarianceTest` command which follows. The `FullReport` indicates that the chi-square distribution is used as is expected. If a significance level of .05 is assumed, the null hypothesis is rejected. Hence, the bolts produced by the manufacturer have diameter less than .09 millimeters.

```

In[17]:=
  diameter={10.03,10.08,10.05,10.03,
            9.89,9.95,9.97,9.99,
            9.99,10.00,10.03,10.08,
            9.96,9.94,9.98,10.02,
            10.10,10.01,10.05,9.98};

In[18]:=
  VarianceTest[diameter, (.09)^2,
               FullReport->True]

Out[18]=
{FullReport ->      Variance      TestStat    DF,
 0.00272921        6.73879      19

  ChiSquare Distribution, OneSidedPValue -> 0.00451858}

```

● 9.8 ConfidenceIntervals.m

- o In Version 2.0, **ConfidenceIntervals.m** is contained in **Statistics**; in Version 1.2 **ConfidenceIntervals.m** is contained in **Data Analysis**.

An important concept in statistics is that of confidence intervals. For the two-sided hypothesis test,

$$H_0 : \mu = \mu_0$$

$$H_a : \mu \neq \mu_0$$

at a significance level α , the null hypothesis is not rejected if μ_0 lies in the $(1 - \alpha)$ -level confidence interval for μ and, the null hypothesis is rejected if μ_0 does not lie in the $(1 - \alpha)$ -level confidence interval for μ .

These confidence intervals can be determined for the normal, Student's t, chi-square, and F distributions for several types of hypothesis tests using commands found in **ConfidenceIntervals.m**.

□ Example:

In the first example below, the data used in the **HypothesisTests.m** section concerning the residential energy expenditures of 15 upper level families located in **energy** is considered. In the previous section, the null hypothesis that the mean expenditure equals \$1123 was tested. Since this is a test concerning the mean, the command **MeanCI[list, options]** is used. Therefore, the confidence interval for this data is determined with **MeanCI[energy]//N**. (**//N** requests that numerical values be given as opposed to exact.) Since 1123 does not lie in this interval, the null hypothesis is rejected as it was in the previous section. One of the options available is that of **ConfidenceLevel->alpha** with default value **.95**. The **(.10)**-level confidence interval is determined with the **ConfidenceLevel->.90** option. Of course, a smaller interval is the result.

```

In[1]:=
  <<ConfidenceIntervals.m

In[2]:=
  energy={1254,1615,1711,1350,1521,1293,
          1227,908,1205,1154,1231,1351,
          1790,1369,1185};

In[3]:=
  MeanCI[energy]//N
Out[3]=
  {1211.85, 1476.68}

In[4]:=
  MeanCI[energy,ConfidenceLevel -> .90]//N
Out[4]=
  {1235.53, 1453.}

```

In this case, it is 95% certain that the population mean lies between 1211.85 and 1476.68.

In this case, it is 90% certain that the population mean lies between 1235.53 and 1453.

□ Example:

In the case of the daily calcium intake in which the standard deviation was known, the null hypothesis that the daily intake of people with incomes below the poverty level equals 800 milligrams was not rejected. This test is investigated below with the data given in `calcium`. This illustrates another option, `KnownStandardDeviation->sd`. For this problem, `sd = 262`. The confidence interval found with `MeanCI` shows that 800 is in the interval. Thus, the null hypothesis is not rejected.

```
In[6]:=
  calcium={879,1096,701,986,828,1077,703,
           555,422,997,473,702,508,530,
           513,720,944,673,574,707,864,
           1199,743,1325,655,1043,599,1008,
           705,180,287,542,893,1052,473};
  MeanCI[calcium, KnownStandardDeviation -> 262]//N
Out[6]=
  {660.515, 834.113}
```

In this case, the confidence interval is (660.515,834.113)

Hypothesis tests for the comparison of two population means can also be considered with confidence intervals. Again, referring to data from the previous section, the command `MeanDifferenceCI[list1, list2, options]` is explored.

□ Example:

Recall that `below` represents the daily protein intake in grams for 10 people with incomes below the poverty level while `above` represents that of 15 people with incomes above the poverty level. The null hypothesis is that the below-poverty mean is not less than the above-poverty mean. The confidence interval obtained below does not contain zero (the hypothesized mean difference). Hence, the null hypothesis is rejected. Therefore, the data indicate that the average person with an income below the poverty level gets less protein than the average person with an income above the poverty level.

```
In[9]:=
  below={51.4,76.7,73.7,66.2,65.5,
         49.7,65.8,62.1,75.8,62.0,
         72.0,55.0,79.7,65.4,73.3};
  above={86.0,59.7,68.6,98.6,87.7,
        69.0,80.2,78.1,69.8,77.2};
  MeanDifferenceCI[below, above]
General::spell1:
  Possible spelling error: new symbol name
  "below" is similar to existing symbol "Below"
General::spell1:
  Possible spelling error: new symbol name
  "above" is similar to existing symbol "Above"
Out[9]=
  {-20.2874, -2.11927}
```

Version 2.0 warns of possible spelling errors. In this case, the warning message can be ignored since no error was made.

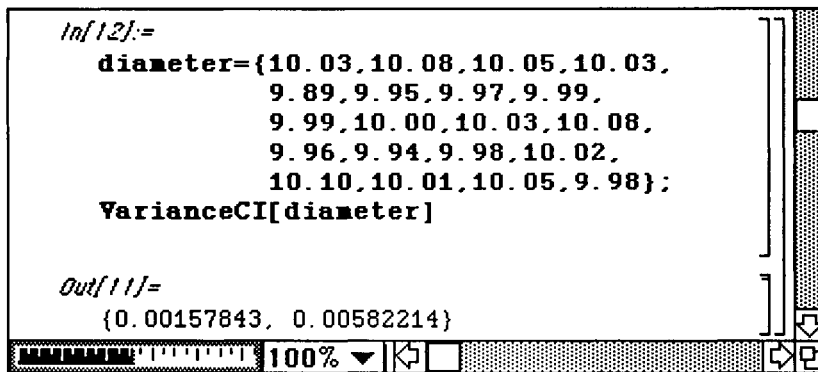
Mathematica can also determine the confidence interval of the variance of a population. (Note that the variance is used instead of the standard deviation.)

□ **Example:**

Recall the information collected by a hardware manufacturer giving the diameter of 20 bolts produced by the company. It was determined that an acceptable standard deviation for bolt diameters is .09 millimeters. Hence, the null hypothesis is that the variance of the diameters equals $(.09)^2 = .0081$. The command which determines the confidence interval for variance is `VarianceCI[list, options]`. When this command is used with `diameter`, the interval obtained does not contain .0081. Thus, the null hypothesis is rejected. However, the manufacturer can be 95% confident that the variance of the diameters of all 10-millimeters bolts produced is somewhere between .00157843 and .00582214.

```
In[12]:=
diameter={10.03,10.08,10.05,10.03,
          9.89,9.95,9.97,9.99,
          9.99,10.00,10.03,10.08,
          9.96,9.94,9.98,10.02,
          10.10,10.01,10.05,9.98};
VarianceCI[diameter]

Out[11]=
{0.00157843, 0.00582214}
```

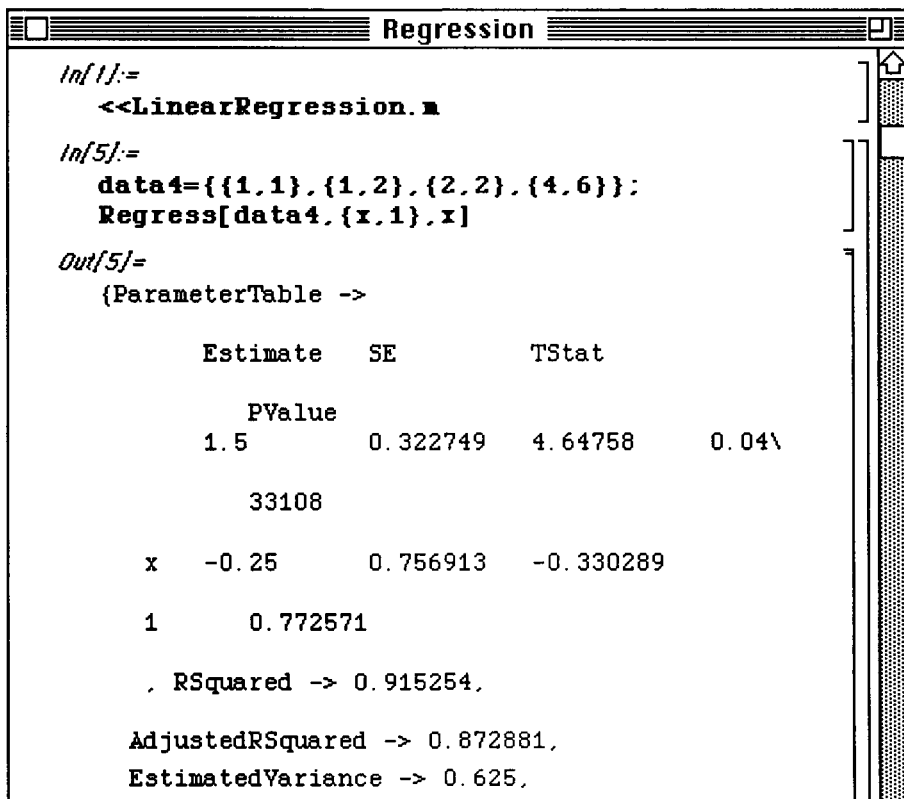


● 9.9 LinearRegression.m

The Version 2.0 `LinearRegression.m` package contains the command `Regress[data, functions, vars]` which leads to the determination of the linear regression equation for the data points in `data` as well as other useful information concerning `data`. In the `Regress` command, the regression equation is formed from the linear combination of functions given in the list `functions`. Also, `vars` represents the variables in the regression equation. Since *Mathematica* can be used for multiple regression, the number of functions and variables will depend on the type of problem to be solved.

○ Example:

Consider the following list of data in `data4`. The regression equation for `data4` is found with `Regress`. The coefficients of the regression equation appear in the column labeled `Estimate` under `ParameterTable` with 1.5 as the coefficient of `x` and -0.25 the coefficient of 1. Hence, the regression equation is $y = 1.5x - 0.25$. The other information given is discussed in later examples.



```

In[1]:=
  <<LinearRegression.m

In[5]:=
  data4={{1,1},{1,2},{2,2},{4,6}};
  Regress[data4,{x,1},x]

Out[5]=
  {ParameterTable ->
    Estimate SE TStat
    PValue
    1.5 0.322749 4.64758 0.04\
    33108
    x -0.25 0.756913 -0.330289
    1 0.772571
    , RSquared -> 0.915254,
    AdjustedRSquared -> 0.872881,
    EstimatedVariance -> 0.625,
  }

```

```

ANOVA Table ->

```

	DoF	SoS	MeanSS	FRatio
		PValue		
	1	13.5	13.5	21.6
		0.0433108		
	2	1.25	0.625	
Model				
Error	3	14.75		
Total				

o Example:

Next, consider the age versus price data collected on a particular type of sports car. This data is as follows and is defined below in **price**.

<u>Age(years)</u>	<u>Price(\$100s)</u>
5	85
4	103
6	70
5	82
5	89
5	98
6	66
6	95
2	169
7	70
7	48

The regression equation is found to be $y = 195.468 - 20.2613x$. Notice in the output that there is not enough room to include all of the information in ANOVA Table on one line. Therefore, the remaining information concerning Model, Error, and Total is given beneath these headings. Most of the information provided is self-explanatory. Since the value of RSquared (the coefficient of determination) is near one (.853373), a good deal of the variation in the sampled prices is explained by the regression line. This implies that age is useful in predicting price.

Another way to analyze the provided information is through hypothesis tests for the slope of the regression line. The variable x is a useful predictor of y if they are linearly related. This can be tested by forming the hypothesis test :

$$H_0 : \beta_1 = 0 \quad (x \text{ is not useful for predicting } y)$$

$$H_a : \beta_1 \neq 0 \quad (x \text{ is useful for predicting } y)$$

where the regression line is of the form $y = \beta_0 + \beta_1x$.

The final entry in the row corresponding to x is the P-value for the hypothesis test. If a significance level of .05 is used, the null hypothesis is rejected since $.00004 < .025$.

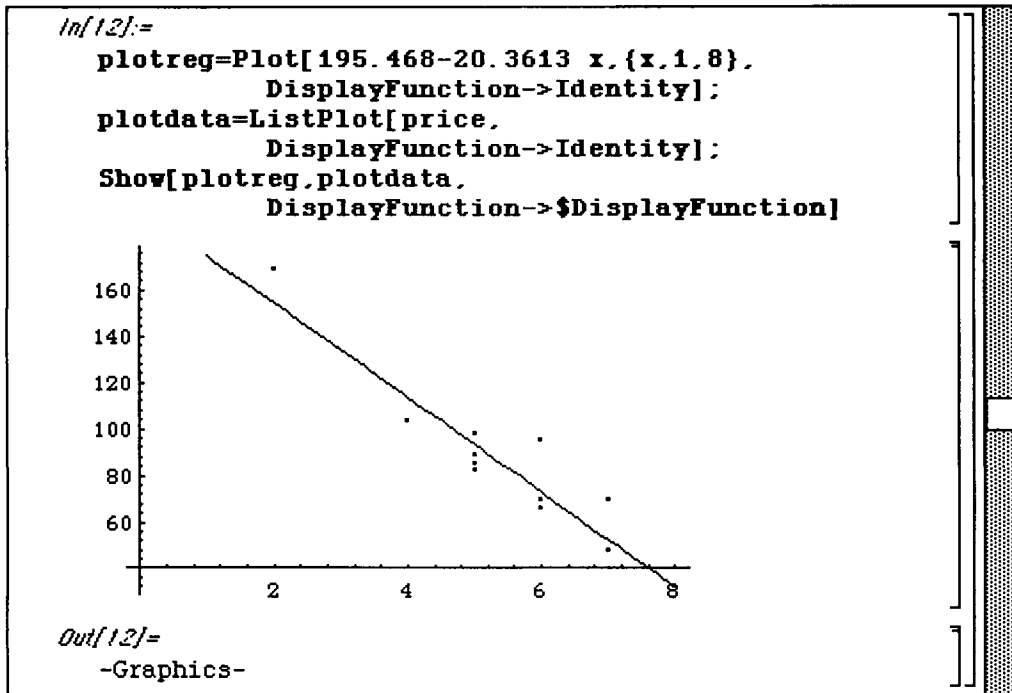
```

In[9]:=
price={{5.85},{4.103},{6.70},{5.82},
        {5.89},{5.98},{6.66},{6.95},
        {2.169},{7.70},{7.48}};
Regress[price,{1,x},x]

Out[9]=
{ParameterTable ->
  Estimate SE TStat PValue
  1 195.468 15.2403 12.8257 0
  x -20.2613 2.79951 -7.23743 0.0000488191
  RSquared -> 0.853373, AdjustedRSquared -> 0.837081,
  EstimatedVariance -> 158.17,
  ANOVATable ->
    DoF SoS MeanSS FRatio
    PValue
  1 8285.01 8285.01 52.3804 0.0000\
    488191
  9 1423.53 158.17
  Model
  Error 10 9708.55
  Total
  }

```

The regression equation is plotted below with the `ListPlot` of the data in `price`. As expected, as the age of the car increases, the price decreases.



As indicated before, `Regress` can be used for multiple regression. This is demonstrated below with the addition of the number of miles driven (in thousands) to the data in the previous example.

o **Example:**

It was determined above that 85.3% (`RSquared*100`) of the variation in the price data is explained by age. Then, by using this additional information (mileage), perhaps this variation is better explained. The revised data is given in `extra` below. The variables in the new regression equation are `x1` and `x2` which represent age and miles, respectively. Using the `Regress` command below, the equation is found to be $y = 183.035 - 9.50427 x_1 - .821483 x_2$. Again, the coefficients are located under the column labeled `Estimate`. To determine if the variation is better explained with the additional data, the value of `RSquared` is considered. In this case, `RSquared` = .936115. Hence, 93.6% of the price variation is explained by age and miles driven. Since age only explained 85.3% of the variation, the multiple regression equation provides a much better explanation of the variation in the price data than the simple linear equation found in the previous example.

As was the case above, hypothesis tests can be used with multiple regression.

Assuming a regression equation of the form $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$, the following hypothesis test is formed :

$$H_0 : \beta_1 = \beta_2 = 0$$

H_a : At least one of β_1 and β_2 is not zero.

This test depends on the the F-statistic. The P-value for this test appears under PValue in the ANOVATable in the output below. Since this P-value (.0000166571) is much smaller than a reasonable significance level, the null hypothesis is rejected. Hence, the previous findings that age and mileage are good predictors of price is verified.

```

In[14]:=
  extra={{5,57,85},{4,40,103},{6,77,70},{5,60,82},
         {5,49,89},{5,47,98},{6,58,66},{6,39,95},
         {2,8,169},{7,69,70},{7,89,48}};
  Regress[extra,{1,x1,x2},{x1,x2}]

Out[14]:=
{ParameterTable ->

      Estimate   SE      TStat    PValue
1      183.035   11.3476  16.1298    0
x1     -9.50427   3.87419  -2.45323  0.0397362
x2     -0.821483  0.255207 -3.21889  0.0122595

RSquared -> 0.936115, AdjustedRSquared -> 0.920144,
EstimatedVariance -> 77.529,
ANOVATable ->

Model
Error
Total

      DoF   SoS      MeanSS   FRatio   PValue
2       2   9088.31   4544.16   58.6124  0.0000166571
8       8   620.232   77.529
10      10  9708.55
  
```

Chapter 10

Getting Help from *Mathematica* and Making *Mathematica* Do What You Want

■ 10.1 Getting Help from *Mathematica*

■ Help Commands

Becoming competent with *Mathematica* can take a serious investment of time. Hopefully, messages that result from syntax errors are viewed lightheartedly. Ideally, instead of becoming frustrated, beginning *Mathematica* users will find it challenging and fun to locate the source of errors. In this process, it is natural that one will become more proficient with *Mathematica*.

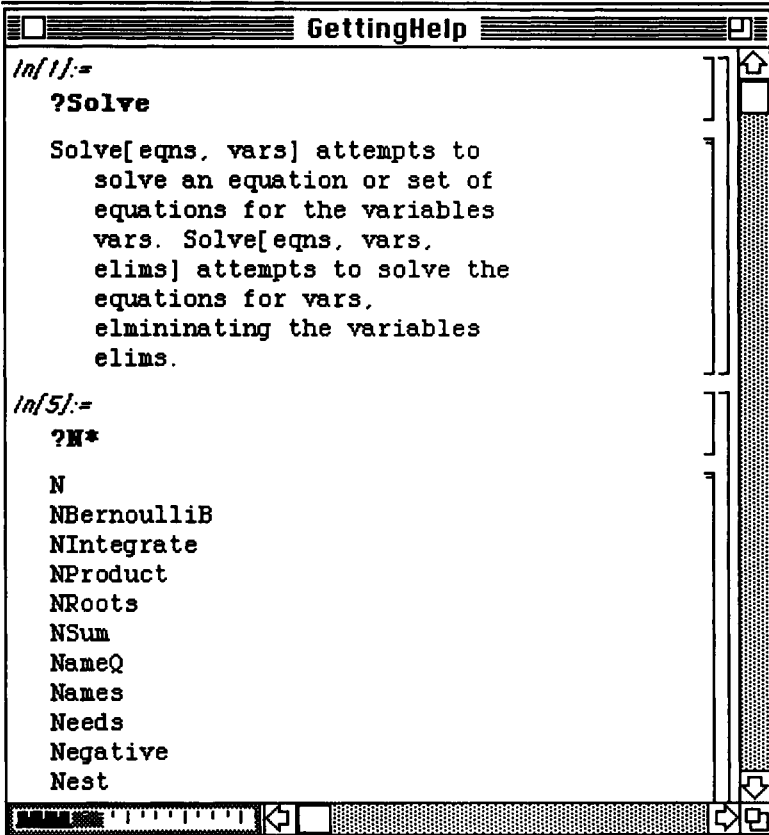
One way to obtain information about commands and functions is the command `?`. `?Name` gives information on the *Mathematica* function **Name**

□ Example:

The following window shows how information is obtained on the command `Solve` as well as the form in which this information is given. Notice how the description includes the particular forms in which the command `Solve` should be entered. This can be quite helpful in attempting to use the command.

The command `?` can be used in several different ways. For example, `?letter*` gives a list of all *Mathematica* commands which begin with `letter`. This is illustrated below with the letter **N**.

File Edit Cell Graph Find Action Style Window



?Solve
yields a brief description of
the **Solve** command.

?N*
gives a list of all commands
that begin with
N.

Another useful application of `?` is in determining the definition of functions. This is especially helpful in verifying that a user-defined function has been defined correctly. The following example shows that after the function `f` is defined, the command `f` gives the definition of `f`. Notice that after `f` is cleared, the formula for `f` is no longer known. Hence, `?f` yields nothing. If the symbol `:=` is used in the definition of a function, the function's formula is not automatically given as output as it is below when the equals sign is used in the definition. Therefore, `?` is of particular help in obtaining the definition.

```

UserFunctions
In[2]:=
  f[x_]=Sin[3x]Cos[4x]
Out[2]=
  Cos[4 x] Sin[3 x]
In[3]:=
  ?f
f
f/: f[x_] = Cos[4 x] Sin[3 x]
In[4]:=
  Clear[f]
In[5]:=
  ?f
f

```

Begin by defining `f[x]=Sin[3x]Cos[4x]`.

Then `?f` yields the definition of `f`.

However, after clearing the definition of `f` with the command `Clear[f]` `?f` yields nothing.

□ Example:

Several other forms of the ? command are shown below :

?*letters gives all *Mathematica* commands which end in **letters**.

?letters* gives all *Mathematica* commands which begin with **letters**.

?function gives a description of the built-in *Mathematica* function, **function**.

Examples which illustrate these commands follow :

⌘ File Edit Cell Graph Find Action Style Window

The screenshot shows a Mathematica notebook window titled "GettingHelp". The notebook has a menu bar with "File", "Edit", "Cell", "Graph", "Find", "Action", "Style", and "Window". The notebook content is as follows:

```

In[6]:=
  ?*olve
DSolve      MainSolve
LinearSolve Solve

In[7]:=
  ?Bessel*
BesselI BesselJ BesselK Bessely

In[9]:=
  ?Bessely
Bessely[n, z] gives the Bessel
function of the second kind.

In[10]:=
  ?Plot*
Plot        PlotJoined
Plot3D      PlotLabel
Plot3Matrix PlotPoints
PlotColor   PlotRange
PlotDivision PlotStyle
  
```

On the right side of the notebook, the output for each command is shown:

- ?*olve** lists all commands with last 4 letters **olve**.
- ?Bessel*** lists all commands with first 6 letters **Bessel**.
- ?Bessely** yields information about the command **Bessely**.
- ?Plot*** lists all commands with first 4 letters **Plot**.

Yet another form of the command ? is **?*letters***. This command gives a list of all *Mathematica* commands that contain **letters**. Several examples which illustrate this are given below.

⌘ File Edit Cell Graph Find Action Style Window

The screenshot shows a Mathematica notebook window titled "GettingHelp". The notebook contains three input cells and their corresponding output cells. The first input cell is `In[10]:=` followed by `?*olve*`. The output cell shows a list of commands: `DSolve`, `Solve`, `LinearSolve`, `SolveAlways`, and `MainSolve`. The second input cell is `In[11]:=` followed by `?*grate*`. The output cell shows `Integrate` and `NIntegrate`. The third input cell is `In[12]:=` followed by `?*lot*`. The output cell shows a list of plotting commands: `ContourPlot`, `Cyclotomic`, `DensityPlot`, `ListContourPlot`, `ListDensityPlot`, `ListPlot`, `ListPlot3D`, `ParametricPlot`, `Plot`, `Plot3D`, `Plot3Matrix`, `PlotColor`, `PlotDivision`, `PlotJoined`, and `PlotLabel`. To the right of the notebook window, there are three text annotations explaining the queries: `?*olve*` gives a list of all the commands containing the letters *olve*; `?*grate*` gives a list of all the commands containing the letters *grate*; and `?*lot*` gives a list of all the commands containing the letters *lot*.

- Version 2.0 users will notice that some commands from earlier versions of *Mathematica* have been made obsolete with the release of Version 2.0. In these cases, *Mathematica* is able to tell what command replaces the outdated command.

o Example:

For example, in Version 2.0 the functions **TrigExpand** and **TrigCanonical** from previous versions of *Mathematica* are obsolete. Nevertheless, the Version 2.0 command **Expand[expression, Trig->True]** performs the same function as the command **TrigExpand[expression]** from earlier versions.

The screenshot shows a Mathematica help window titled "Version2.0Help". The window contains the following text:

```

In[1]:=
<<Trigonometry.m

In[6]:=
?Trig*

Trig          TrigFactor
TrigCanonical TrigReduce
TrigExpand    TrigToComplex

In[7]:=
?TrigExpand

TrigExpand[expr] is obsolete. Its
  functionality is provided by
  Expand[expr, Trig->True].

In[9]:=
?TrigCanonical

TrigCanonical[expr] is obsolete.
  Its functionality is now
  built-in.
  
```

Annotations on the right side of the window explain the output:

- "After loading the package **Trigonometry.m** the command **?Trig*** causes *Mathematica* to list all commands that begin with the four letters **Trig**."
- "Version 2.0 tells which command replaces **TrigExpand**"
- "and **TrigCanonical**."

The window also features a scroll bar, a home button, and a status bar at the bottom showing "100%" zoom and navigation icons.

Another way to obtain information on *Mathematica* commands is the command **Options**.

Options[Command] gives all of the available options associated with **Command**. This is quite useful when working with a *Mathematica* command such as **Plot** which has many options. Notice that the default value (the value automatically assumed by *Mathematica*) for each option is given in the output.

File Edit Cell Graph Find Action Style Window

```

In[11]:=
  Options[Plot]
Out[11]:=
{PlotRange -> Automatic,
  PlotPoints -> 25,
  DisplayFunction ->
    $DisplayFunction,
  PlotStyle -> Automatic,
  AspectRatio ->  $\frac{1}{\text{GoldenRatio}}$ ,
  PlotColor -> Automatic,
  Axes -> Automatic,
  PlotLabel -> None,
  AxesLabel -> None

```

Options[Plot]
lists all options and their
default values associated
with the **Plot**
command.

The command `??Command` gives a brief description of `Command` as well as the list of `Options` available to `Command`.

□ Example:

This is illustrated below with the *Mathematica* command `ListPlot`.

File Edit Cell Graph Find Action Style Window

```

In[13]:=
??ListPlot

ListPlot[{y1, y2, ...}] plots
a list of values. The x
coordinates for each point
are taken to be 1, 2, ....
ListPlot[{{x1, y1}, {x2,
y2}, ...}] plots a list of
values with specified x and
y coordinates.
Attributes[ListPlot] =

{Protected}
ListPlot/:

Options[ListPlot] =

{PlotJoined -> False,
PlotRange -> Automatic,
PlotStyle -> Automatic,
DisplayFunction :>
$DisplayFunction,

```

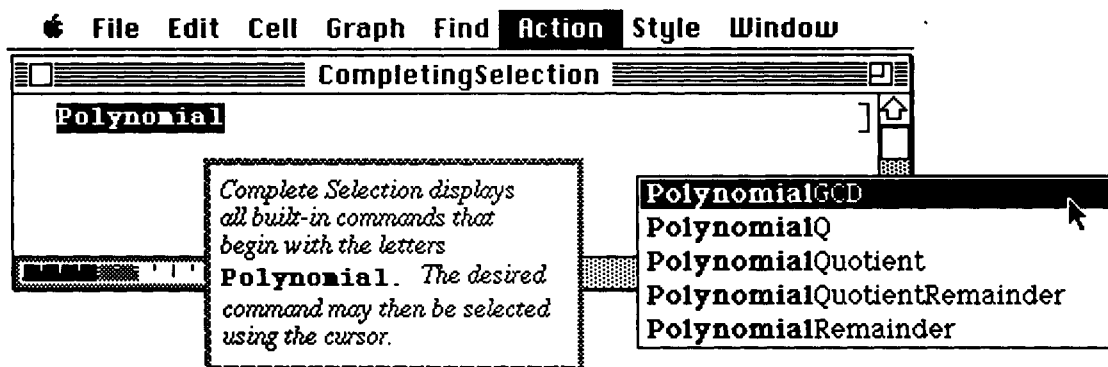
The command `?ListPlot` would yield this information about the command `ListPlot`.

The command `??ListPlot` yields the information given by both the command `?ListPlot` and the command `Options[ListPlot]`.

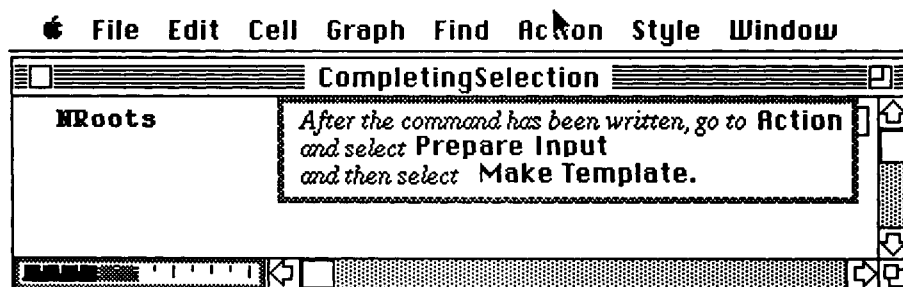
Yet another method for acquiring information on *Mathematica* commands is through the use of **Complete Selection**. This is located under **Prepare Input** in the **Action** submenu and is useful when attempting to complete a command.

□ **Example:**

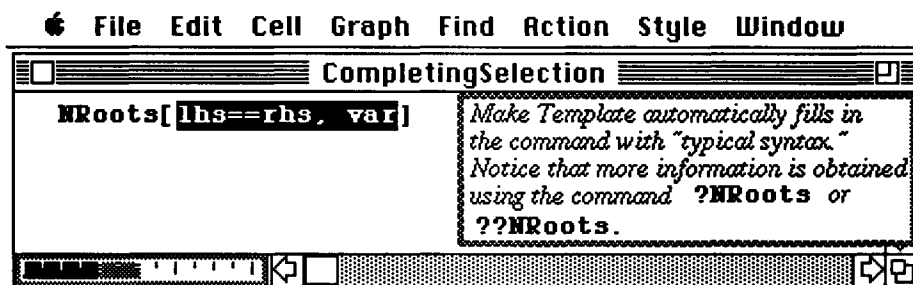
For example, if the user wishes to use a command which begins with **Polynomial**, but does not remember the rest of the command, help can be obtained in the following manner : (1) Type the word **Polynomial**, (2) Move the cursor to the **Action** heading and use to the mouse to obtain the **Action** submenu, (3) Choose **Complete Selection** from the submenu (This causes a list of commands which begin with **Polynomial** to be displayed), (4) Move the cursor to the desired command in the list and click. The correct command is then completed on the screen.



Similar steps may be taken to obtain the proper syntax of a command. The following windows illustrate how **Prepare Input** can be chosen from the **Action** submenu to yield the arguments of **NRoots**.

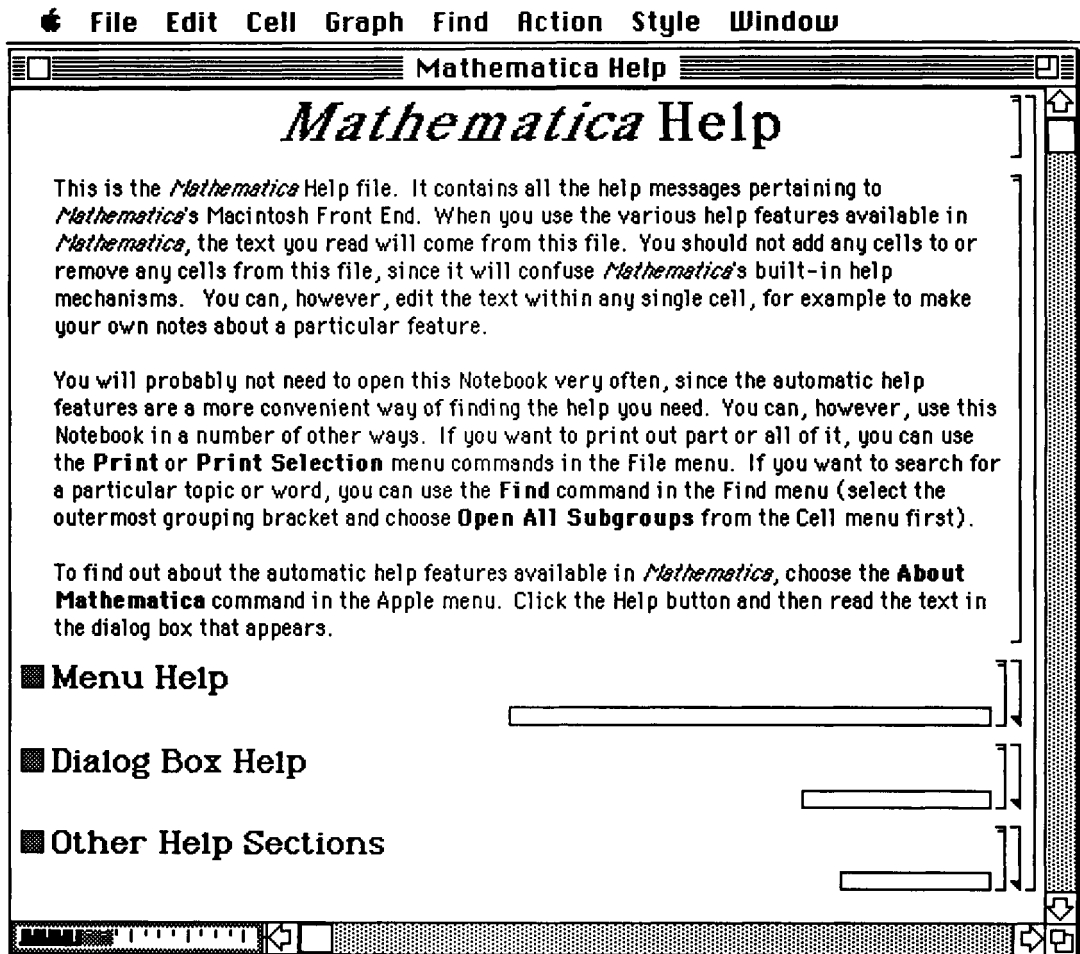


The same steps which were described above for using **Complete Selection** are followed in this case with the exception that **Make Template** is chosen instead of **Complete Selection**. Note that if other *Mathematica* commands begin with the word **NRroots**, then they would all be displayed as they were in the previous example with **Polynomial**.



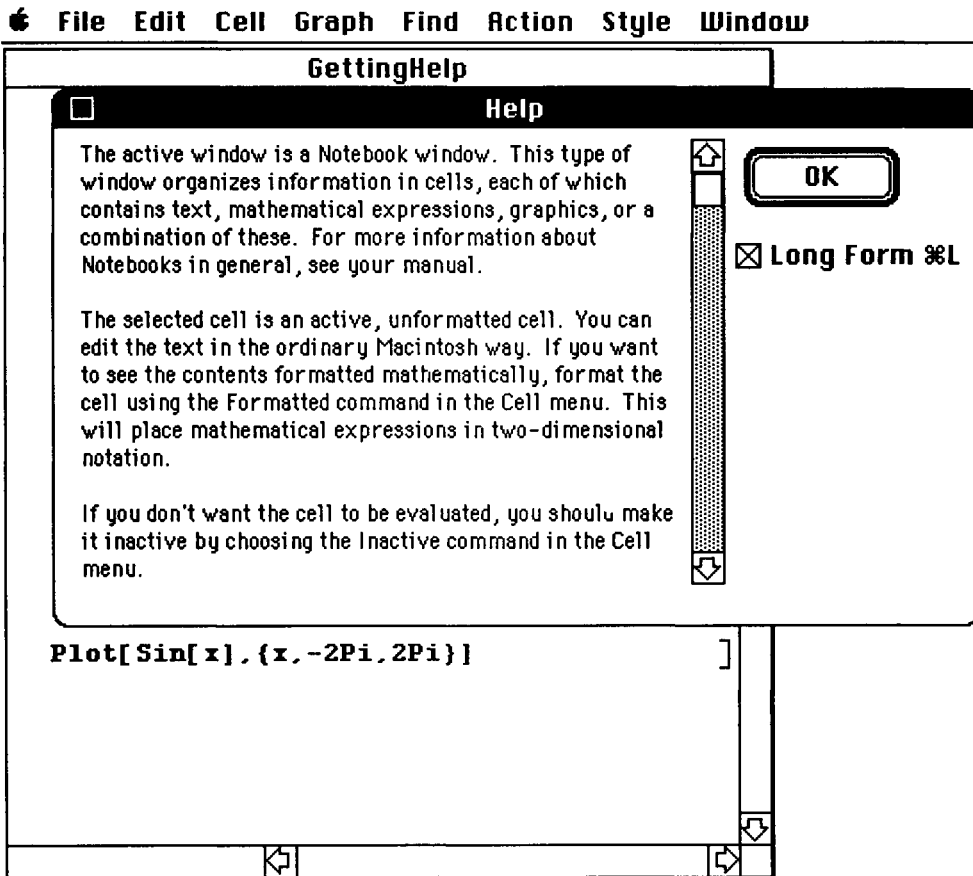
■ *Mathematica* Help

The *Mathematica Help* file is located in the *Mathematica* folder. The information in the window below is given when this file is opened. This describes what is contained in the file and points out that it should be rarely opened. No changes should be made to *Mathematica Help*, since this would only cause confusion. Information can be printed from this file, however, by using **Print...** or **Print Selection...** from **File**.



□ **Example:**

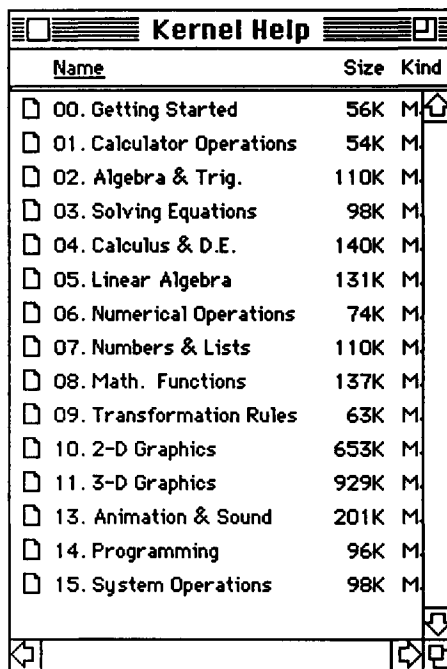
The following window appears when the command `Plot[Sin[x], {x, -2Pi, 2Pi}]` is selected and **Explain Selection...** is chosen from under the apple icon on the *Mathematica* Menu. The window gives a description of the cell and explains how the cell can be altered.



- **Version 2.0 Help and Kernel Help**

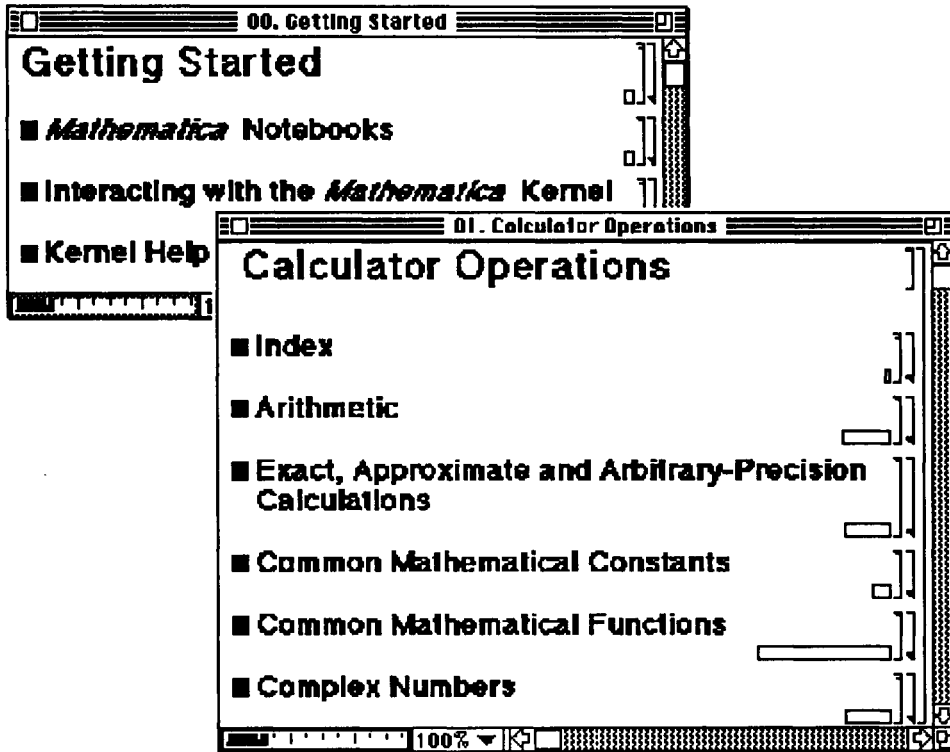
The folder Kernel Help is installed on the desk top when Version 2.0 of *Mathematica* is installed on the computer.

The folder **Kernel Help** contains sixteen notebooks that provide examples of nearly every built-in *Mathematica* command. Since these notebooks can be opened during a *Mathematica* session, they can be particularly helpful sources of documentation.

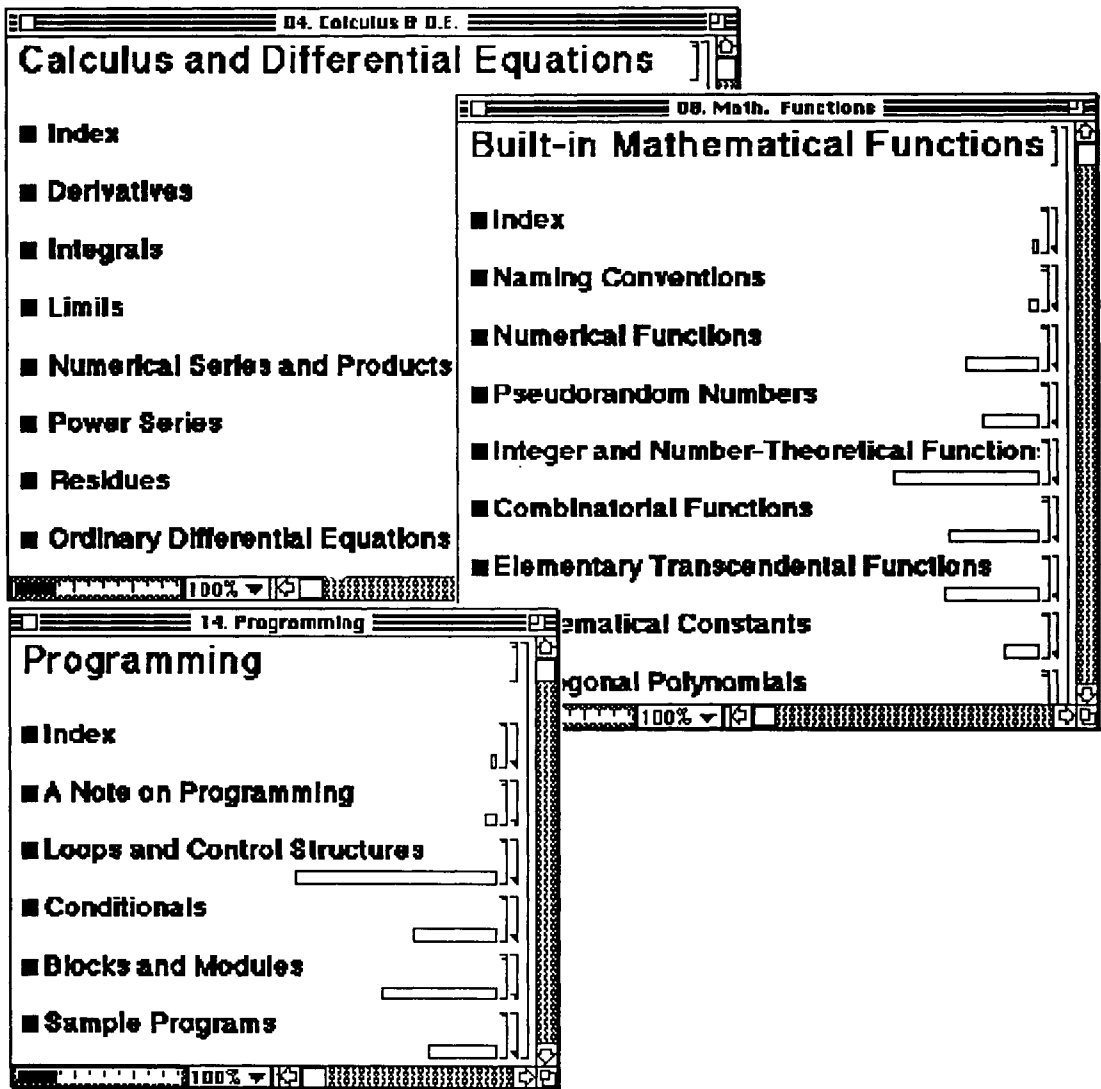


Name	Size	Kind
00. Getting Started	56K	M
01. Calculator Operations	54K	M
02. Algebra & Trig.	110K	M
03. Solving Equations	98K	M
04. Calculus & D.E.	140K	M
05. Linear Algebra	131K	M
06. Numerical Operations	74K	M
07. Numbers & Lists	110K	M
08. Math. Functions	137K	M
09. Transformation Rules	63K	M
10. 2-D Graphics	653K	M
11. 3-D Graphics	929K	M
13. Animation & Sound	201K	M
14. Programming	96K	M
15. System Operations	98K	M

The first two notebooks, **Getting Started** and **Calculator Operations**, look as follows when they are opened:

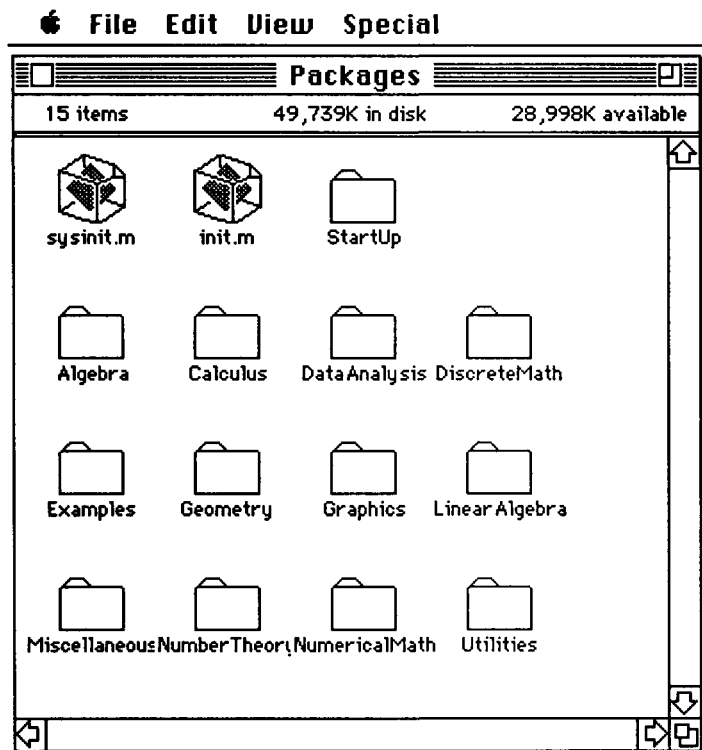


Similarly, **Calculus and Differential Equations**, **Built-in Mathematical Functions**, and **Programming** look the same:



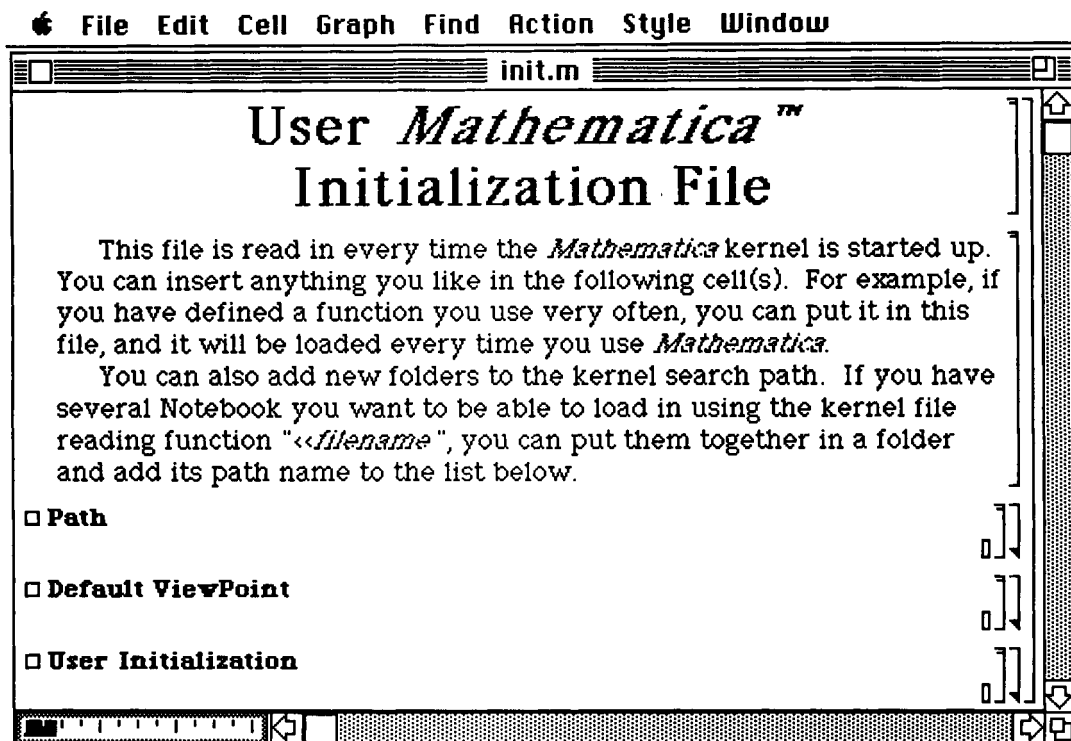
■ 10.2 The `init.m` file

The `init.m` file gives the user the opportunity to supply information to be read each time the *Mathematica* kernel is started. Since this file is read each time the kernel is started, any information included in the file is automatically loaded. The `init.m` file is found inside of the **Packages** folder. The window obtained by double-clicking on the **Packages** folder is shown below:

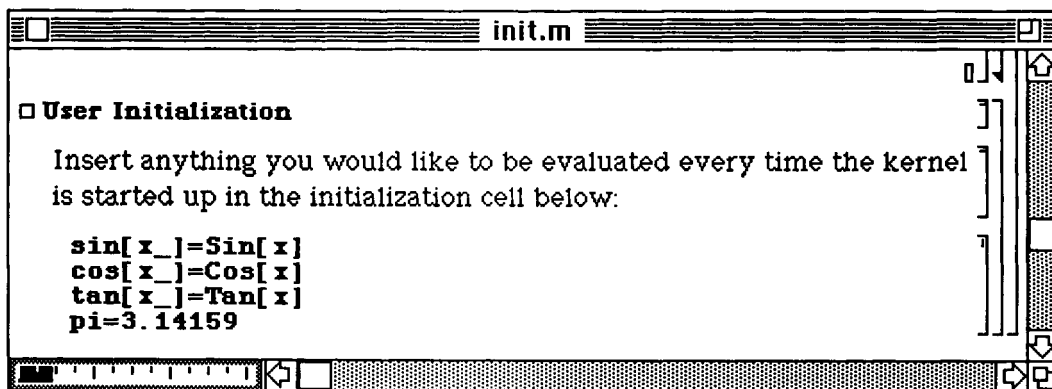


The file `init.m` is contained in the folder **Packages**.

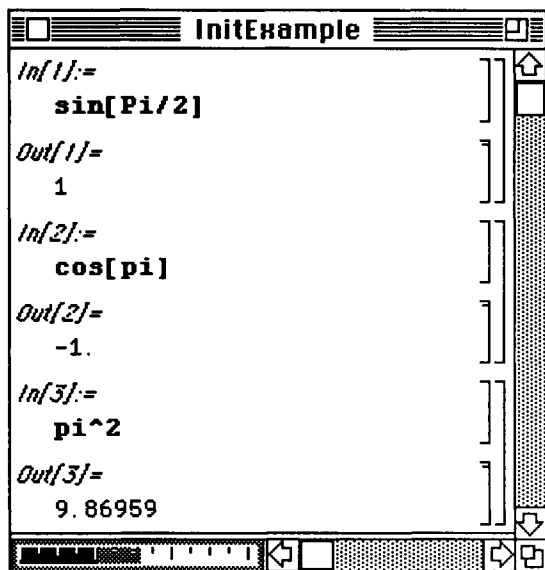
Double-clicking on the `init.m` icon yields the following window which gives a brief description of the file and how new information can be entered.



A useful property of the `init.m` file is user initialization of functions and commands. This gives the user the opportunity to "customize" his or her own commands or define functions which the user often employs. The following is an illustration of how this is done. If the user prefers not to capitalize the trigonometric functions, they can be redefined with small letters in terms of the associated built-in *Mathematica* commands. The same can be done with the constant `Pi`. Below, the user chooses to use a five-place decimal approximation of the well-known constant and name it `pi`. Hence, whenever the *Mathematica* kernel is opened, these new definitions will be read in and, thus, will be recognized when used.



After initializing, *Mathematica* uses the user-defined functions to give the correct results. Of course, the built-in commands would be properly evaluated as well, so these newly-defined commands have not replaced the original ones.



Since the file **init.m** is automatically loaded when the kernel is started, these commands yield the desired values instead of error messages.

■ 10.3 Explanation of the *Mathematica* Menu

■ The Version 1.2 Menu

For a complete discussion of the menu as it appears in either Version 1.2 or Version 2.0, see the User's Guide for the Macintosh in the *Mathematica* software package.

The menu below which indicates the editing capabilities of *Mathematica* appears under **Edit** in the *Mathematica* Menu. The usual Macintosh editing features are included in this list along with the addition of several commands for working with *Mathematica* cells. The main item to notice in the list is **Settings**. These **Settings** options are discussed in greater detail below.

Edit

Can't Undo
Cut
Copy
Paste
Clear
Paste and Discard
Convert Clipboard
Select All Cells
Nesting
Divide Cell
Merge Cells
Settings

In Version 2.0, **Settings** has been changed to **Preferences**; **Divide Cell** and **Merge Cells** are found under **Cell**.

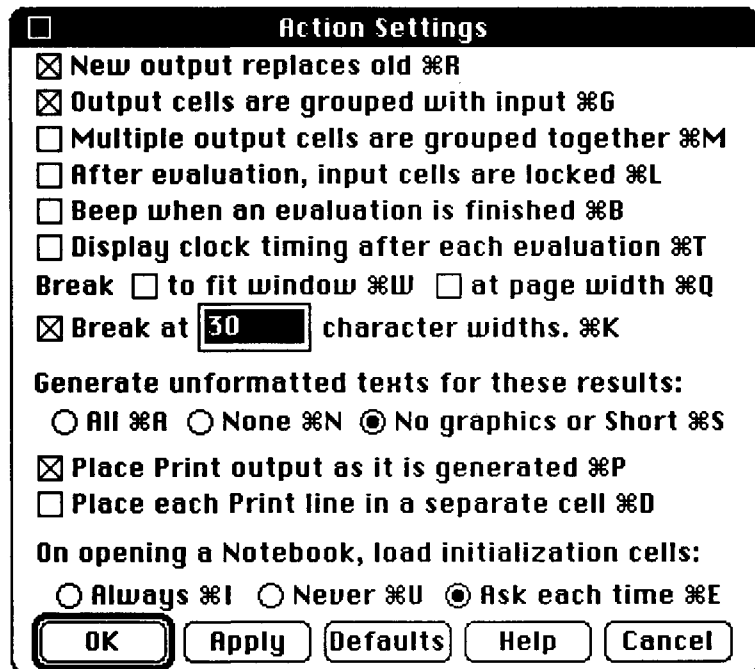
The **Settings** options which are available under **Edit** are accessed by moving the cursor to the **Edit** heading on the *Mathematica* Menu and dragging the cursor to the last entry which is **Settings**. A list of six **Settings** options is then displayed: **Display**, **Graphics**, **Color**, **Animation**, **Action**, and **Startup**. To obtain a particular **Settings** window, move the cursor to the desired **Settings** option and release the mouse button.

The first of these options discussed are those found under **Startup Settings**. Instead of loading certain files or tables during a *Mathematica* session, the user has the option of requesting that they be loaded automatically. This is done on the window below by placing an X in the box next to the package which is to be loaded each time a *Mathematica* session begins. (To place the X in a box, move the cursor inside the box and click once with the mouse; to remove the X, place the cursor on the X which is to be deleted and click once with the mouse.)



The checked boxes indicate the packages that Mathematica automatically loads when the Mathematica kernel is loaded. Many users like to have both `msg.m` and `info.m` loaded each time. In addition, if you are frequently computing definite integrals, you may also want Mathematica to load the package `IntegralTables.m`.

Several settings can be made under **Action Settings**. These are listed on the **Action Settings** window which is shown below. The user has selected those options which are checked. Some of the more useful of these options include, displaying the computation time of calculations and having the output displayed so that it fits in a window of specified width.

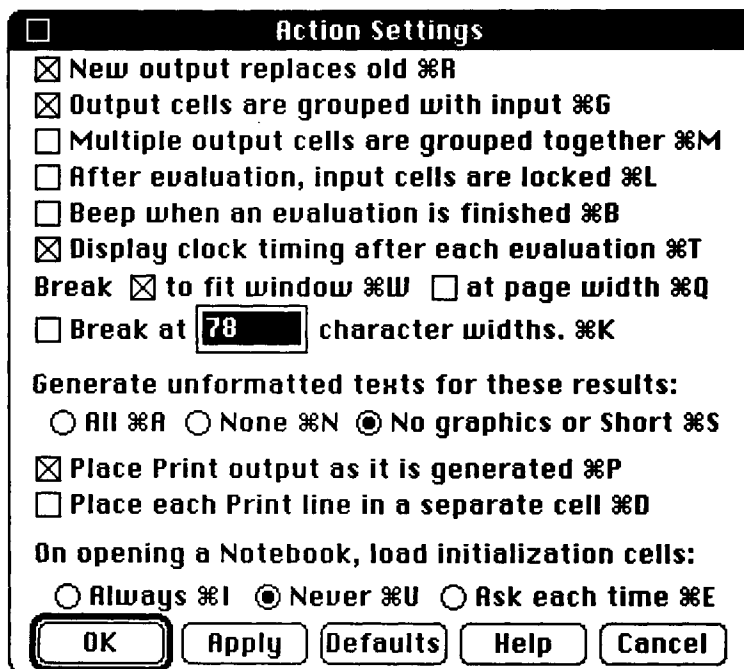


Users can customize Mathematica by modifying various options.

A checked box means that Mathematica will perform the option.

In the following window, the user has requested that 78 characters be included on each line of output. This may lead to the results being in a form which are more easily read, since longer expressions can be printed on one line.

Action settings:



The image shows a dialog box titled "Action Settings" with a standard Mac OS-style title bar. The dialog contains several sections of settings:

- New output replaces old %R
- Output cells are grouped with input %G
- Multiple output cells are grouped together %M
- After evaluation, input cells are locked %L
- Beep when an evaluation is finished %B
- Display clock timing after each evaluation %T
- Break to fit window %W at page width %Q
- Break at character widths. %K

Generate unformatted texts for these results:

- All %A
- None %N
- No graphics or Short %S

- Place Print output as it is generated %P
- Place each Print line in a separate cell %D

On opening a Notebook, load initialization cells:

- Always %I
- Never %U
- Ask each time %E

At the bottom of the dialog are five buttons: "OK", "Apply", "Defaults", "Help", and "Cancel". The "OK" button is highlighted with a thick border.

When working with the animation of graphics, the user may find the options located under **Animation Settings** helpful. Most importantly, the user can set the speed of animation by increasing or decreasing the number of frames viewed per second. This number can be changed by simply typing the desired value for the speed (when the box is darkened) or by moving the cursor to the box and clicking the mouse button once to obtain a vertical flashing cursor. Then, any changes can be made in the usual manner. Other options include the order in which the animation is viewed. The user selects the desired order by placing a dot in the circle corresponding to the appropriate order (**Forward**, **Backward**, or **Cyclic**). Dots are placed and removed in the same manner as boxes are checked.

Animation options can also be modified through the different settings.

Cell options are found under the **Cell** heading in the *Mathematica* Menu. These options allow the user to create a *Mathematica* notebook in any form desired. These options are used by selecting a cell or cells in a *Mathematica* notebook and then choosing the appropriate **Cell** option. One of the more useful **Cell** options is that of grouping cells. This **Group Cells** option is illustrated in greater detail in a subsequent window. Another option is that of locking cells. If the **Locked** option is chosen, then no changes can be made in that cell. (Output cells are automatically locked.)

- In Version 2.0, the options **Page Break** are found under **Style** and the options **Formatted**, **Inactive**, **Initialization**, **Locked**, **Closed**, and **Fixed Height** are contained under **Attributes** which is also found under **Style**.

Cell

Formatted
Inactive
PostScript
Locked
Closed
Initialization
Fixed Height
Page Break
Group Cells
Ungroup Cells
Open All Subgroups
Close All Subgroups
Closed Group
Evaluation Group

The Cell Options allow notebooks to be customized.

Initialization cells are cells within a notebook that are automatically evaluated when the Mathematica kernel is started.

Cells may be grouped, ungrouped, opened or closed.

The following sequence of windows illustrates how **Group Cells** is implemented. Consecutive cells can be grouped by first selecting the cells and then choosing the **Group Cells** option. When cells are grouped, they can be closed simultaneously by double-clicking on the outermost cell which encloses the group. In the first window below, all of the cells are selected by first selecting the uppermost cell and then dragging down through all of the cells. After these cells are selected (as shown in the second window), choosing **Group Cells** from the list of **Cell** options causes the selected cells to be enclosed in a single (outermost) cell. This is displayed in the third window below.

The image shows three Mathematica notebook windows, each titled "Grouping", illustrating a sequence of operations:

- Window 1 (Left):**
 - Input: `Clear[f]` and `f[x_] := (x-1)/(2x-1)`
 - Input: `Plot[f[x], {x, -1/1, 3/2}, AspectRatio->1]`
 - Output: A plot of the function $f(x) = \frac{x-1}{2x-1}$ with a vertical asymptote at $x = 0.5$. The x-axis ranges from -1 to 1.5, and the y-axis ranges from -10 to 20.
 - Output: `Out[18]= -Graphics-`
 - Input: `Together[f'[x]]`
 - Output: `Out[19]=` followed by the expression $(-1 + 2x)^{-2}$.
- Window 2 (Middle):**
 - Input: `Plot[f[x], {x, -1/1, 3/2}, AspectRatio->1]`
 - Output: The same plot of $f(x)$ as in Window 1.
- Window 3 (Right):**
 - Input: `Plot[f[x], {x, -1/1, 3/2}, AspectRatio->1]`
 - Output: The plot of $f(x)$.
 - Input: `Plot[f'[x], {x, -1/1, 3/2}, AspectRatio->1]`
 - Output: A plot of the derivative $f'(x) = (-1 + 2x)^{-2}$, which is a curve with a vertical asymptote at $x = 0.5$ and a horizontal asymptote at $y = 0$.

Action

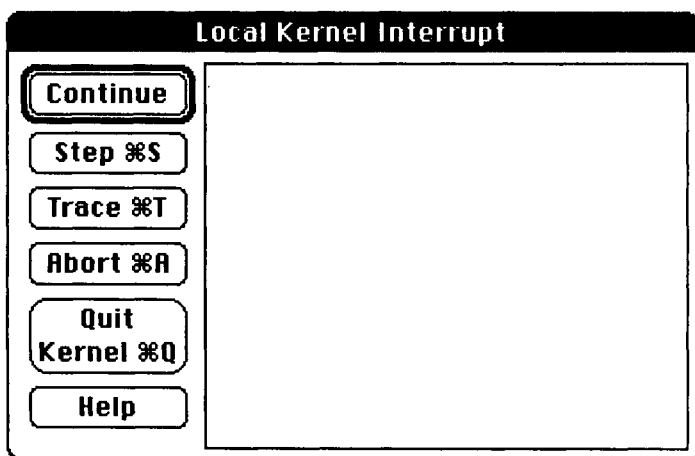
Prepare Input
Evaluate Selection
Evaluate Next Input
Evaluate Notebook
Evaluate Initialization
Interrupt
Kernels
Clear Kernel History
Quit Kernel

When **Prepare Input** is selected, this menu appears:

Copy Input from Above
Copy Output from Above
Complete Selection
Make Template
3D ViewPoint Selector...
Color Selector...

Interrupt can be used to stop *Mathematica* calculations. The **Interrupt** menu is discussed below.

An important option located under **Action** is **Interrupt**. This can be used to interrupt a calculation if the calculation is taking longer than it should or if the user notices a mistake in the command which would lead to undesirable results. The following window appears when **Interrupt** is selected from the menu. This window includes several options. **Step** and **Trace** display the calculations that *Mathematica* is performing. These calculations appear within the window to the right of the four options. **Abort** causes the calculation to be stopped. In the case of **Abort**, the current *Mathematica* session may be continued. On the other hand, **Quit Kernel** causes the calculation to cease, and the user must exit and restart *Mathematica* if more calculations are desired.



If a calculation is taking longer than expected and one wishes to abort the calculation, click abort. To quit the kernel completely (in which case, one must completely exit the *Mathematica* session and restart to continue using *Mathematica*) click quit kernel. In addition, one may view the calculations *Mathematica* is performing by clicking on either step or trace.

- In Version 2.0, users can abort a calculation directly from the **Action** menu. However, to **Trace** a calculation, Version 2.0 users must select **Enter Dialog** from the Version 2.0 **Action** menu and then use the commands **Stack** or **Trace**.

Mathematica notebooks can be customized by taking advantage of the features under **Style** and **Window**.

Style

Font
Face
Size
Color
Format
Cell Style
Uniform Style
Default Style
All Default Styles...

In Version 1.2, **textFont**, **Face**, **Size**, **Color**, and **Format**, in addition to **Cell Style** can be changed from the **Style** menu.

Format contains alignment, scrolling, and word-wrapping options as well as cell variations or "dingbats"

Cell styles are modified by going to **Window** and selecting **Styles Window**. Once the **Styles Window** appears on the screen, fonts, sizes, and faces of the various cell types are modified using **Font**, **Face**, and **Size** found under **Style**. Selecting **All Default Styles...** resets all cell styles to their defaults.

- In Version 2.0, **Style** is considerably expanded, containing many of the options found under **Cell** and **Window** in Version 1.2.

Window

Stack Windows
Tile Windows Wide
Tile Windows Tall
Network Window
Defaults Window
Styles Window
Clipboard Window
<i>(open notebooks)</i>

Stack Windows, Tile Windows Wide, and Tile Windows Tall are various ways of viewing several open Mathematica notebooks simultaneously.

The Defaults Window shows Mathematica's default values which may be modified. Similarly, the Styles Window shows the font and size for each type of cell which may also be changed.

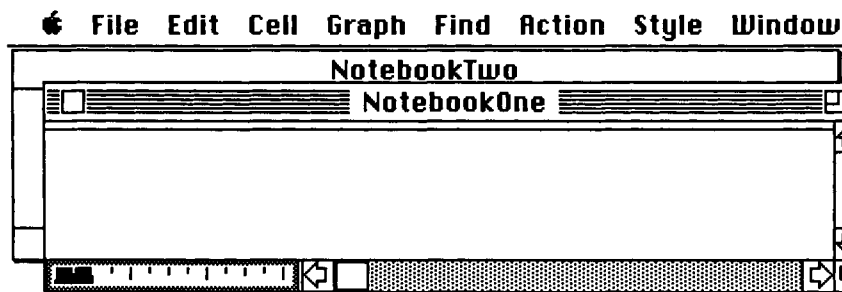
All open notebooks are shown; when a notebook is selected, it is brought to the front of the screen and becomes the active window.

Several options are found under the **Window** heading on the *Mathematica* Menu. The first group of options deals with viewing several *Mathematica* notebooks at once. These options, **Stack Windows**, **Tile Windows Wide**, and **Tile Windows Tall**, are illustrated individually after the menu below.

Two of the windows which may be viewed are the **Defaults Window** and the **Styles Window**. The **Defaults Window** is displayed and explained below. The **Styles Window** displays all of the styles (font, face, and size) used for each particular type of cell in the *Mathematica* notebook. These styles can be changed by selecting a cell (or cells) and choosing another font, face, or size. This window, therefore, allows the user to customize the notebook.

The last entry in the list of **Window** options is a list of all open *Mathematica* notebooks. Hence, an opened notebook can be brought to the front of the screen by selecting it from the list with the cursor.

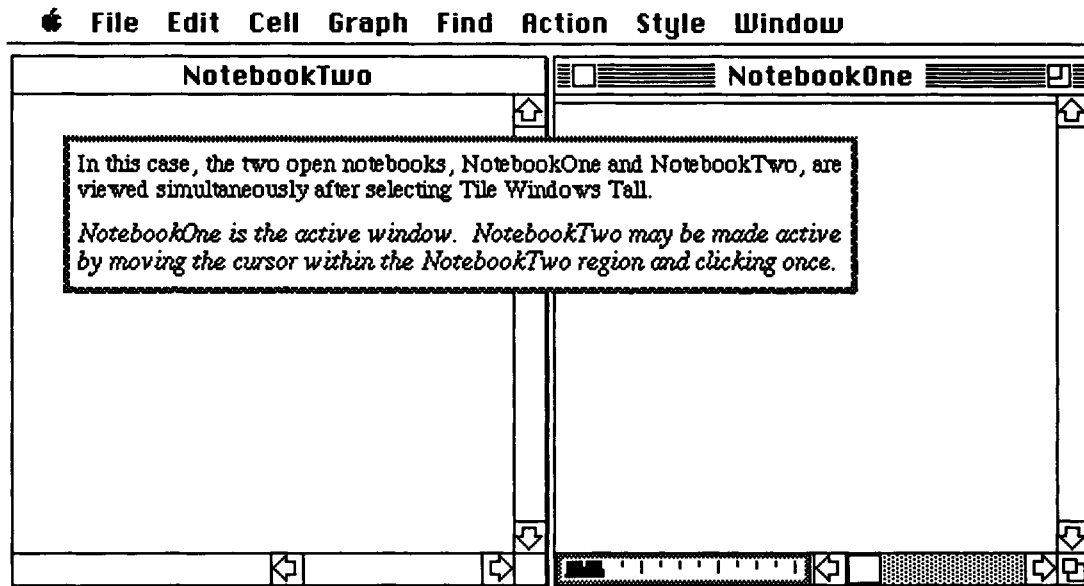
If **Stack Windows** is chosen, then the notebook windows are stacked one behind the other so that only the notebook in front can be fully viewed. The other open notebooks can only be partially viewed. However, a notebook can be brought to the front by simply moving the cursor to that notebook and clicking the mouse button once.



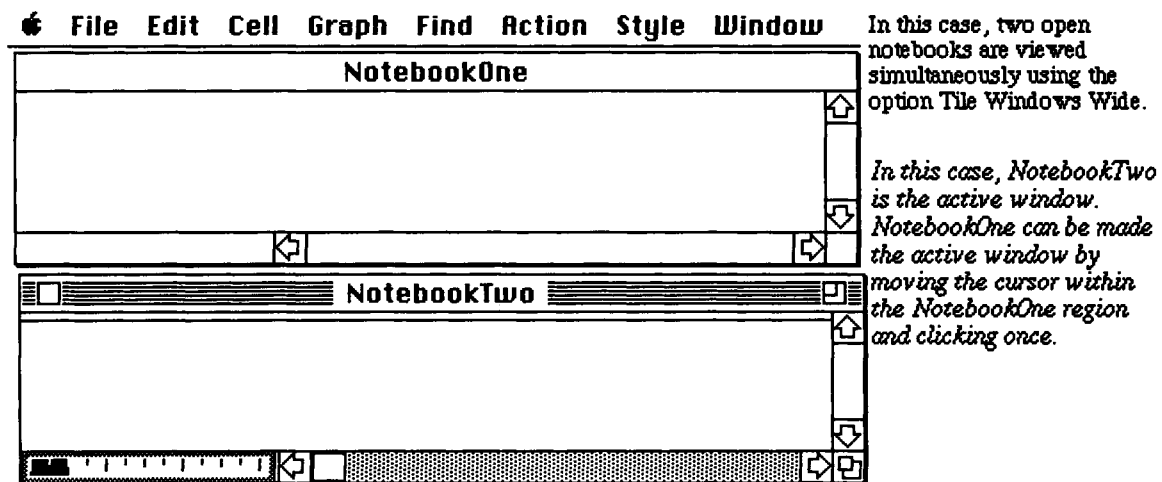
When Stack Windows is selected, the open notebooks are "layered".

In this case, NotebookOne is the active window and NotebookTwo may be made active by clicking in the NotebookTwo region.

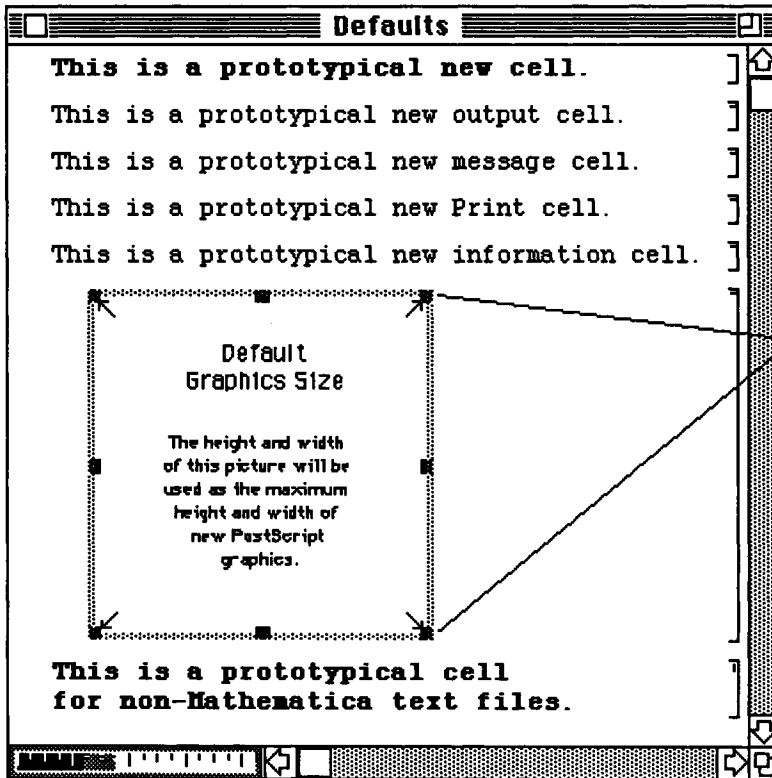
Tile Windows Tall, illustrated below with **NotebookOne** and **NotebookTwo**, changes the width of each notebook window so that they fit side-by-side on the screen. (Note that each window has a complete boundary.)



Tile Windows Wide alters the height of each notebook window so that the reduced windows fit on the computer screen simultaneously. (Again, each window has a complete boundary.) **NotebookOne** and **NotebookTwo** are displayed in the following manner with **Tile Windows Wide**.



The defaults of the fonts, faces, and sizes used in each of the cell types can be viewed in the **Defaults Window**. This window also includes the graphics size. The window obtained when **Default Window** is selected is shown below with a description of the steps necessary for changing the default size of graphics cells. Changes in the cell styles are accomplished through opening the **Styles Window**.

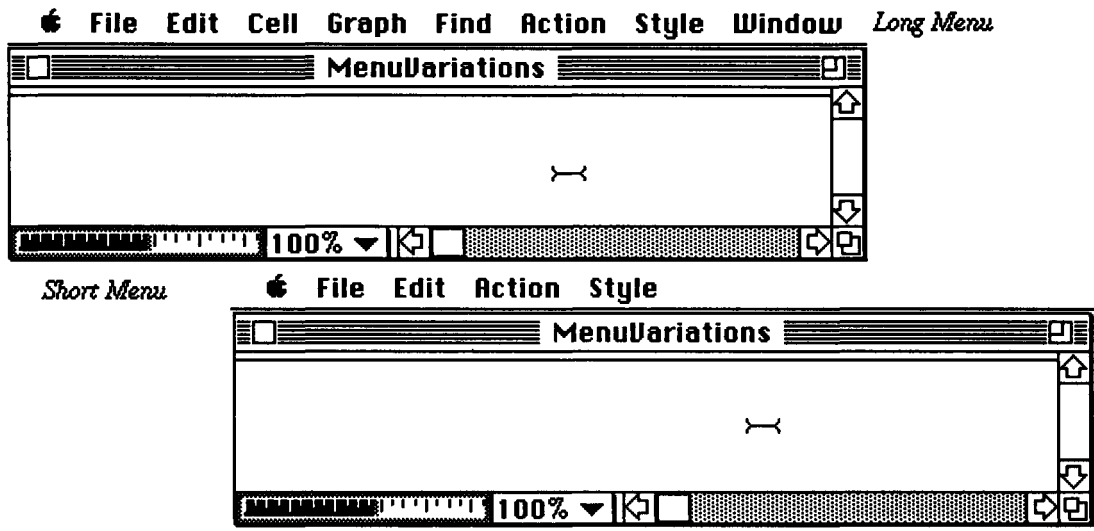


The defaults for each cell type may be changed using the fonts and sizes available under Styles.

The default size for graphics cells can be changed by using the mouse to move the cursor to a corner. When the cursor is on a corner, click and drag to the desired size.

● **The Version 2.0 Menu**

The *Mathematica* Menu in Version 2.0 appears at first glance to be identical to that of Version 1.2. Upon further inspection, however, the user easily notices that many of the features in Version 1.2 have been rearranged under the menu headings. Version 2.0 also has a **Short Menu** option which is located under **Edit**. The **Long Menu** is displayed if a check mark is placed beside **Long Menu**. This can be changed to **Short Menu** by clicking once on this check mark. Both menus are shown below.



Edit	
Undo Typing	
Cut	
Copy	
Paste	
Clear	
Paste and Discard	
Convert Clipboard...	
Select All Cells	
Nesting	
Preferences	
Long Menus	

Version 2.0 Edit Settings differ somewhat from earlier versions. **Action** and other settings are contained under **Preferences**.

■ Action Settings for Version 2.0:

The **Action Preferences** shown below are the same as those in **Action Settings** in Version 1.2, however.

Action Preferences

Output cells are grouped with input
 Multiple output cells are grouped together
 After evaluation, input cells are locked
 Beep when an evaluation is finished
 Display clock timing after each evaluation
 Append kernel name to In/Out names

Break to fit window at page width
 Break at character widths

Generate unformatted text for these results:
 All None No graphics or Short

Place Print output as it is generated
 Place each Print line in a separate cell

On opening a Notebook, load initialization cells:
 Always Never Ask each time

The Action Settings available in Version 2.0.

The changes in **Startup Preferences** are clear. The option of automatically loading **msg.m**, **info.m**, **IntegralTables.m**, **Elliptic.m**, and **Series.m** at the beginning of each *Mathematica* session is not offered in Version 2.0 as it was in Version 1.2.

*The Startup Preferences in Version 2.0 are substantially different from prior versions. In particular, packages such as **IntegralTables.m** are automatically loaded when the kernel is started.*

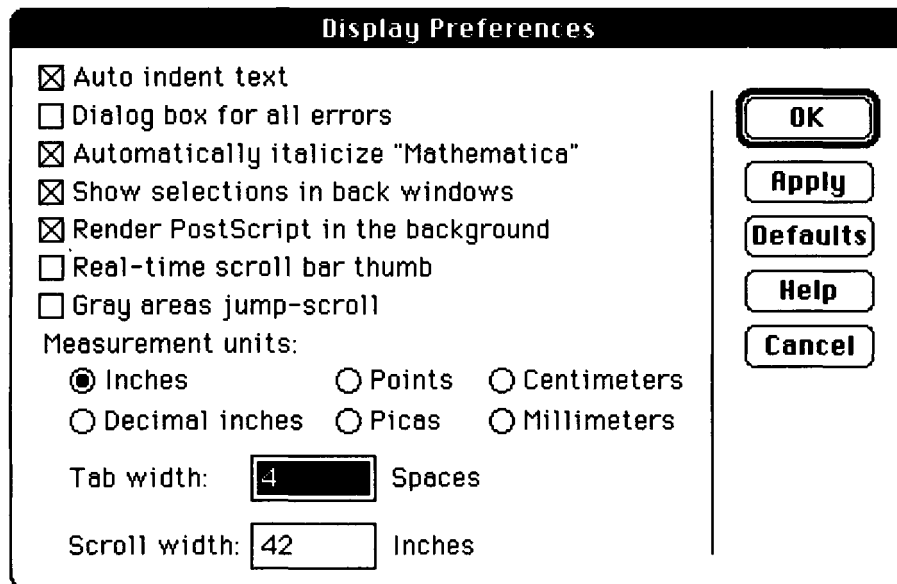
Startup Preferences

Stack size (number of KBytes):
 Current: 512 Requested:

Automatically start a kernel on launch

When automatically starting a kernel, start the following kernel:

Display Preferences are basically the same as **Display Settings** in Version 1.2 with the exception of the cell sizing options which were included in Version 1.2. Version 2.0 includes several options such as **Gray Areas jump-scroll** and **Measurement units**, however, which were not found in Version 1.2.



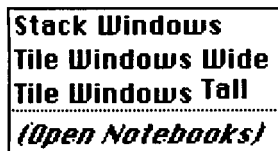
Display Preferences available in Version 2.0.

Version 2.0 **Cell** settings differ substantially from earlier versions. Notice that **Divide Cell** and **Merge Cells**, which were contained under **Edit** in prior versions, are now contained under **Cell**.



The menu which accompanies the **Windows** heading no longer contains the windows for **Network**, **Defaults**, **Styles**, and **Clipboard**. However, the remaining menu members perform the same tasks as those in Version 1.2.

Window



Stack Windows, **Tile Windows wide**, and **Tile Windows Tall** perform the same task as in Version 1.2.

The main difference in **Action** is found under **Interrupt Calculation**. The window which was displayed with **Interrupt** in Version 1.2 no longer exists. Instead, the options found in this window are listed when **Interrupt Calculation** is selected. These are shown under **Interrupt Calculation** in the window below even though this list is actually hidden until **Interrupt Calculation** is selected.

Action

Prepare Input	▶
Edit Connections...	
Terminals	▶
Current Kernel	▶
Notebook's Kernel	▶
Connect Remote Kernel...	
Quit/Disconnect Kernel	
Evaluate Selection	
Evaluate Next Input	
Evaluate in Dialog	
Don't Evaluate	
Evaluate Notebook	
Evaluate Initialization	
Interrupt Calculation	
Abort Calculation	
Abort to Top	
Enter Dialog	
Exit Dialog	
Auto Save after Each Result	

In Version 2.0, a calculation may be aborted by selecting **Action** and then **Abort Calculation**. Version 2.0 is able to stop calculations much faster than prior version. Also note that if the option **Auto Save After Each Result** is checked, then *Mathematica* will save the file after each calculation. When lengthy calculations are being performed and there is fear that the computer may crash, this option can often help avoid heartache.

- While *Mathematica* is performing calculations, Version 2.0 users can select **Enter Dialog** which pauses the current calculation and allows the user to perform other calculations. **Exit Dialog** causes the suspended calculation to resume.

Style

Cell Style	▶
Attributes	▶
Font	▶
FaceSize	▶
Leading	▶
AlignmentText Color	▶
Background Color	▶
Page Breaks	▶
Formatter	▶
Evaluator	▶
Show Ruler	
Edit Styles...	
Uniform Style	
All Default Styles...	

In earlier versions, **Page Break** was contained under **Cell**; now it is contained under **Style**.

Similarly, **Edit Styles** was contained under **Windows** as **Styles Window**. In Version 2.0, the styles may be changed by selecting **Styles** and then **Edit Styles**.

The menu under **Style** has several changes although many of these changes are only in appearance. **Attributes** contains the option **Dingbats...** which allows for the use of symbols in creating a notebook. These symbols include the circle and block which were found in the section and subsection styles in Version 1.2. Also under **Attributes** are many of the options such as **Formatted**, **Locked**, **Closed**, and **Fixed Height** found under **Cell** in Version 1.2. Useful additions located under **Face** include **Superscript**, **Subscript**, and **Overstrike**. Another obvious change is **Show Ruler**. If this is chosen, then a ruler which includes three alignment options (left, right, and center) is displayed in the notebook. The **Styles Window** which no longer appears under **Windows** is opened by selecting **Edit Styles...** in Version 2.0.

■ 10.4 Some Common Errors and Their Remedies

Learning to recognize and correct errors will alleviate many of the frustrations that some first-time users encounter when working with *Mathematica* and will enable the user to make the most of *Mathematica*'s vast capabilities. Some of the more common errors and their remedies are illustrated below.

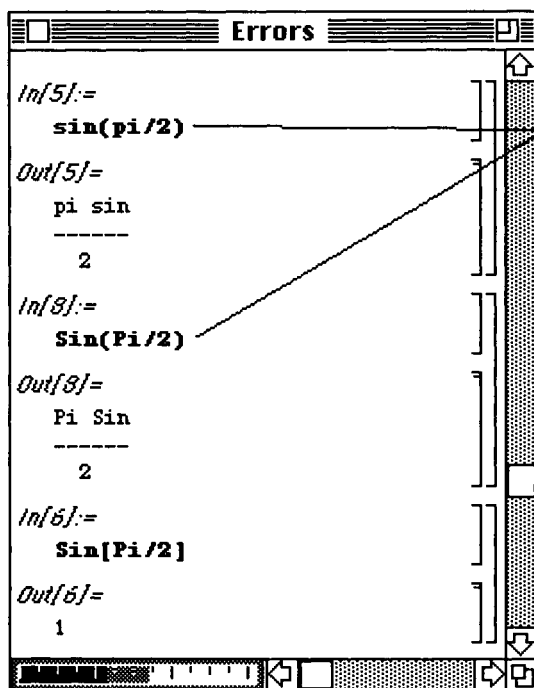
- o A list of all *Mathematica* warning messages is contained in Technical Report: *Mathematica* Warning Messages, by David Withoff which is included in the *Mathematica* packaging box.

One of the most commonly made mistakes occurs when using built-in *Mathematica* commands, functions, or constants. The user must always remember to use square brackets and/or capital letters. Several examples are shown below which demonstrate these types of errors.

In the first example, *Mathematica* interprets the command `sin(pi/2)` as `sin*(pi/2)` which certainly was not intended. The user also failed to capitalize the Sine function and the constant Pi.

The second example demonstrates that even if capital letters are used correctly, the absence of square brackets yields almost the same output as the first command. In this case, however, the expressions are capitalized.

Finally, the third example shows that when square brackets are used, *Mathematica* interprets the command correctly and gives the exact value of `Sin[Pi/2]` which, of course, is 1.



Sometimes Mathematica will return a command to you that is entered incorrectly.

Neither of these commands produce the desired output because every Mathematica command begins with a capital letter and the argument is enclosed by square brackets.

Entering Sin[Pi/2] produces the desired output.

Error messages can be disturbing to receive. In some cases, however, they can be ignored. For example, in the following window, the user attempts to load a package, `IntegralTables.m`, which has already been loaded. Hence, the error messages are given, but they may be deleted.

- Version 2.0 users need never load the package `IntegralTables.m`. Nevertheless, if one reads in a package and then reads it in again, messages of this sort often appear.

```

In[14]:=
<<IntegralTables.m

Set::write:
  Symbol IntBase
  is write protected.

Attributes::locked:
  Attributes of IntBase
  are locked.

Set::write:
  Symbol ExpQuad
  is write protected.

Attributes::locked:
  Attributes of ExpQuad
  are locked.

Set::write:
  Symbol ExpLinear
  is write protected.

General::stop:
  Further output of Set::wr
  will be suppressed duri
  calculation
  
```

If the package `IntegralTables.m` has already been loaded and the user attempts to load it again, these error messages appear. They may be ignored or deleted.

The following error is quite common to new *Mathematica* users. The user attempts to plot a function g without properly defining the function beforehand. Hence, there is no function to graph. After receiving the error message, an easy way to check that this is the problem is to use `?g`. If the function is undefined, the output is simply g as shown below. Otherwise, the formula for g would be displayed.

```

In[1]:=
Plot[g[x],{x,0,5}]

Plot::notnum:
g[x] does not evaluate to a
real number at x=0..

Plot::notnum:
g[x] does not evaluate to a
real number at x=0.208333

Plot::notnum:
g[x] does not evaluate to a
real number at x=0.416667

General::stop:
Further output of Plot::
notnum will be suppressed
during this calculation.

Out[1]=
-Graphics-

In[2]:=
?g

g

```

This message indicates that g is not a properly defined function.

*Notice that when we enter the command `?g` we confirm that g is not a well-defined function. In this case, we would re-enter the correct definition of g and then re-enter the **Plot** command.*

Another common mistake occurs when trying to work with the elements of a table produced with two indices. In the example below, a table of Legendre polynomials is formed in `lps`. Note that the output is in the form of a list in which every element is itself a list of two elements. (*Mathematica* computes the polynomials in pairs, one pair for each value of `n`.) Therefore, the `Plot` command as it is stated below cannot plot the members of `lps`.

Tables of Functions

```
In[19]:=
lps=Table[LegendreP[n,m,x],{n,2,6,2},{m,1,2}]
Out[19]=
```

$$\left\{ \left\{ -3 x \sqrt{1-x^2}, 3(1-x^2) \right\}, \left\{ \frac{5 \sqrt{1-x^2} (3x-7x^3)}{2}, \frac{15(-1+8x^2-7x^4)}{2} \right\}, \left\{ \frac{21 \sqrt{1-x^2} (-5x+30x^3-33x^5)}{8}, \frac{105(1-19x^2+51x^4-33x^6)}{8} \right\} \right\}$$

Computes a table of LegendreP[n,m,x] for n=2,4,6 and m=1,2.

```
In[20]:=
Length[lps]
Out[20]=
3
```

lps is a nested list with three elements. Each element of lps is a list of two expressions in x.

```
In[21]:=
Plot[Evaluate[lps],{x,-1,1}]
Plot::plnr: CompiledFunction[{x}, <<1>>, -CompiledCode-]
[<<1>>] is not a machine-size real number at x = -1.
General::stop:
Further output of Plot::plnr
will be suppressed during this calculation.
```

Even though we have included the command Evaluate Mathematica generates error messages and fails to produce the desired graphs.

To remedy the problem, `Flatten[lps]` must be used before trying to plot the Legendre polynomials in `lps`. Notice that `Flatten[lps]` removes the inner brackets contained in `lps` and converts it to a list of length 6 called `lpstwo`. A table of `GrayLevel` values is created in `grays`.

```

In[22]:=
  lpstwo=Flatten[lps]
Out[22]=
  {-3 x Sqrt[1 - x2], 3 (1 - x2),
    $\frac{5 \text{ Sqrt}[1 - x^2] (3 x - 7 x^3)}{2}$ ,  $\frac{15 (-1 + 8 x^2 - 7 x^4)}{2}$ ,
    $\frac{21 \text{ Sqrt}[1 - x^2] (-5 x + 30 x^3 - 33 x^5)}{8}$ ,
    $\frac{105 (1 - 19 x^2 + 51 x^4 - 33 x^6)}{8}$ }

In[23]:=
  Length[lpstwo]
Out[23]=
  6

In[26]:=
  grays=Table[GrayLevel[i/10], {i,0,5}]:

```

*converts **lps** from a nested list to a list of expressions in **x**. The resulting list is named **lpstwo**.*

***lpstwo** consists of six elements; each element is an expression in **x** and can thus be graphed.*

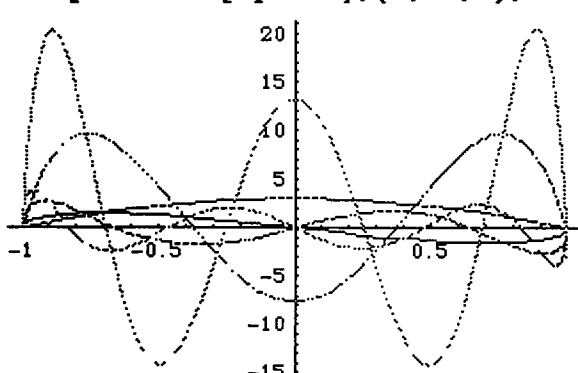
After making the appropriate changes, the six Legendre polynomials found in `lpstwo` are correctly plotted using `grays` in the command below.

```

In[27]:=
  Plot[Evaluate[lpstwo], {x,-1,1}, PlotStyle->grays]
Out[27]=
  -Graphics-

```

***lpstwo** is graphed according to the shades of gray created in **grays**.*



Throughout *Mathematica by Example*, user-defined functions have always been defined using lower-case letters. Since every built-in command begins with a capital letter, we have been sure to avoid any ambiguity with built-in functions. Nevertheless, if one does attempt to define a function that conflicts with a built-in function, errors like the following result:

```

In[4]:=
  Sin[x_] := x^2 Cos[x]
Set::write: Symbol Sin is write protected

In[5]:=
  Sin[Pi/2]

Out[5]:=
  1

```

In this case, we have attempted to define
 $\text{Sin}(x) = x^2 \text{Cos}(x)$.
*The definition conflicts with the built-in function **Sin[x]** Consequently, the user-definition is refused and **Sin[Pi/2]** returns the "correct" value.*

■ 10.5 Additional References

■ Additional References regarding Macintosh Computers:

- Getting Started With Your Macintosh, Apple Computer, Inc.
- Macintosh Reference, Apple Computer, Inc.

■ Additional references regarding *Mathematica*:

- Crandall, Richard E., *Mathematica for the Sciences*, Addison-Wesley Publishing Co. (1991);
- Gray, Theodore and Glynn, Jerry, *Exploring Mathematics with Mathematica*, Addison-Wesley Publishing Co. (1991);
- Maeder, Roman, *Programming in Mathematica*, Addison-Wesley Publishing Co. (1990);
- Wagon, Stan, *Mathematica in Action*, W. H. Freeman and Co. (1991);
- Wolfram Research, Inc., *Mathematica: A System for Doing Mathematics by Computer. User's Guide for the Macintosh* (1991);
- Wolfram, Stephen, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley Publishing Co. (1988);
- Wolfram, Stephen, *Mathematica: A System for Doing Mathematics by Computer*, Second Edition, Addison-Wesley Publishing Co. (1991); and
- *The Mathematica Journal*, published quarterly by the Advanced Book Program, Addison-Wesley Publishing Co.

■ Additional references regarding the mathematical topics that appeared in *Mathematica by Example*:

- Arnold, Steven F., *Mathematical Statistics*, Prentice-Hall (1990);
- Cheney, Ward and Kincaid, David, *Numerical Mathematics and Computing*, Second Edition, Brooks/Cole Publishing Co. (1985);
- Hillier, Frederick S. and Lieberman, Gerald L., *Introduction to Operations Research*, Fifth Edition, McGraw-Hill Publishing Co. (1990);
- Jordan, D. W. and Smith, P., *Nonlinear Ordinary Differential Equations*, Second Edition, Oxford University Press (1988);
- Kreyszig, Erwin, *Advanced Engineering Mathematics*, Sixth Edition, John Wiley & Sons (1988);
- Powers, David L., *Boundary Value Problems*, Second Edition, Academic Press (1979);
- Strang, Gilbert, *Linear Algebra and its Applications*, Third Edition, Harcourt Brace Jovanovich, Publishers (1988);
- Weiss, Neil A. and Hassett, Matthew J., *Introductory Statistics*, Second Edition, Addison Wesley Publishing Co. (1991); and
- Wilson, R. J., and Watkins, J. J., *Graphs: An Introductory Approach*, John Wiley & Sons (1990).

Appendix

Introduction to Programming in *Mathematica*

The **Appendix** provides a brief introduction to programming in *Mathematica*. Examples include some of the programs that were used to create some of the graphics objects in *Mathematica By Example*. However, users that intend to become proficient *Mathematica* programmers should refer to Maeder's book *Programming in Mathematica*.

In Version 1.2, local variables are declared using the command **Block**. The following example illustrates the use of local variables within a **Block**. Notice that changes in the local variable **j** in the function **value** do not affect the value of the previously defined global variable **j = 0**. This value remains zero although the local variable **j** has value 4.

- o In Version 2.0, the command **Module** replaces the command **Block**; although **Block** is still supported under Version 2.0.

The screenshot shows a Mathematica notebook window titled "Version 1.2 LocalVariable". The notebook contains the following code and output:

```

In[5]:=
  Clear[j]
  j=0
Out[5]=
  0
In[2]:=
  value[i_]:=Block[{j},
    j=i;
    Print[j]
  ]
In[6]:=
  value[4]
  4
In[7]:=
  ?j
  j
  j = 0
  
```

Annotations on the right side of the notebook explain the code:

- Clear[j]**
j=0
defines the variable j to be zero.
- On the other hand, j is also defined to be a variable local to the function value.
- value[4] assigns the value 4 to j locally.
- The global value of j remains 0.

Functions can be defined to perform various tasks using *Mathematica* programming skills. This is shown below with the definition of the function **arclength** which calculates the length of the curve given by \mathbf{x} over the interval from $t = \mathbf{a}$ to $t = \mathbf{b}$. This function depends on the local variables **rprime**, **length**, and **integrand** which give the derivative of \mathbf{x} , the number of components of \mathbf{x} , and the integrand given in the integral formula to determine arc length, respectively. A particular function $\mathbf{x}[t]$ is then defined to illustrate the use of **arclength**.

```

In[49]:=
arclength[r_., {t_., a_., b_}] :=
  Block[ {rprime, length, integrand} .

  rprime=D[r, t];

  length=Length[rprime];

  integrand=Sqrt[
    Sum[ (rprime[[i]])^2, {i, 1, length} ]
  ];

  NIntegrate[integrand, {t, a, b}]
]

In[50]:=
r[t_]={t-Sin[t], 1-Cos[t]}

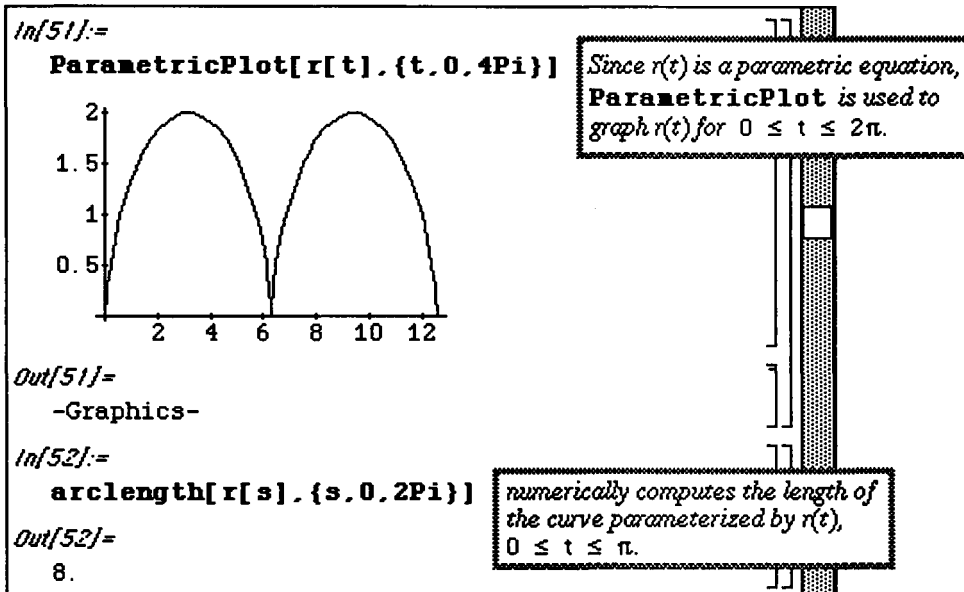
Out[50]=
{t - Sin[t], 1 - Cos[t]}

```

The variables **rprime**, **length**, and **integrand** are declared to be local to the function **arclength**. Notice that a semi-colon is placed at the end of each command (except for the last).

Defines $r(t) = \{t - \text{Sin}(t), 1 - \text{Cos}(t)\}$.

To better understand the use of `arclength`, the function $\mathbf{r}[t]$ is plotted below from $t = 0$ to $t = 4\pi$. Then, the length of this curve is determined with `arclength`. Note that the dependent variable is not of importance in the use of this function as s is used in this command instead of the variable t .



Another point of interest is the manner in which `arclength` is defined. Notice that it is defined in terms of the number of components in the function \mathbf{r} . Hence, `arclength` can be used with functions of more than two components as illustrated below with the function $\mathbf{v}[t]$.

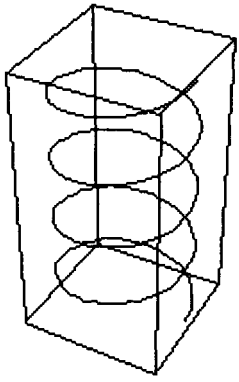
After plotting this curve with **SpaceCurve** (located in the package **ParametricPlot3D.m**), the length of this curve in three-dimensions is found with **arclength**. Note that **Release** must be used with **v[t]** in order for it to be evaluated at various values of **t** in the **SpaceCurve** command.

- o In Version 2.0, the command **Release** is replaced by the command **Evaluate**; **SpaceCurve** is replaced by **ParametricPlot3D**.

```

In[53]:=
  v[t_]= {Cos[t], Sin[t], t}
Out[53]=
  {Cos[t], Sin[t], t}
In[54]:=
  <<ParametricPlot3D.m
In[55]:=
  SpaceCurve[Release[v[t]],
    {t,0,8Pi,8Pi/200},BoxRatios->{1,1,2}]

```



```

Out[55]=
  -Graphics3D-
In[56]:=
  arclength[v[t], {t,0,2Pi}]
Out[56]=
  8.88577

```

Defines the curve $v(t) = \{\cos(t), \sin(t), t\}$.

The vector-valued function v can be graphed with the command **SpaceCurve** which is contained in the package **ParametricPlot3D.m**

The command **SpaceCurve** only graphs three-dimensional vectors. Hence, **Release[v[t]]** must be included so that Mathematica first computes **v[t]**. Otherwise, Mathematica will assume **v[t]** is a one-dimensional vector and not graph v .

Numerically computes the arc length of the curve parameterized by $v(t)$, $0 \leq t \leq 2\pi$.

Mathematica also includes several typical programming techniques. These include the **If** statement and the **Do** loop. Before illustrating these ideas, however, several built-in *Mathematica* commands must be introduced. These include **Divisors[n]** which lists all divisors of the integer **n**, including **n**, and **Drop[list, -1]** which deletes the last element of **list** and returns the resulting list. In the example below, the divisors of 6 are computed with **Divisors[6]**. This list is called **div6**. Next, the last term in **div6** is removed with **Drop[div6, -1]** and the resulting list named **divs**. Finally, the sum of the elements of **divs** is found with **Apply[Plus, divs]**. Similar steps will be used in the example which follows.

<i>In[61]:=</i>	div6=Divisors[6]	div6=Divisors[6]	<i>computes a list of all divisors of 6 and names the list div6.</i>
<i>Out[61]=</i>	{1, 2, 3, 6}		
<i>In[62]:=</i>	divs=Drop[div6, -1]	divs=Drop[div6, -1]	<i>removes the last element from the list div6 and names the resulting list divs.</i>
<i>Out[62]=</i>	{1, 2, 3}		
<i>In[63]:=</i>	Apply[Plus, divs]	Apply[Plus, divs]	<i>computes the sum of the elements of the list divs.</i>
<i>Out[63]=</i>	6		<i>Since the sum of all proper divisors of 6 is 6, 6 is a perfect number.</i>

The calculations previously discussed can be used to find perfect numbers. Recall that a number n is perfect if the sum of its divisors (not including n) equals the number n itself. A function `perfectq[n]` is defined below using the steps illustrated above. Note that there is an `If` statement within this function. Syntax for an `If` statement is `If[condition, then, else]`. In the case of the function below, if the sum of the divisors is n , then a value of `yes` is assumed while a `no` is assumed otherwise. Next, the function `printp[j]` is defined to print a number if it is perfect. Finally, a `Do` loop is used to find all of the perfect numbers between 1 and 10,000. Note that the loop `Do[expression, {i, imin, imax}]` evaluates `expression` from $i = \text{imin}$ to $i = \text{imax}$.

```

In[57]:=
  perfectq[n_]:=
    Module[{divs1,divs2,sumn},
      divs1=Divisors[n];
      divs2=Drop[divs1,-1];
      sumn=Apply[Plus,divs2];
      If[sumn==n,yes,no]
    ]

In[58]:=
  printp[j_]:=
    If[perfectq[j]==yes,Print[j]]

In[60]:=
  Do[printp[i],{i,1,10000}]

6
28
496
8128

```

*In Version 2.0, the command **Module** has replaced the command **Block** (although Version 2.0 supports the command **Block**).*

*The function **perfectq[n]** first computes a list of all divisors of n , then removes the last element from the list (which is n), computes the sum of the list, and if the sum is n , prints "yes"; if not, prints "no".*

*The function **printp[j]** computes **perfectq[j]** and prints j if j is a perfect number and does nothing if j is not a perfect number.*

*Computes **printp[i]** for $i = 1, \dots, 10000$. We conclude that the only perfect numbers between 1 and 10000 are 6, 28, 496, and 8128.*

□ Example:

The command `solidrev` was used to create the solids of revolution found in Chapters 3 and 6. A brief description of `solidrev` is given below. Notice that the arguments of this command include the function `f`, the domain `{a,b}`, the axis about which `f` is to be revolved (either `xaxis` or `yaxis`), and the `solid` option which graphs the resulting solid of revolution.

```

solidrev::usage="solidrev[f, {a,b}, axis]
yields a three-dimensional meshed
image of the function f[x] defined
on the domain [a,b] revolved about
the xaxis or yaxis.solidrev[f, {a,b}, axis, solid]
yields a solid surface.
The interval [a,b] is automatically
divided into 10 subintervals. This
may be changed by substituting {a,b,n} for {a,b}
where n is the desired number of subintervals."

```

```

solidrev[f_, {a_, b_, m_ : Automatic}, axis_, ll_ : Automatic] :=
Block[

  {n, ll, xaxis, yaxis, un, list1, s, t, list2, q, list4, poly},
  xaxis=0;
  yaxis=1;
  uu=axis;

  If[m===Automatic, n=10, n=m];

  list1=Table[{x, f[x]}, {x, a, b, (b-a)/n}] // N;

  list2=If[uu==0,
    s[{{x_, y_}}:=Table[{x, y sincostab[[i, 2]],
      y sincostab[[i, 1]]}, {i, 1, Length[sincostab]}];
    Map[s, list1],

    t[{{x_, y_}}:=Table[{x sincostab[[i, 1]],
      x sincostab[[i, 2]], y}, {i, 1, Length[sincostab]}];
    Map[t, list1]
  ];

  un[k_]:=Partition[k, 2, 1];

  list3=Map[un, list2];

  q[i_, j_]:=Join[list3[[i, j]], Reverse[list3[[i+1, j]]]];

  list4=Flatten[Table[q[i, j], {j, 1, Length[list3[[1]]]},
    {i, 1, Length[list3]-1}], 1];

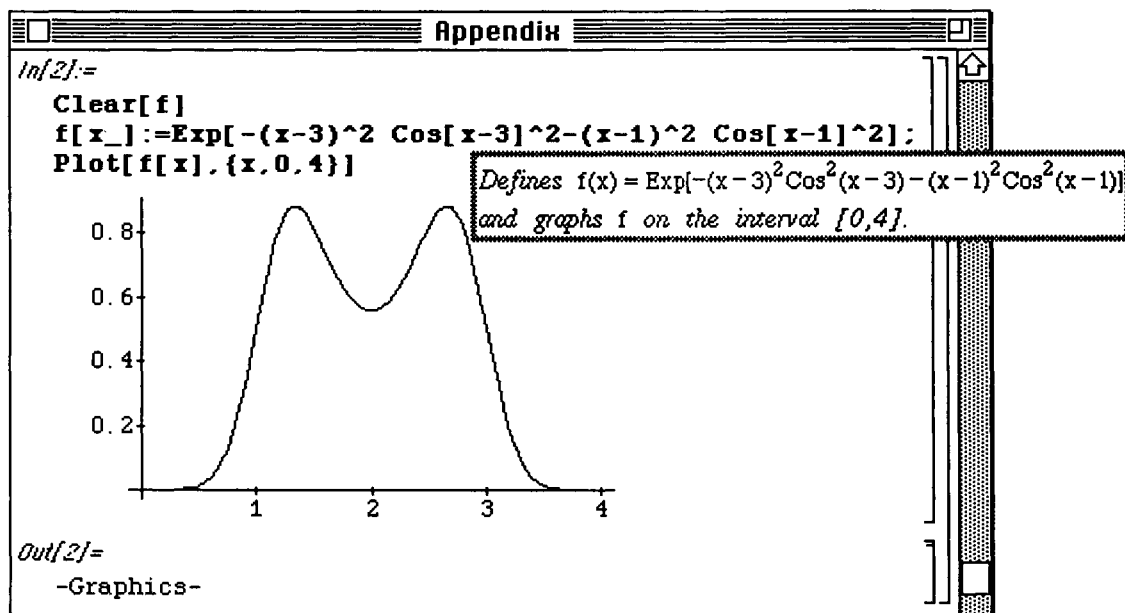
  poly=If[ll===Automatic, Map[Line, list4], Map[Polygon, list4]];

  Show[Graphics3D[poly]]
];

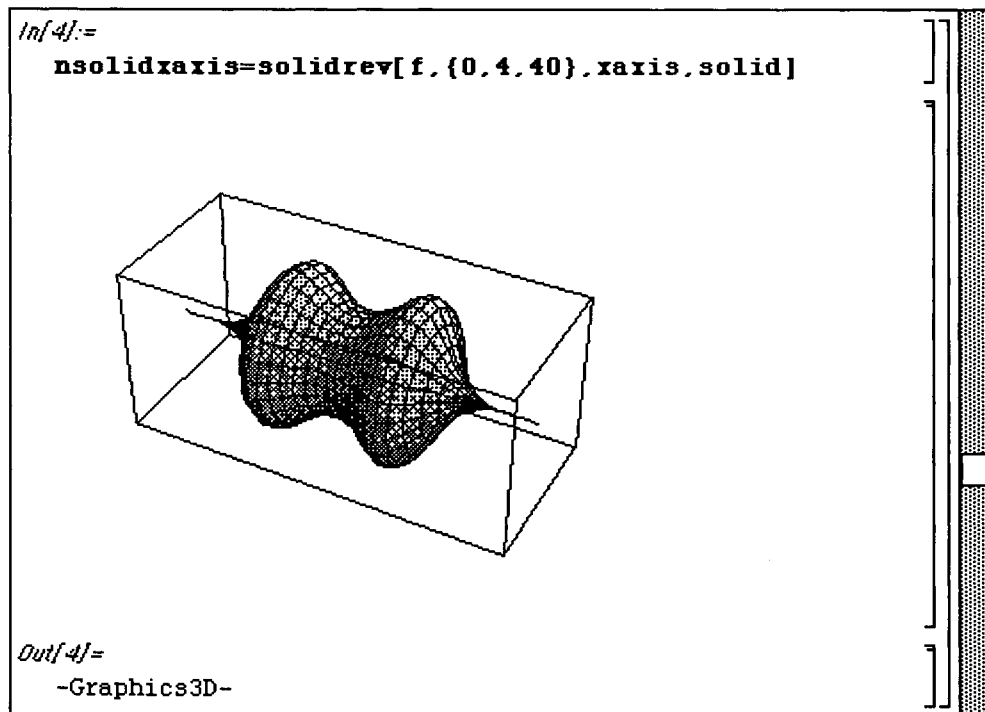
```

□ Example of `solidrev`:

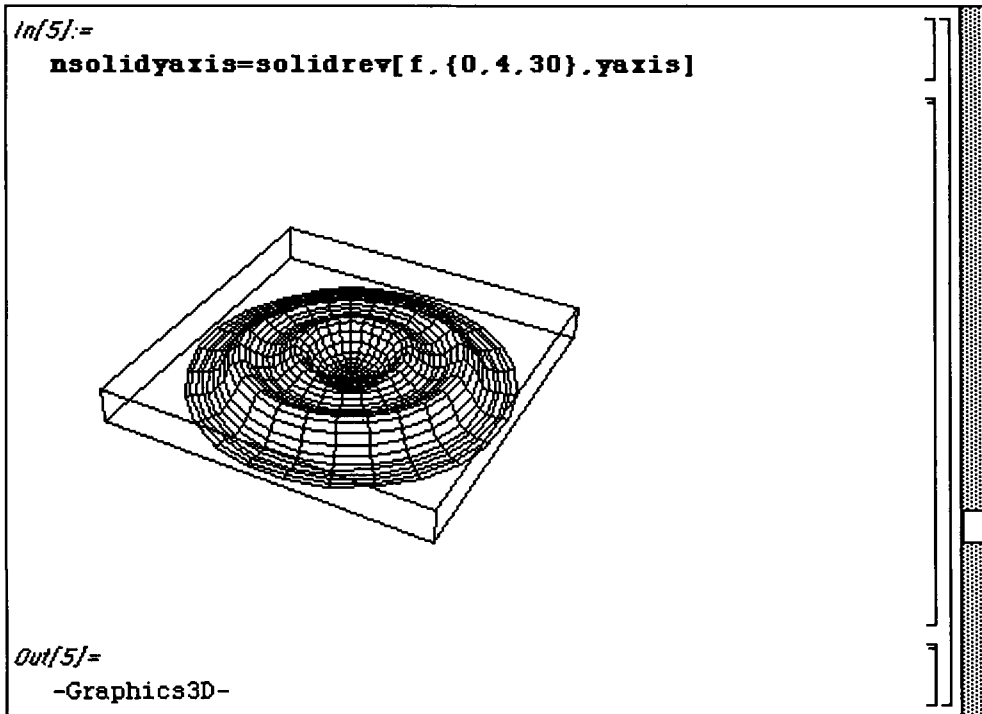
An illustration of the use of `solidrev` is given below. A function f is first defined and plotted.



This function is then revolved about the x-axis with `solidrev`. Note the manner in which the arguments are entered in `solidrev`.



The same function is then revolved about the y-axis.



Some other graphics which were illustrated earlier in *Mathematica By Example* without explanation occurred in the section on Lagrange multipliers. These graphics were produced with the function **lagrangem** below. This function graphs the curves **f** and **g** for values of **x** over the interval **xmin** to **xmax** by evaluating each function at **n** points and joining the points obtained with line segments. Graphs of this type are useful in giving a geometrical impression of where the optimum values of **f** occur subject to the constraint **g**.

□ Example:

```

lagrangem[{f_, g_, cc_:0},
  {xmin_, xmax_, n_:15}, u_:{1, 1, .4}]:=

Block[{values, graphf, graphg, y, ycoords, t1, t2, coords, ff,
  fpoints},

values=Table[N[xmin + i(xmax-xmin)/n], {i, 0, n}];

y[k_] :=NRoots[g[k, y]==0, y];

ycoords=Map[y, values];

t1=Table[{values[[i]], ycoords[[i, 1, 2]], cc}, {i, 1, n+1}];

t2=Table[{values[[i]], ycoords[[i, 2, 2]], cc}, {i, 1, n+1}];

graphg1=Graphics3D[{GrayLevel[.3], Line[t1]}];

graphg2=Graphics3D[{GrayLevel[.3], Line[t2]}];

ff[{a_, b_, c_}] :={a, b, f[a, b]};

fpoints1=Map[ff, t1];

fpoints2=Map[ff, t2];

graphf1=Graphics3D[Line[fpoints1]];

graphf2=Graphics3D[Line[fpoints2]];

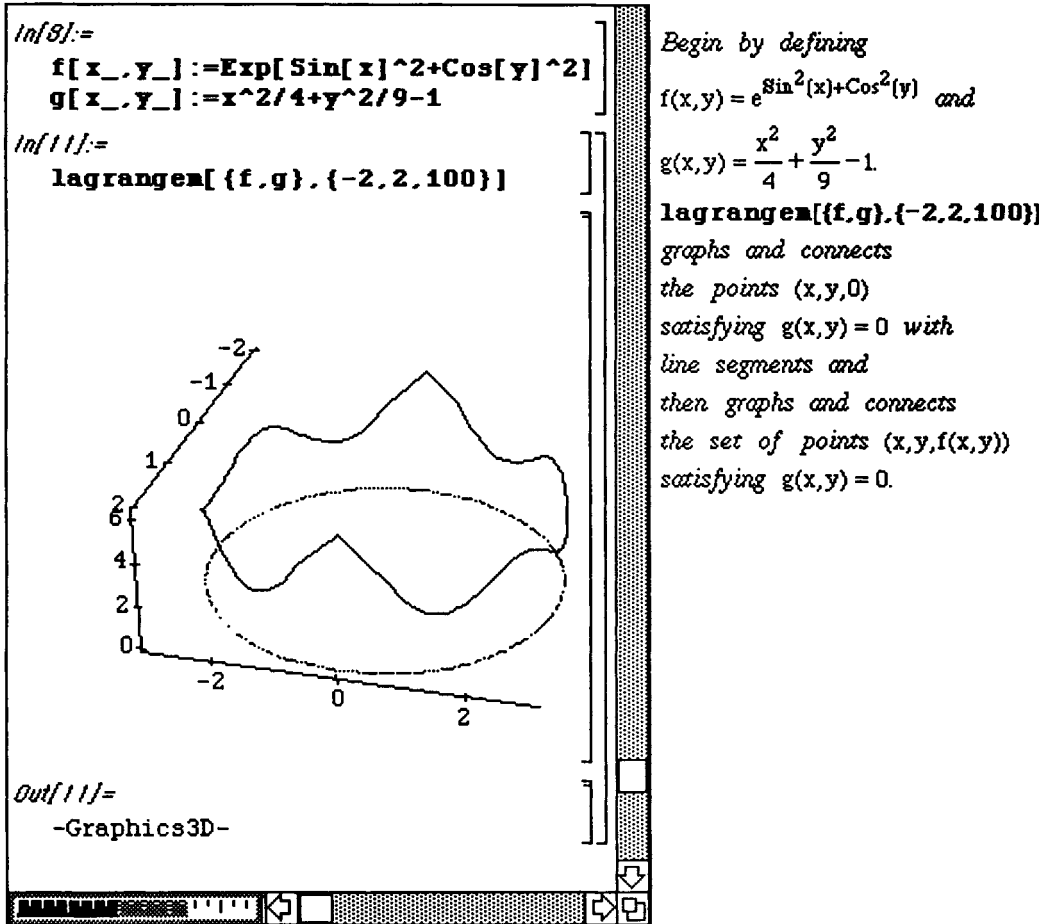
Show[graphf1, graphf2, graphg1, graphg2, Axes->Automatic,

  Boxed->False, BoxRatios->u, ViewPoint->{3.880, 0.950, 2.220}]
]

```


□ Example of lagrangem:

The function **lagrangem** is illustrated below with the two functions, **f** and **g**. This is done over the interval from -2 to 2 using 100 points. Notice where the maximum and minimum values of the function **f** occur.



Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

&& (logical connective “and”), 394
%, 38
/., 27
//, 37
3D ViewPoint Selector, 524
?, 594-599
??, 601
@, 37

A

Abs, 20
absolute value, 20
Action (Menu) (2), 625
Action (Menu), 618
Action Settings, 613-614
Airy’s equation, 379
Algebra folder, 423
amortization
 annual interest paid, 223
 annual principle paid, 223
 current interest paid, 222
 current principle paid, 221
 monthly payments, 219
 total interest paid, 220
 unpaid balance, 221
Analytic (2), 90
Animation Settings, 240,615
annihilator, 356
annuities
 future value, 214
 deferred, 217
 present value, 216
annuity due, 214, 215
Apart, 27,29
AppendColumns, 447
AppendRows, 447
Apply, 204
applying operations to lists, 204, 209
approximation of functions
 with polynomials (2), 559-564
Approximations.m (2), 544-548
arc length, 133,135
ArcCos, 25
ArcCosh, 25
ArcCot, 25
ArcCoth, 25
ArcCsc, 25
ArcSec, 25
ArcSech, 25
ArcSin, 25
ArcSinh, 25
ArcTan, 25

ArcTanh, 25
area between two curves, 130-132
arithmetic operations, 16
Array, 262
AspectRatio, 54,229
Axes (Plot3D option), 309
Axes, 54
AxesLabel, 54
AxesLabel, 63
AxesOrigin (2), 58

B

Background (2), 68
BarChart, 504-506
Bessel functions
 of the first kind, 322, 402
 of the second kind, 402
 zeros of, 407
Bessel’s equation, 377, 378
BesselJ, 8,322,402
BesselY, 402
Binomial, 481-483
Block, 576
Boxed (Plot3D option), 309

C

Calculus folder, 470
calculus
 differential, 92-122
 integral, 123-146
 multi-variable, 147-189
Cancel, 27,29
Cartesian coordinates,303
CartesianMap (2), 539, 540
CatalanNumber,485
Cell (Menu), 616, 617
cell
 active, 6
 changing style, 9
 graphics, 6
 inactive, 6
 input, 4
 output, 6
 text, 6
characteristic polynomial
 of a matrix, 282
Chebyshev polynomials, 461
Cholesky.m, 435-441
CholeskyDecomposition, 435
Chop, 173, 554
Coefficient, 357
CombinatorialFunctions.m, 481-489

Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

CombinatorialSimplification.m, 490-493

combining fractions, 26

complex numbers

real and imaginary parts 427-435

ComplexExpand (2), 43

ComplexMap.m (2), 539-542

ComplexToTrig, 388, 429

Compose (o), 39

Composition (2), 39,43

composition of functions, 39

computing limits, 87

ConfidenceIntervals.m, 586-588

ConfidenceLevel, 586

conic sections

plotting of (2), 530-532

conservative vector field, 303

ConstrainedMax, 288

ConstrainedMin, 288

continuous distribution

mean value of, 571

variance of, 571

ContinuousDistributions.m, 571-578

ContourPlot, 119-124, 147,148,150,152, 153

ContourPlot, options (2)

Contours,121, 122, 152, 153

ContourShading, 120-122, 150-153

ContourSmoothing, 120-122, 153

PlotPoints, 121, 122

coordinate systems (built-in), 304

Cos, 8, 20, 23

Cot, 20

counting distinct elements

of a list, 575-577

creating a list of functions, 231

creating a nested list, 193

creating lists, 192

critical points, 99

classification of, 160-166

locating, 98-103

Cross.m, 442

Csc, 20

Cube, 509

curl of a vector field, 306

Curl, 307

cursor shapes, 10

Cylindrical coordinates, 303

Defaults Window, 621

defining

inequalities, 289

matrices, 262

piecewise functions, 251

series recursively, 257

vector-valued functions, 205

vectors, 266

definition of replacement rules, 431

Denominator, 27

Density, 571, 572

Derivative, 156-159

derivatives

computing, 92-96

higher, 97

numerical (2), 555-557

partial, 154-155

DescriptiveStatistics.m, 571-578

Det, 267

determining the area of a triangle, 442

difference quotient, 88,89

differential equations, see

ordinary differential equations

partial differential equations

differential operator, 356

Dimensions, 452, 453

Direction (2), 91

DiscreteMath folder, 481

DispersionReport, 575

display clock, 280

DisplayFunction, 52,66, 242

displaying multiple graphics, 51,60, 61,62

distance formula, 113,114

distance from a point to a line, 111,112, 444

distance from a point

to a subspace, 457, 458

distribution functions available, 571

Div, 304

Divergence Theorem, 314, 315

divergence of a vector field, 303

Do loop, 258

Dodecahedron, 509

dot product of vectors, 310

DSolve (2), 362, 371, 377-379

DSolve, 336

Dt, 114-116, 350

dual linear programming problem, 290

D

D, 92-96, 154-156

Data Analysis folder, 571-593

DefaultFont (2), 63

Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

E

E, 19,21
Edit (Menu) (2), 622
Edit (Menu), 612-615
edit, 12
Eigenvalues, 270
Eigenvectors, 272
equations
 approximate solutions, 76,81
 exact solutions, 71-74
 graphing, 118
equilibrium points, 385
errors, common , 627-633
Euler's constant, **EulerGamma**, 426
Evaluate (2), 231
evaluating a list of functions, 200
evaluating expressions, 28,29
evaluating functions, 30
evaluation of function at values
 of list, 207, 208
Exp, 20,21
Expand, 6,26
Expand, Trig->True (2), 42, 599
Exponent, 357
exponential function, 20,21
exponents, 17
expressions, graphing, 48
extracting elements of lists, 233-236
extracting elements of matrices, 266

F

Factor, 26,28
Factor, Trig option (2), 599
factoring polynomials, 26,28
falling body problem, 341-344
Fibonacci sequence,
 application of, 488
Fibonacci, 486, 487
file, 11
FindRoot, 76,81, 102-105, 132,134
First, 204
Fit, 241-248
FittingPolynomial (2), 559
Flatten, 400
Floor, 572, 573
flux, 314
FontForm (2), 63
Fourier series, 249-250, 477-480
FourierTransform.m (2), 250, 477-480
FourierTrigSeries (2), 477

Frame (2), 54,58
Framed (o), 54
FrameTicks (2), 59
FromCycles, 495, 496
FullReport, 580
function@list, 207, 208
functions
 composing, 39
 defining, 30
 graphing, 48
 periodic extension of, 479
 plotting complex-valued (2), 539-542
 recursive definition of, 486

G

gamma function, 556
Gamma[x], 556
GaussianQuadrature.m (2), 549-552
GaussianQuadratureWeights (2), 549
genus of a surface, 522
GoldenRatio, 485
GosperSum (o), 424
GosperSum.m (o), 423
Grad, 304
gradient of a scalar function, 303
GramSchmidt, 455-459
graph theory application, 448-450
Graphics, 61,62
Graphics folder, 497-542
graphics, displaying multiple, 60,61,62
Graphics.m, 497-508
GraphicsArray, 60,66, 169,416, 420-421
graphing
 equations, 118
 functions , 48
 functions and derivatives, 95, 113
 implicit functions, 117-122, 529-531
 options, 68,54
 options, examples, 55
 parametric equations, 228-230
 multiple, 51
 piecewise defined
 functions, 67,68
 Version 2.0, 57
graphs
 locating intersection points, 79
Green's Theorem, 312
GridLines (2), 57
Group Cells, 616, 617

Index

- (o) implies obsolete in Version 2.0;
- (2) implies applicable only to Version 2.0.

H

Helix, 520, 521
Help (2), 606-608
Help file, 604, 605
help
 commands, 594-608
 completing a command, 602, 603

HermiteH, 194

higher order derivatives, 96,97

HilbertMatrix, 451

Hofstadter function, 488

Hofstadter, 488, 489

HypothesisTests.m, 580-585

I

I, 19,21

Icosahedron, 509

IdentityMatrix, 286

Im, 427

implicit differentiation, 114-118

ImplicitPlot (2), 117, 118-120, 235, 529-531

ImplicitPlot.m (2), 529-532

Infinity, 87,92

inflection points, locating, 98-103

init.m file, 609-611

initializing functions, 610, 611

inner product of space

 of continuous functions, 458

InnerProduct

 (**GramSchmidt** Option), 458,459, 461, 462

InputForm (2), 225

integral calculus, 123-136

integral tables, loading, 123

integrals

 approximation, 129

 definite (2), 129

 definite, 125-129

 indefinite, 124, 125

 multiple, 176-184

 numerical approximation by

Gaussian Quadrature (2), 549-552

IntegralTables.m, 125

Integrate (double integrals), 176

Integrate, 123-128

interest compounded daily, 210

InterpolatingFunction, 414, 415

 plotting of, 415

interpolation with a rational

 function (2), 544-548

Interrupt, 618

intersection points of graphs, 79

inverse trigonometric functions, 24

Inverse, 269

InverseLaplaceTransform (2), 471

investments, 224-227

J

Jacobian matrix, 390

K

Kernel Help (2), 606-608

L

Lagrange multipliers

 (optimization problems), 170-176

LaguerreL, 202

LaplaceTransform (2), 471

LaplaceTransform.m (2), 471-476

laplacian of a scalar field, 303

Laplacian, 304

Last, 204

Legendre polynomials, 459, 460

Legendre's equation, 377, 378

length of vector, 311, 317-318

Length, 204

Limit (2), 90, 91

Limit, 86-89

limits

 computation of, 86-91

 numerical approximation of (2), 553-558

 one-sided, 91,92

Linear Algebra folder, 435

linear programming, 288-302

LinearProgramming, 295

LinearRegression.m, 589-593

LinearSolve, 279,451

list, 192

list//function, 207, 208

ListPlot, 198,202

Log, 20,21,22

Logical Expand, 145,149, 394

Long Menu (2), 622

M

Map, 200, 201

matrix

 multiplication,

 powers of matrices, 274, 275

 operations, 268

 adjacency, 448

 complex conjugate transpose, 435

Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

condition number of, 452, 453
Hermitian, 435
Hilbert, 451
norm of, 452, 453
positive definite, 435
symmetric, 435
well-conditioned, 452, 453
MatrixForm, 263
MatrixManipulation.m, 447-454
MatrixPower, 275, 276, 450
maxima and minima, 109-114
maximization/minimization (calculus)
 problems, 107-110
Mean, 571, 572
MeanCI, 586
MeanDeviation, 574
MeanDifferenceTest, 582
MeanDifferenceTest, 582
MeanTest, 580
Median, 574
MedianDeviation, 574
Menu (2), 622-626
Menu, 612-626
menu, 11
minimal polynomial of a matrix, 282
MiniMaxApproximation (2), 547
Mod, 37, 44, 207
Module (2), 576
MoebiusStrip, 513, 518, 519
multi-variable calculus, 147-189
Multinomial, 481-483

N

N, 6, 18, 22
naming graphs, 50
naming objects, 28
natural logarithm, 20
ND (2), 555-557
NDSolve (2), 396-397, 414-415, 417-419
Nest, 40
NFourierTrigSeries (2), 477, 479
NIntegrate (double integrals), 176
NIntegrate, 127, 129
NLimit.m (2), 553-558
Normal, 140, 141, 186, 395
NormalDistribution, 572
Normalized
 (**GramSchmidt** Option), 459
notebook, 4
NRoots, 76, 100, 101, 102, 130
NSolve (2), 78, 103
NSum, 426

Numerator, 27
Numerical Math folder (2), 543
numerical
 differentiation (2), 555-557
 integration, 129
 limits (2), 553-555
 solutions of differential
 equations (2), 396-397, 414-415, 417-419

O

objective function, 298
Octahedron, 510
operation@@list, 209
Options, 600
ordinary differential equations
 Cauchy-Euler, 369-371
 characteristic equation, 363
 equilibrium points of, 385, 389-390
 exact, 345-354
 finite element method
 of solution of, 465-469
 first-order linear (**DSolve**), 336-344
 initial value problem, 366
 initial value system (**DSolve**), 380-381
 linear n-th order
 homogeneous, 363
 linearization of
 nonlinear systems, 385, 389-391
 numerical solution of (2), 414-416
 numerical solution of systems
 of (2), 417-421
 power series solutions, 143-146
 Runge-Kutta approximate
 solution to, 565-570
 series solutions, 392-401
 solution by Laplace
 transform (2), 473-476
 systems of linear
 (**DSolve**), 380-385
 undetermined coefficients, 355-362
 variation of parameters, 372-376
Orthogonalization.m, 455-463
Out[n], 38
output, Version 2.0, 41

P

p-value, 580
Packages folder, 422
ParametricPlot, 229, 476
ParametricPlot3D, 413, 523
ParametricPlot3D.m, 523-527

Index

- (o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

- partial differential equations
 series solutions to, 402-413
partial fraction decomposition, 27
pendulum equation, nonlinear, 414-416
PermutationQ, 494
Permutations.m, 494-496
Pi, 8,19,21
piecewise defined functions, 67,68
PieChart, 507-508
Plot, 8,48
Plot options, 54
 PlotRange, 141
 PlotRange, 545
 PlotStyle, 48
 RGBColor, 48
Plot3D, 8,147, 149,151
Plot3D options
 Axes, 166
 Boxed, 165
 BoxRatios, 165
 DisplayFunction, 169
 Mesh, 165
 PlotLabel, 165
 PlotPoints, 149
 Shading, 149
 Ticks, 165
PlotField.m (2), 533-534
PlotField3D.m (2), 535-538
PlotGradientField (2), 534
PlotGradientField3D (2), 535, 537
PlotJoined (**ListPlot** option), 202
PlotLabel, 54,59, 63
PlotODESolution, 569
PlotPoints (**Plot3D** option), 309,153
PlotRange, 54,229
PlotStyle, 8,53
plotting a list of functions, 199,231
plotting tangent line to curve, 237-239
PlotVectorField (2), 533
PlotVectorField3D (2), 536
PolarListPlot, 501-503
PolarMap (2), 540-542
PolarPlot, 497-500
PolyGamma, 556
Polyhedra.m, 509-512
Polyhedron, 509
polynomial approximation
 with **Legendre** polynomials, 462, 463
PolynomialFit (2), 559
PolynomialFit.m (2), 559-564
PolynomialMod (2), 46
power series, 137-146
power series of a function of more
 than one variable, 185-189
power series, remainder term of, 141-143
Preferences (2), 623, 624
Prime, 45
prime notation, 94
Prime, 197
Print (2), 225
Projection, 457
- ## Q
- quadratic equations
 (quadratic form), 440, 441
Quartiles, 574
- ## R
- Random**, 573
RandomPermutation, 495, 496
RationalInterpolation (2), 544
Re, 427
Rectangle (2), 61,62
references, 634
Regress, 589, 593
ReIm.m, 427-435
Release, 8,199, 231
resizing graphics cell, 505
retrieving unnamed output, 38
RotateShape, 518, 519, 521
RungeKutta, 565
RungeKutta.m, 565-570
- ## S
- saving output files, 332-334
scalar function, 303
Sec, 20
second-derivative
 test for extrema, 160
Series, 137,139, 185,393
series, truncation of, 395
SetCoordinates[System], 303
Settings
 Action, 613, 614
 Animation, 615
 Startup, 612, 613
Shading (**Plot3D** option), 153,309
Shapes.m, 513-522
Short, 573
Short Menu (2), 622
Short, 200
Show, 51,61,62
Simplify, 32,88,90
simultaneous plots with **Show**, 243, 245, 248

Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

Sin, 8,20, 23
SingularValues, 453, 454
solidrev, 413
solids of revolution, 134-136
Solve, 71-75,80
solving equations, 71,76
solving systems of
 linear equations, 277-282
solving systems with
 LinearSolve[A,b], 279, 282
solving systems with **Solve**, 277, 278, 281
Sort, 573
SpaceCurve (o), 524-527
Sphere, 515-517
Spherical coordinates, 303
spring motion, 380-385
Sqrt, 18
Stack Windows, 619
Startup Settings, 612, 613
Statistics folder (2), 579
statistics
 confidence intervals, 586-588
 ConfidenceLevel, 586
 Density, 572
 DispersionReport, 575
 hypothesis tests, 580-585
 KnownStandardDeviation, 582, 587
 mean, 571-575
 Mean, 572
 MeanCI, 586
 MeanDifferenceTest, 582
 MeanTest, 580
 p-value, 580
 Quartiles, 574
 variance, 571-575
 Variance, 572
 VarianceCI, 588
 VarianceTest, 585
Stellate, 511, 512
Stoke's Theorem, 316-321
Style (Menu) (2), 626
Style (Menu), 9,12, 618
Styles Window, 621
Subfactorial, 484
sums,
 closed form expressions of, 424-426
SymbolicSum (2), 425
SymbolicSum.m (2), 224, 225, 423

T

Table, 8,45, 192
TableForm, 45,230

Tan, 20,23
tangent lines, 104-106
 graphing, 106,107, 108
 horizontal, 101
tangent planes, 167-169
Taylor polynomials, 142
Taylor remainder, 144
test equality, 331, 347
Tetrahedron, 510
Ticks, 54,228
Tile Windows Tall, 620
Tile Windows Wide, 620
ToCycles, 495, 496
Together, 26
Torus, 514, 522
total differential, 350
trace of a matrix, 282
TranslateShape, 518, 519, 522
transportation problem, 297-302
Transpose, 269,436, 437,439, 441
Tridiagonal.m, 464-469
TridiagonalSolve, 464
TrigCanonical (o), 599
TrigExpand (o), 320, 360
trigonometric functions, 20,23
Trigonometry.m, 387,429
TrigReduce, 432

U

unit normal vector, 305, 317
user-defined functions, 30

V

Van der Pol's equation , 417-421
Variables, 357
Variance, 571, 572
VarianceCI, 588
VarianceTest, 585
vector calculus, 303-321
vector-valued functions, 34, 35
VectorAnalysis.m, 303
vectors
 cross product of, 442
 Gram-Schmidt orthogonalization
 of basis, 455-459
 length of, 443, 457
 projection onto subspace, 456
volume, 136
volume,
 computation with multiple integral, 180-184
volumes of solids of revolution, 134-136

Index

(o) implies obsolete in Version 2.0;
(2) implies applicable only to Version 2.0.

W

wave equation

(with initial and boundary conditions), 403-404

wave equation, solution to, 408-413

Window (Menu) (2), 624

Window (Menu), 619, 620

window, 12

WireFrame, 516

Wronskian matrix, 372