1

Introduction to *Mathematica*

Mathematica is a computer algebra system that performs numeric, symbolic, and graphical computations. As characterized by the developers, Wolfram Research, Inc. *Mathematica* is "a system for doing mathematics by computer." *Mathematica* is different from other programming languages that are used by economists (FORTRAN, BASIC, Pascal, C, etc.). It is an interpreted language, i.e., each input statement produces immediate output. Although *Mathematica* can be used as a programming language, its high level structure is more appropriate for performing sophisticated operations through the use of built-in functions. For example, *Mathematica* can find limits, derivatives, integrals, and determinants, as well as plot the graphs of functions and perform symbolic computations. The number of built-in functions in *Mathematica* is enormous. Our goals in this introductory chapter are modest. Namely, we introduce a small subset of *Mathematica* commands necessary to explore the mathematics discussed in this book.

Mathematica is composed of two parts: the *Mathematica* front end and the *Mathematica* kernel. The *front end* accepts input, displays output, and generally organizes information in a *Mathematica* session. The *kernel* is the part of the program that does the actual calculations. The front end and kernel communicate through a communications protocol called **MathLink**. It is possible to run the front end and kernel on different computers, although most people run them on the same machine. There are three basic types of front ends: Microsoft Windows, Macintosh, and UNIX. These front ends use *Mathematica* notebooks as the interface between the user and the kernel. *Mathematica* notebooks are analogous to electronic worksheets that allow the integration of *Mathematica* input statements, kernel output, and text. Except for minor differences, *Mathematica* notebooks for each front end appear quite similar. We will not discuss *Mathematica* notebooks in this text, but it will be necessary to familarize yourself with the notebook features on your machine before you try to use *Mathematica*.

□ 1.1 Doing Arithmetic with *Mathematica* □

At its most primitive level, *Mathematica* can be viewed as a calculator. It can perform the five basic operations of arithmetic: addition (+), subtraction (−), multiplication (*, or a blank space), division (/), and exponentiation (^). Later we will discuss how to

perform these arithmetic computations. Unlike a basic calculator, however, *Mathematica* does rational arithmetic and keeps a history of the operations that are performed by the kernel.

1.1.1 The **In** and **Out** tags

Mathematica keeps track of each input expression sent to the kernel and each output statement that it produces in a session. This bookkeeping of statements is displayed in the front end by the use of **In** and **Out** tags. To add 5 to 3, we type 5+3, then press the <enter> key on the Macintosh front end, or the <shift> and <enter> keys on the Microsoft Windows front end.

In[1]:=
```
5+3
```
Out[1]=
```
8
```

Notice that in addition to the input to the kernel, 5+3, and the output, 8, from the kernel, the front end has attached the tags, **In[1]:=** and **Out[1]=**. This is the sequencing scheme to keep track of inputs and outputs to and from the kernel.

To multiply 13 by 203.8, we type 13*203.8 or 13 203.8 (a space in an expression is interpreted as multiplication) and press the <enter> key.

In[2]:=
```
13*203.8
```
Out[2]=
```
2649.4
```

Since this was the second input to the kernel (in this session), it has been tagged as **In[2]:=** and its output as **Out[2]=**. The *Mathematica* kernel keeps track of all input and output statements in a *Mathematica* session (a session is defined as the interval between launching and quitting a *Mathematica* kernel).

It is possible to refer to an input statement as **In[k]** or an output statement as **Out[k]** where **k** is an integer that refers to the *k*th statement in the *Mathematica* session. **In** and **Out** are, in fact, *Mathematica* functions that can be used in input expressions. For example, we can ask *Mathematica* to execute the second input statement again.

In[3]:=
```
In[2]
```
Out[3]=
```
2649.4
```

Note that the second input instructed *Mathematica* to multiply 13 by 203.8. Next, we add the previous two outputs.

In[4]:=
```
Out[1] + Out[2]
```
Out[4]=
```
2657.4
```

This result is the same as 8 + 2649.4. You can refer to the *k*th previous output and input statements as **Out[−k]** and **In[−k]**, respectively. A short form of **Out[−1]** is **%**, and **Out[−2]** is **%%**. Also, **%k** is a short form for **Out[k]**. For example, here is a sequence of *Mathematica* statements:

In[5]:=
```
3-4
```

Out[5]=
```
-1
```

Multiply 5 to the last result, i.e., 5*(−1).

In[6]:=
```
5*%
```

Out[6]=
```
-5
```

Multiply 6 to the last result, i.e., 6*(−5).

In[7]:=
```
6*%
```

Out[7]=
```
-30
```

Give the third previous result, i.e., −1.

In[8]:=
```
In[-3]
```

Out[8]=
```
-1
```

Generally speaking, blank spaces are ignored by *Mathematica* unless they can be interpreted as multiplication. For example, **2+3** is treated the same as **2 + 3**.

It is important to remember that the sequence of statements inputed to and outputed from the kernel in a *Mathematica* session, is not necessarily the literal sequence of input and output statements that appear in the *Mathematica* notebook. *Mathematica* commands do not have to be executed in a top to bottom fashion in a *Mathematica* notebook.

1.1.2 Numbers, symbols, and assignment

Mathematica does arithmetic differently than ordinary calculators. It does its calculations using rational arithmetic. This allows certain types of calculations to be performed with infinite precision. For example, the computation of an expression such as (4+1)/3 on a calculator would produce the decimal approximation 1.666666666 and not the exact answer 5/3. *Mathematica* treats 5/3 as a symbol in the rational number system and does not approximate it by its decimal expansion. Another example of *Mathematica*'s representation of numbers as symbols is computing the area of a circle of radius 3.

In[9]:=
```
Pi*3^2
```

Out[9]=
9Pi

Here **Pi** is the symbol for irrational number π. There are other special symbols in *Mathematica*. Here is a table of some of the important ones.

Pi	$\pi \approx 3.14159$
E	$e \approx 2.71828$
I	$i = \sqrt{-1}$
Infinity	∞
-Infinity	$-\infty$

Notice that the *Mathematica* built-in constants always begin with a capital letter (or **$**). *Mathematica* is case sensitive, e.g., **Pi** is different from **pi**, and **E** is different from **e**. Since all the built-in constants and functions in *Mathematica* begin with a capital letter, it is a good practice to start with a lowercase letter for user-defined constants or functions.

To assign *Mathematica* expressions to user-defined symbols, we use the **Set** operator (**=**), or the **SetDelayed** operator (**:=**). These two operators differ in the way the assignment is made. We will discuss the differences between the two operations once we learn how to use the **Set** function.

Let us assign the symbol *u* to –3 and the symbol *v* to (1–9)π.

In[10]:=
```
u = -3
```

Out[10]=
–3

In[11]:=
```
v = (1-9)*Pi
```

Out[11]=
–8Pi

Parentheses in *Mathematica* input and output expressions are used as algebraic deliminators. We can now use *u* and *v* in other *Mathematica* expressions. For example, let us calculate $3u-5v$ and assign its value to the symbol *w*.

In[12]:=
```
w = 3*u - 5*v
```

Out[12]=
–9 + 40Pi

One can always ask *Mathematica* what expression has been assigned to a symbol by entering the symbol. Let us inquire about the assignment to *v*.

In[13]:=
```
v
```

Out[13]=
–8Pi

To remove

In[14]:=
```
Clea
```

Now we a

In[15]:=
```
v
```

Out[15]=
v

When *Ma*
to this syn
to *v*. An a
ator (=.). I
following

In[16]:=
```
v =
```

In any of
operation
We nov
When we
evaluates
expr. In t
until the s
that show
value –3.

In[17]:=
```
b
```

Out[17]=
–3

Let us as
both sym

In[18]:=
```
imm
```

In[13]:=
```
v
```

Out[13]=
```
-8Pi
```

To remove any assignments to a symbol, we use the **Clear** function.

In[14]:=
```
Clear[v]
```

Now we ask what expression is presently assigned to the symbol *v*.

In[15]:=
```
v
```

Out[15]=
```
v
```

When *Mathematica* returns the name of the symbol, it indicates that any assignment to this symbol cannot be evaluated, or in this case, no expression has been assigned to *v*. An alternative method to clear a symbol assignment is to use the **Unset** operator (=.). For example, to remove any assignments to the symbol *v*, we could use the following expression:

In[16]:=
```
v =.
```

In any of the above assignment of symbols, we could have used the **SetDelayed** operation (:=) instead of the **Set** operation (=).

We now demonstrate the differences between the **Set** and **SetDelayed** functions. When we enter a *Mathematica* statement of the form *symb* = *expr*, *Mathematica* evaluates the right-side expression (*expr*) and assigns the symbol *symb* the value of *expr*. In the statement *symb* := *expr*, the evaluation of the right-side *expr* is delayed until the symbol *symb* is used in another *Mathematica* statement. Here is an example that shows the difference. Let *b* denote a constant which we initially assign the value –3.

In[17]:=
```
b = -3
```

Out[17]=
```
-3
```

Let us assign the expression 1+*b* to two new symbols: *immediate* and *delayed*. Since both symbols are user-defined variables, lowercase letters are used. The statement

In[18]:=
```
immediate = 1 + b
```

Out[18]=
```
-2
```

uses the **Set** operator and as expected, 1+b was immediately evaluated and assigned to the symbol *immediate*. However, the statement

In[19]:=
```
delayed:= 1 + b
```

uses the **SetDelayed** operator and produces no output. The right-side expression of the **SetDelayed** statement, 1+*b*, is not evaluated until the symbol *delayed* is actually used. Now we inquire about what is stored in the symbol *delayed*.

In[20]:=
```
delayed
```

Out[20]=
```
-2
```

This is expected since the calculation 1+(−3) was performed in this last input statement. Suppose we change the value of *b*, say, to 5.

In[21]:=
```
b = 5
```

Out[21]=
```
5
```

What is assigned to the symbols **immediate** and **delayed**?

In[22]:=
```
immediate
```

Out[22]=
```
-2
```

In[23]:=
```
delayed
```

Out[23]=
```
6
```

Notice the differences: immediate was assigned a value of −2; on the other hand, delayed, 1+*b*, was recalculated in the last statement with $b = 5$ and now has the value of 6.

□ 1.2 Functions □

Mathematica has an impressive list of built-in mathematical functions. It also allows users to define their own functions.

1.2.1 Built-in *Mathematica* functions

Mathematica built-in functions have names that are similar to the standard mathematical notations. All built-in functions in *Mathematica* start with a capital letter and the arguments are enclosed in brackets. For example, the sine function, sin x, is represented as **Sin[x]** in *Mathematica*. Here is a table of commonly used functions.

Abs[x]	absolute value, $\|x\|$
Sqrt[x]	square root, $\sqrt{x}$
Exp[x]	exponential, e^x
Log[x]	natural logarithm, ln x
Log[a,x]	logarithm to base a, $\log_a x$
Sin[x],Cos[x],Tan[x], . . .	trigonometric functions with arguments in radians
ArcSin[x], ArcCos[x], . . .	inverse trigonometric functions
Factorial[n] (n!)	factorial function
Random[ ]	uniform pseudorandom number between 0 and 1
N[x]	numerical value of x
N[x,n]	value of x with n-digit precision
Re[z]	real part of the complex number z
Im[z]	imaginary part of the complex number z
Conjugate[z]	conjugate complex number of z

Mathematica has a built-in help feature that allows us to obtain information about any function. To obtain information about a function, we enter the expression **?** *functionname*. For example, if we want to find information about the function **Log**, we would input the following expression.

In[24]:=
```
?Log
```

Log[z] gives the natural logarithm of z (logarithm to base E). Log[b, z] gives the logarithm to base b.

As we can see, *Mathematica* describes two forms for the logarithm function, **Log[z]** for the natural logarithm, log z, and the **Log[b,z]** for logarithm to base b, $\log_b z$.

Functions may have other functions as their arguments. For example, to enter the expression sinh(log(sin $\pi/4$)), we would use the following *Mathematica* statement.

In[25]:=
```
Sinh[Log[Sin[Pi/4]]]
```

Out[25]=
$$\frac{-1}{2\text{Sqrt}[2]}$$

Mathematica often is aware of certain fundamental properties of a mathematical function. For example, it knows that the exponential function and the natural logarithm function are inverses.

In[26]:=
```
Clear[x];
Exp[Log[x]]
```

Out[27]=

x

Mathematica possesses a built-in function that will compute the decimal representation of an expression. The numerical function **N** will compute the numerical value of an expression.

In[28]:=

```
?N
```

N[expr] gives the numerical value of expr. N[expr, n] does computations to n-digit precision.

For example, we can find a numerical approximation to π.

In[29]:=

```
N[Pi]
```

Out[29]=

3.14159

Alternatively, **N** with two arguments computes the numerical value of the first argument to the precision specified by the second argument.

In[30]:=

```
N[Pi,25]
```

Out[30]=

3.14159265358979323846264338

Mathematica functions may accept complex numbers as arguments and the values of these functions may be complex numbers (even with arguments that are real numbers).

In[31]:=

```
Sqrt[-2]
```

Out[31]=

I Sqrt[2]

In[32]:=

```
Sqrt[-4*I]
```

Out[32]=

$-2(-1)^{3/4}$

To input a complex number we use the symbol **I**, which denotes the imaginary number $\sqrt{-1}$. Suppose that the symbol *a* is assigned the expression 4+3i and the symbol *b* the expression 2−i. *Mathematica* can then perform various arithmetic operations on these complex numbers.

In[33]:=

```
a = 4+3*I
```

Out[33]=
```
4 + 3I
```

In[34]:=
```
b = 2-I
```

Out[34]=
```
2 - I
```

In[35]:=
```
a + b
```

Out[35]=
```
6 + 2I
```

In[36]:=
```
a/b
```

Out[36]=
```
1 + 2I
```

We can also find a numerical approximation to a complex expression.

In[37]:=
```
N[Exp[a]]
```

Out[37]=
```
-54.0518 + 7.70489I
```

The functions, **Re**, **Im**, and **Conjugate** compute the real part, imaginary part, and conjugate of a complex number, respectively.

In[38]:=
```
Re[a]
```

Out[38]=
```
4
```

In[39]:=
```
Im[b]
```

Out[39]=
```
-1
```

In[40]:=
```
Conjugate[a]
```

Out[40]=
```
4 - 3I
```

An alternate way to apply a function to an expression is to use the operator (//). The statement, *expr* // *fct*, is interpreted as *fct*[*expr*]. For example, the following statement is identical to **Sin[Pi/4]**:

In[41]:=
```
Pi/4 // Sin
```

Out[41]=
$$\frac{1}{\text{Sqrt[2]}}$$

The following is the same as **N[Pi]**:

In[42]:=
```
Pi // N
```

Out[42]=
3.14159

1.2.2 User-defined functions

In addition to built-in functions, *Mathematica* allows users to define their own functions. To define a function, we must specify the symbol that will represent the name of the function (in *Mathematica* lingo, the **Head** of the function) as well as the rules that prescribe its action on its arguments. The format of defining a function of one variable is:

symb0[*symb1*_] := definition in terms of *symb1*.

symb0 represents the **Head** of the function, i.e., the name of the function. *symb1* is the independent variable, or argument, of the function. The expression *symb1*_ (with an underscore at the end) denotes a pattern. *symb1* may be a number, a symbol, or another function. For example, to define the function $f(x) = x^2-3x+4$, we would use the following statements:

In[43]:=
```
Clear[f,x];
f[x_]:= x^2 - 3*x + 4
```

We could use the **Set** operator (=) or the **SetDelayed** operator (:=) in the defining statement. As a general rule, it is better to use **SetDelayed**. The symbol **x_** is a pattern. This means that the argument of the function can be anything, not just x. For example, consider the following:

In[45]:=
```
f[4]
```

Out[45]=
8

In[46]:=
```
f[x^2]
```

Out[46]=
$4 - 3x^2 + x^4$

In[47]:=
```
f[z]
```

Out[47]=
$4 - 3z + z^2$

Suppose we did not use the underscore (_) in the definition.

In[48]:=
```
Clear[g];
g[x]:= x^2 - 3*x + 4
```

When we ask *Mathematica* to evaluate this function when its argument is 4, we find the following:

In[50]:=
```
g[4]
```

Out[50]=
g[4]

Mathematica returns the input as output since it does not know how to evaluate **g[4]**. The only thing that has been defined with regards to **g** is the symbol **g[x]**.

Functions of more than one variable are defined in a similar manner. For example, to define a Cobb-Douglas production function with labor (L) and capital (K) as inputs,

$$f(L,K) = 10L^{0.6}K^{0.3},$$

we would use the following *Mathematica* definition:

In[51]:=
```
Clear[f,L,K];
f[L_,K_]:= 10*L^0.6*K^0.3;
```

The function is then evaluated at different values of L and K.

In[53]:=
```
f[5,8]
```

Out[53]=
49.0127

In[54]:=
```
f[6,3]
```

Out[54]=
40.7406

When we define a function in *Mathematica*, it is sometimes necessary to include some conditions on the domain for the function. For example, suppose we define the function

$$f(x) = \begin{cases} x^2, & x < 0 \\ 2x, & x > 0 \end{cases}$$

To define this function in *Mathematica*, we use the **Condition** function (**/;**). Here is the description of this function.

In[55]:=
```
?Condition
```

patt /; test is a pattern which matches only if the evaluation of test yields True. lhs :> rhs /; test represents a rule which applies only if the evaluation of test yields True. lhs := rhs /; test is a definition to be used only if test yields True.

As we can see from the description, the format for using this operator in a definition is *patt* **/;** *test*. *patt* is a pattern and the expression test must be in the form of a logical expression such as $x > 0$ or $x == 0$. These logical expressions are evaluated by *Mathematica* as **True** or **False**, or unevaluated if the expression cannot be determined to be either **True** or **False**. For example, consider the following logical expressions:

In[56]:=
```
-3 < 4
```

Out[56]=
True

In[57]:=
```
3 == 5
```

Out[57]=
False

In[58]:=
```
Clear[y];
y < 3
```

Out[59]=
y < 3

In the last example, *Mathematica* could not evaluate this expression as either **True** or **False** and it thus returned the input statement unevaluated. We use the condition operator to define $f(x)$ given above.

In[60]:=
```
Clear[f,x];
f[x_ /; x < 0]:= x^2
f[x_ /; x > 0]:= 2*x
```

We now inquire about the values of $f(x)$ for different values of x.

In[63]:=
```
f[-1]
```

Out[63]=
```
1
```

In[64]:=
```
f[2]
```

Out[64]=
```
4
```

In[65]:=
```
f[0]
```

Out[65]=
```
f[0]
```

One can always inquire about the definition of a symbol. Suppose we ask *Mathematica* what definitions it has attached to the **Head** *f*. This is accomplished by using **?**.

In[66]:=
```
?f
```

```
Global`f

f[x_ /; x < 0] := x^2

f[x_ /; x > 0] := 2*x
```

Global`f refers to the context of the symbol *f* and is something that we generally can ignore.

1.2.3 Algebraic manipulations

Mathematica has several built-in functions to manipulate expressions. These functions will perform the algebraic tasks of expanding products, factoring, finding common denominators, and doing partial fraction expansions. The most important manipulation functions are **Expand**, **Factor**, **Together**, **Apart**, and **Simplify**. Below a description of these functions is given along with an example.

- **Expand**

In[67]:=
```
? Expand
```

```
Expand[expr] expands out products and positive integer powers in expr.
  Expand[expr, patt] avoids expanding elements of expr which do not
  contain terms matching the pattern patt.
```

In[68]:=
```
Clear[x]
Expand[(x-2)*(x+1)^2]
```

Out[69]=

$$-2 - 3x + x^3$$

- **Factor**

In[70]:=

```
?Factor
```

Factor[poly] factors a polynomial over the integers. Factor[poly, Modulus→p] factors a polynomial modulo a prime p.

In[71]:=

```
Clear[x]
Factor[x^3 - 3*x - 2]
```

Out[72]=

$$(-2 + x)(1 + x)^2$$

- **Together**

In[73]:=

```
?Together
```

Together[expr] puts terms in a sum over a common denominator, and cancels factors in the result.

In[74]:=

```
Clear[x]
Together[1/(x+1)^2 - 2/(x-2) + 9/x]
```

Out[75]=

$$\frac{-18 - 31x - 3x^2 + 7x^3}{(-2 + x)\, x\, (1 + x)^2}$$

- **Apart**

In[76]:=

```
?Apart
```

Apart[expr] rewrites a rational expression as a sum of terms with minimal denominators. Apart[expr, var] treats all variables other than var as constants.

In[77]:=

```
Clear[x]
Apart[(-18-31*x-3*x^2+7*x^3)/((-2+x)*x*(1+x)^2)]
```

Out[78]=

$$\frac{-2}{-2 + x} + \frac{9}{x} + (1 + x)^{-2}$$

- **Simplify**

In[79]:=

```
?Simplify
```

Simplify[expr] performs a sequence of transformations on expr, and returns the simplest form it finds.

In[80]:=
```
Clear[x]
Simplify[Sin[x]^2 + Cos[x]^2 - 1/x + 1/(x+1)]
```

Out[81]=

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

1.2.4 Lists and tables

A list in *Mathematica* is a collection of objects enclosed by braces { }. The collection can be quite general. It may contain numbers, functions, symbols, graphics, or other lists. Here is an example of a list in *Mathematica*.

In[82]:=
```
Clear[apple,x,y,w,z];
a = {-5, apple, 3^(-4), {x,y,{w,z}}}
```

Out[83]=

$$\{-5,\ \text{apple},\ \frac{1}{81},\ \{x,y,\{w,z\}\}\}$$

The *n*th element of a list is referenced by using double brackets, **[[ ]]**. This is a shorthand notation for a built-in function called **Part**. If **a** is a list, then its first element is **a[[1]]**, its second element is **a[[2]]**, etc. If some elements of a list are themselves lists, then the elements of the sublists can be referenced by using more than one argument in the double brackets **[[ ]]**. The following examples illustrate this referencing:

In[84]:=
```
a[[1]]
```

Out[84]=
```
-5
```

In[85]:=
```
a[[4,2]]
```

Out[85]=
```
y
```

In[86]:=
```
a[[4,3,1]]
```

Out[86]=
```
w
```

Some basic entities of mathematics are often represented as lists in *Mathematica*. For example, vectors and matrices are represented as lists in *Mathematica*. Furthermore, *Mathematica* allows the basic arithmetic operators to be applied to lists. Here is the addition operator with two lists.

In[87]:=
```
a = {2,9,5};
b = {-3,1,-6};
a + b
```

Out[89]=
{−1, 10, −1}

We can even apply some functions to lists.

In[90]:=
```
Sqrt[a]
```

Out[90]=
{Sqrt[2], 3, Sqrt[5]}

Some built-in functions in *Mathematica* produce lists as their output. One of these functions is called **Table**.

In[91]:=
```
?Table
```

Table[expr, {imax}] generates a list of imax copies of expr. Table[expr, {i, imax}] generates a list of the values of expr when i runs from 1 to imax. Table[expr, {i, imin, imax}] starts with i = imin. Table[expr, {i, imin, imax, di}] uses steps di. Table[expr, {i, imin, imax}, {j, jmin, jmax}, ...] gives a nested list. The list associated with i is outermost.

As we can see, the **Table** function has several different forms. Let us consider the form: **Table**[*expr*,{*i*,*imin*,*imax*,*di*}]. It generates a list of the values of *expr* when *i* runs from *imin* to *imax* incremented by *di*. Here is an example of a list generated by the **Table** function.

In[92]:=
```
Table[i^2,{i,2,10,1.5}]
```

Out[92]=
{4, 12.25, 25., 42.25, 64., 90.25}

□ 1.3 Algebra and Calculus □

Mathematica is capable of performing many algebraic and calculus problems symbolically. It can compute sums and products of sequences, find roots of a polynomial or solve equations, and differentiate and integrate functions.

1.3.1 Sums and products

Mathematica can be used to compute the sum and product of a sequence. To compute the sum

$$\sum_{i=n}^{m} x_i$$

we use the *Mathematica* **Sum** function. Let us inquire about its format.

In[93]:=
```
?Sum
```

Sum[f, {i, imax}] evaluates the sum of f with i running from 1 to imax. Sum[f, {i, imin, imax}] starts with i = imin. Sum[f, {i, imin, imax, di}] uses steps di. Sum[f, {i, imin, imax}, {j, jmin, jmax}, ...] evaluates a multiple sum.

For example, to compute 1+2+3+...+100, we use the following *Mathematica* expression.

In[94]:=
```
Sum[i,{i,1,100}]
```

Out[94]=
5050

The sum

$$1+\frac{2}{3}+\left(\frac{2}{3}\right)^2+\left(\frac{2}{3}\right)^3+\ldots+\left(\frac{2}{3}\right)^{10}$$

is evaluated using the following expression.

In[95]:=
```
Sum[(2/3)^i,{i,0,10}]
```

Out[95]=
$$\frac{175099}{59049}$$

Symbolic sums are also allowed.

In[96]:=
```
Clear[f];
Sum[f[i],{i,1,3}]
```

Out[97]=
f[1] + f[2] + f[3]

For a multiple sum,

$$\sum_{i=n}^{m}\sum_{j=p}^{q}x_{ij}$$

we again use the **Sum** function with a different form for its arguments, namely, two iterators.

In[98]:=
```
Clear[x];
Sum[x[i,j],{i,1,3},{j,1,i}]
```

Out[99]=
x[1, 1] + x[2, 1] + x[2, 2] + x[3, 1] + x[3, 2] + x[3, 3]

Mathematica cannot symbolically sum many infinite series, say,

$$s = \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^i$$

In[100]:=
```
s = Sum[(2/(3*i))^(i),{i,1,Infinity}]
```

Out[100]=
Sum[$(\frac{2}{3})^i(\frac{1}{i})^i$}, {i, 1, Infinity}]

However, it can often provide an approximation to a sum.

In[101]:=
```
N[s]
```

Out[101]=
0.789567

To compute products such as

$$\prod_{i=n}^{m} x_i \quad \text{or} \quad \prod_{i=n}^{m}\prod_{j=p}^{q} x_{ij}$$

we use the **Product** function. Here is a description of this function.

In[102]:=
```
?Product
```

Product[f, {i, imax}] evaluates the product of f with i running from 1 to imax. Product[f, {i, imin, imax}] starts with i = imin. Product[f, {i, imin, imax, di}] uses steps di. Product[f, {i, imin, imax}, {j, jmin, jmax}, ...] evaluates a multiple product.

Here are two examples.

In[103]:=
```
Clear[x,y];
Product[(i+x)^i,{i,1,5}]
```

Out[104]=
$(1 + x)\ (2 + x)^2\ (3 + x)^3\ (4 + x)^4\ (5 + x)^5$

In[105]:=
```
Product[(i+x)(j+y),{i,1,3},{j,1,2}]
```

Out[105]=
$$(1 + x)^2 (2 + x)^2 (3 + x)^2 (1 + y)^3 (2 + y)^3$$

1.3.2 Solving equations

The **Solve** function in *Mathematica* attempts to find exact solutions of an equation or a system of equations.

In[106]:=
```
?Solve
```

Solve[eqns, vars] attempts to solve an equation or set of equations for the variables vars. Any variable in eqns but not vars is regarded as a parameter. Solve[eqns] treats all variables encountered as vars above. Solve[eqns, vars, elims] attempts to solve the equations for vars, eliminating the variables elims.

As we can see, in the format of this function, the first argument contains the equations to be solved and the second argument contains the variable(s) for which we want solutions. The following *Mathematica* statements solve the cubic polynomial equation $x^3 + 2x^2 - x - 2 = 0$. Notice that in *Mathematica* the equal sign is represented by the logical operator **==**.

In[107]:=
```
Clear[x];
solution = Solve[x^3+2*x^2-x-2==0,x]
```

Out[108]=
$$\{\{x \to -2\}, \{x \to -1\}, \{x \to 1\}\}$$

The output, **solution**, of the **Solve** function is a list of replacement rules in the form $x \to a$. It states that the equation, $x^3 + 2x^2 - x - 2 = 0$, is satisfied if x is replaced by either -2 ($x \to -2$), -1 ($x \to -1$), or 1 ($x \to 1$). If we like the solutions to be expressed as a list, $\{-2, -1, 1\}$, instead of a list of rules, we can use the **ReplaceAll** operator (**/.**) (no space between the **/** and **.**). Here is a description of this built-in function.

In[109]:=
```
?ReplaceAll
```

expr /. rules applies a rule or list of rules in an attempt to transform each subpart of an expression expr.

One can view the **ReplaceAll** function as the function that does a replacement or substitution. For example, the following statement replaces y by 9 in the expression $y^2 + 5$.

In[110]:=
```
Clear[y];
y^2 + 5 /. y->9
```

Out[111]=
86

Similarly, the following statement sequentially replaces y by 9, –2, and 4 in the expression. The result is a list of numbers.

In[112]:=
```
y^2 + 5 /. {{y->9},{y->-2},{y->4}}
```

Out[112]=
{86, 9, 21}

The following statements replace the list of three rules obtained from solving the cubic polynomial equation by the list of solutions, {a,b,c} = {–2, –1, 1}.

In[113]:=
```
Clear[x];
solution = Solve[x^3+2*x^2-x-2==0,x]
```

Out[114]=
{{x → –2}, {x → –1}, {x → 1}}

In[115]:=
```
{a,b,c} = x /. solution
```

Out[115]=
{–2, –1, 1}

For many equations, *Mathematica* cannot find exact solutions. In fact, there are no general mathematical methods for finding explicit solutions of a polynomial equation of degree five or more. For example, *Mathematica* is not able to find the roots of the polynomial $x^5 + x^3 + 1 = 0$.

In[116]:=
```
Clear[x];
Solve[x^5+x^3+1==0,x]
```

Out[117]=
{ToRules[Roots[x^3+ x^5 == –1, x]]}

This output indicates that **Solve** could not find the solutions to the equation. However, there is a companion function to **Solve**, called **NSolve**, that attempts to find numerical solutions. Here is a description of this function.

In[118]:=
```
?NSolve
```

NSolve[eqns, vars] attempts to solve numerically an equation or set of equations for the variables vars. Any variable in eqns but not vars is regarded as a parameter. NSolve[eqns] treats all variables encountered as vars above. NSolve[eqns, vars, prec] attempts to solve numerically the equations for vars using prec digits precision.

Using **NSolve** to find approximations to the roots of the above polynomial, we find that it is has one real root and four complex roots.

```
In[119]:=
    Clear[x];
    NSolve[x^5+x^3+1==0,x]

Out[120]=
    {{x → -0.83762}, {x → -0.217853 - 1.16695 I},
      {x → -0.217853 + 1.16695 I}, {x → 0.636663 - 0.664702 I},
      {x → 0.636663 + 0.664702 I}}
```

Finding solutions of transcendental equations, it is often necesary to employ a numerical scheme such as Newton's Method. *Mathematica* implements other such numerical methods in the function **FindRoot**.

```
In[121]:=
    ?FindRoot

    FindRoot[lhs == rhs, {x, x0}] searches for a numerical solution to the
      equation lhs == rhs, starting with x == x0.
```

The first argument of **FindRoot** contains the equation to be solved. The second argument is a list that includes the variable for which we want to solve and some initial guess to the solution. For example, let us find an approximation for the solution of the equation $x = \log(x) + 2x^2$.

```
In[122]:=
    Clear[x];
    FindRoot[x==Log[x]+2*x^2,{x,1}]

Out[123]=
    {x → 0.723576}
```

We note that there may be other solutions. To find them, we choose different initial guesses.

A system of equations can also be solved with the functions **Solve**, **NSolve**, and **FindRoot**. For example, the system of equations

$$x + y = 1$$
$$xy = -2$$

has two solutions, (–1,2) and (2,–1).

```
In[124]:=
    Clear[x,y];
    Solve[{x+y==1,x*y==-2},{x,y}]

Out[125]=
    {{x → -1, y → 2}, {x → 2, y → -1}}
```

1.3.3 Calculus

Three basic calculus operations are evaluating limits, derivatives, and integrals. *Mathematica* has the functions, **Limit**, **D**, and **Integrate**, to perform these operations.

The **Limit** function attempts to evaluate the limit of an expression as one of the symbols in the expression approaches a particular value.

In[126]:=
```
?Limit
```

Limit[expr, x→x0] finds the limiting value of expr when x approaches x0.

Here is an example of finding the limit of a rational expression.

In[127]:=
```
Clear[x];
Limit[(x^3-3*x^2+3*x-1)/(x-1),x->1]
```

Out[128]=
```
0
```

Derivatives in *Mathematica* are computed using the **D** function.

In[129]:=
```
?D
```

D[f, x] gives the partial derivative of f with respect to x. D[f, {x, n}] gives the nth partial derivative with respect to x. D[f, x1, x2, ...] gives a mixed derivative.

In its most basic form, the first argument of **D** is the function to be differentiated and the second argument is the variable of differentiation. For example, to differentiate the function $x\cos(2x)$ with respect to x, we would use the following expression.

In[130]:=
```
Clear[x];
D[x*Cos[2*x],x]
```

Out[131]=
```
Cos[2x] - 2xSin[2x]
```

An alternative method of differentiating functions of a single variable is to use the prime notation (′) for the derivative.

In[132]:=
```
Clear[f,x];
f[x_]:= x*Cos[2*x];
f'[x]
```

Out[134]=
```
Cos[2x] - 2xSin[2x]
```

Higher order derivatives can be computed by nesting the **D** function, using multiple primes, or specifying a list in the second argument of **D**. The list should be of the form $\{x,n\}$ where n is a positive integer that denotes the order of the derivative. For example, to compute the third derivative of the above function, we could use any of the following expressions.

In[135]:=
```
Clear[x];
D[D[D[x*Cos[2*x],x],x],x]
```

Out[136]=
$-12\text{Cos}[2x] + 8x\text{Sin}[2x]$

In[137]:=
```
f'''[x]
```

Out[137]=
$-12\text{Cos}[2x] + 8x\text{Sin}[2x]$

In[138]:=
```
D[x*Cos[2*x],{x,3}]
```

Out[138]=
$-12\text{Cos}[2x] + 8x\text{Sin}[2x]$

If an expression involves more than one variable, then the **D** function computes partial derivatives. Higher order partial derivatives are found by adding arguments to the **D** function. For example, to compute the partial derivative

$$\frac{\partial^3}{\partial x \partial y^2}(x^3y^2 + x^4)$$

we would use the following *Mathematica* statement.

In[139]:=
```
Clear[x,y];
D[x^3*y^2+x^4,y,y,x]
```

Out[140]=
$6x^2$

The total differential of a function can be computed with the **Dt** function. Here is *Mathematica*'s description of this function.

In[141]:=
```
?Dt
```

Dt[f, x] gives the total derivative of f with respect to x. Dt[f] gives the total differential of f. Dt[f, {x, n}] gives the nth total derivative with respect to x. Dt[f, x1, x2, ...] gives a mixed total derivative.

Let us compute the total differential of xy^2+z^3.

In[142]:=
```
Clear[x,y,z];
Dt[x*y^2+z^3]
```

Out[143]=
y^2Dt[x] + 2xyDt[y] + $3z^2$Dt[z]

In this output, **Dt[x]** plays the role of dx, **Dt[y]** the role of dy, and **Dt[z]** the role of dz.

Mathematica is a very powerful integrator. The **Integrate** function can be used to evaluate both definite and indefinite integrals. Here is the description of this function.

In[144]:=
```
?Integrate
```

Integrate[f,x] gives the indefinite integral of f with respect to x.
Integrate[f,{x,xmin,xmax}] gives the definite integral.
Integrate[f,{x,xmin,xmax},{y,ymin,ymax}] gives a multiple integral.

As an example, we compute the definite integral

$$\int_0^{\pi} x\cos(x)\mathrm{d}x$$

In[145]:=
```
Clear[x];
Integrate[x*Cos[x],{x,0,Pi}]
```

Out[146]=
−2

Let us find an antiderivative (indefinite integral) of $x^2 \sin x$.

In[147]:=
```
Clear[x];
Integrate[x^2*Sin[x],x]
```

Out[148]=
2Cos[x] − x^2Cos[x] + 2xSin[x]

Notice that *Mathematica* does not include an arbitrary constant of integration. Our last example is the following iterated integral

$$\int_{-1}^{2}\left[\int_{x}^{x^2} xy^2\mathrm{d}y\right]\mathrm{d}x$$

In[149]:=
```
Clear[x,y];
Integrate[Integrate[x*y^2,{y,x,x^2}],{x,-1,2}]
```

Out[150]=
$$\frac{337}{40}$$

□ 1.4 Graphics in *Mathematica* □

One of the outstanding features of *Mathematica* is its graphing capabilities. It can graph functions of one or two variables or even functions specified parametrically.

1.4.1 Plot

To graph a function of a single variable, $y = f(x)$ on an interval $[a,b]$, we use the **Plot** function. Here is *Mathematica*'s description of this function.

In[151]:=

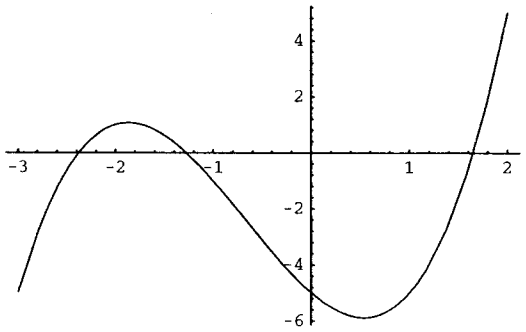
```
?Plot
```

Plot[f, {x, xmin, xmax}] generates a plot of f as a function of x from xmin to xmax. Plot[{f1, f2, ...}, {x, xmin, xmax}] plots several functions fi.

In its most basic form, the first argument of **Plot** is the function to be plotted and the second argument is a list that specifies the interval over which the function is to be plotted. The following expression graphs the function $f(x) = x^3+2x^2-3x-5$ over the interval $-3 \le x \le 2$.

In[152]:=

```
Plot[x^3+2*x^2-3*x-5,{x,-3,2}];
```



The **Plot** function may have more than two arguments. These additional arguments are called *options* and are used to enhance the graph. There are several plotting options available in *Mathematica*. We can inquire about them by using the *Mathematica* function **Options**.

In[153]:=

```
Options[Plot]
```

Out[153]=

{AspectRatio → $\frac{1}{\text{GoldenRatio}}$, Axes → Automatic,
AxesLabel → None, AxesOrigin → Automatic,
AxesStyle → Automatic, Background → Automatic,
ColorOutput → Automatic, Compiled → True,

```
DefaultColor → Automatic, Epilog → { }, Frame → False,
FrameLabel → None, FrameStyle → Automatic,
FrameTicks → Automatic, GridLines → None, MaxBend → 10.,
PlotDivision → 20., PlotLabel → None, PlotPoints → 25,
PlotRange → Automatic, PlotRegion → Automatic,
PlotStyle → Automatic, Prolog → { }, RotateLabel → True,
Ticks → Automatic, DefaultFont :> $DefaultFont,
DisplayFunction :> $DisplayFunction}
```

Notice that the options are in the form of rules and these rules represent the default settings. Many of these are self-explanatory. The default settings of these options suffice for plotting most functions. Some of the more useful options are **AxesLabel** (labels the horizontal and vertical axes), **AspectRatio** (determines the scaling of the axes with respect to one another), **PlotRange** (specifies the range of the vertical and/or horizontal axes), **PlotStyle** (specifies the line style used to draw the graphs), and **PlotLabel** (puts a title on the graph). Here are the descriptions of some of these options.

In[154]:=
```
?AxesLabel
```

AxesLabel is an option for graphics functions. With AxesLabel → None, no labels are drawn on the axes in the plot. AxesLabel → label specifies a label for the y axis of a two-dimensional plot, and the z axis of a three-dimensional plot. AxesLabel → {xlabel, ylabel, ... } specifies labels for different axes.

In[155]:=
```
?AspectRatio
```

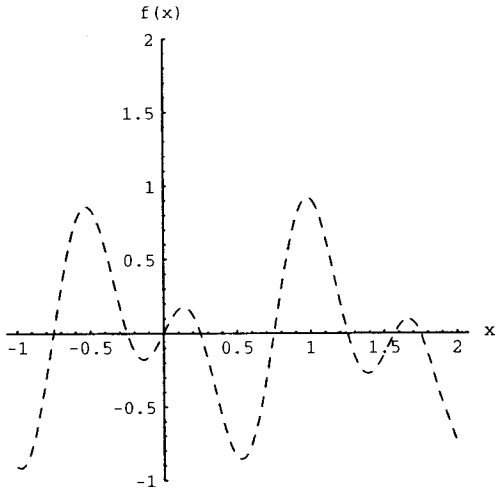
AspectRatio is an option for Show and related functions. With AspectRatio → Automatic, the ratio of height to width of the plot is determined from the actual coordinate values in the plot. AspectRatio → r makes the ratio equal to r.

In[156]:=
```
?PlotStyle
```

PlotStyle is an option for Plot, ParametricPlot and ListPlot. PlotStyle → style specifies that all lines or points are to be generated with the specified graphics directive, or list of graphics directives. PlotStyle → {{style1}, {style2}, ... } specifies that successive lines generated should use graphics directives style1, style2, ...

As an example, let us plot the function $y = \sin(2x)\cos(2\pi x)$ over the interval $-1 \leq x \leq 2$ using several options.

In[157]:=
```
Plot[Sin[2*x]*Cos[2*Pi*x],{x,-1,2},
      AxesLabel->{"x","f(x)"},PlotRange->{-1,2},
      AspectRatio->Automatic,
      PlotStyle->Dashing[{0.015}]];
```

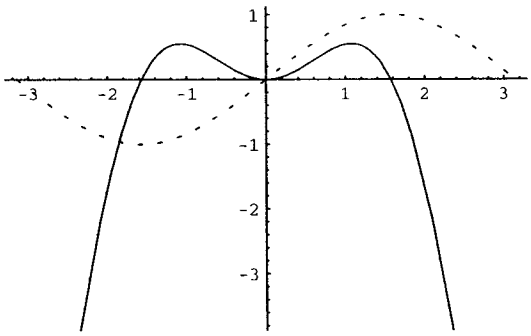


In the above input statement, we have indented the second, third, and fourth lines of the **Plot** expression for clarity. When a single *Mathematica* expression extends over more than one line, we will indent the beginning of each line beyond the first line to indicate this situation.

It is possible to plot two or more functions on the same set of axes. The functions to be plotted must be grouped as a list {*f1*,*f2*, . . }. Each function can have a different **PlotStyle**. This is accomplished by entering the plot option in the form of **PlotStyle→**{*plotsty1*, *plotsty2*, ..}. The following example plots two functions, $x^2\cos x$ and sin x, over the interval $-\pi \le x \le \pi$ with the first graph using the default style and the second graph using a dashed style. The default style is denoted as the empty list { }.

In[158]:=

```
Plot[{x^2*Cos[x],Sin[x]},{x,-Pi,Pi},
        PlotStyle->{{},Dashing[{0.005,0.02}]}];
```



1.4.2 Plot3D

Functions of two variables are plotted using the **Plot3D** function. The function **Plot3D** is very similar to **Plot** in its format.

In[159]:=
```
?Plot3D
```

Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}] generates a three-dimensional plot of f as a function of x and y. Plot3D[{f, s}, {x, xmin, xmax}, {y, ymin, ymax}] generates a three-dimensional plot in which the height of the surface is specified by f, and the shading is specified by s.

Some options are different because the graph will be a surface in a three-dimensional space. Here are the default settings for these options.

In[160]:=
```
Options[Plot3D]
```

Out[160]=
```
{AmbientLight → GrayLevel[0], AspectRatio → Automatic,
  Axes → True, AxesEdge → Automatic, AxesLabel → None,
  AxesStyle → Automatic, Background → Automatic, Boxed → True,
  BoxRatios → {1, 1, 0.4}, BoxStyle → Automatic,
  ClipFill → Automatic, ColorFunction → Automatic,
  ColorOutput → Automatic, Compiled → True,
  DefaultColor → Automatic, Epilog → { }, FaceGrids → None,
  HiddenSurface → True, Lighting → True,
  LightSources →
   {{{1., 0., 1.}, RGBColor[1, 0, 0]},
    {{1., 1., 1.}, RGBColor[0, 1, 0]},
    {{0., 1., 1.}, RGBColor[0, 0, 1]}}, Mesh → True,
  MeshStyle → Automatic, PlotLabel → None, PlotPoints → 15,
  PlotRange → Automatic, PlotRegion → Automatic,
  Plot3Matrix → Automatic, Prolog → { }, Shading → True,
  SphericalRegion → False, Ticks → Automatic,
  ViewCenter → Automatic, ViewPoint → {1.3, -2.4, 2.},
  ViewVertical → {0., 0., 1.}, DefaultFont :> $DefaultFont,
  DisplayFunction :> $DisplayFunction}
```

Some of these options require some explanation. For example, let us check the meaning of the options **BoxRatios** and **ViewPoint**.

In[161]:=
```
?BoxRatios
```

BoxRatios is an option for Graphics3D and SurfaceGraphics. BoxRatios → {rx, ry, rz} gives the ratios of side lengths for the bounding box of the three-dimensional picture. BoxRatios → Automatic determines the ratios using the range of actual coordinate values in the plot.

In[162]:=
```
?ViewPoint
```

ViewPoint is an option for Graphics3D and SurfaceGraphics which gives the point in space from which the objects plotted are to be viewed. ViewPoint → {x, y, z} gives the position of the view point relative to the center of the three-dimensional box that contains the object being plotted.

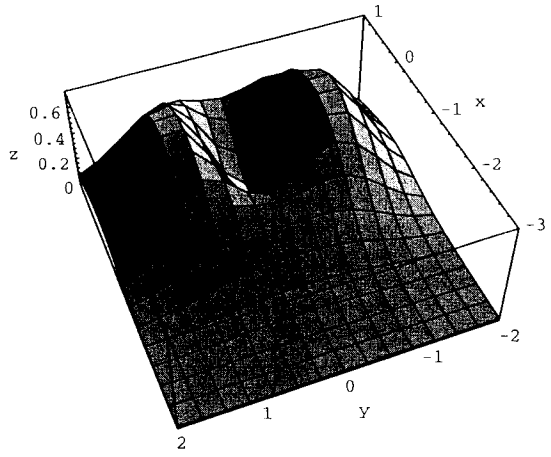
To illustrate this function, let us plot the graph of the function

$$f(x,y) = (x^2+2y^2)\, e^{-(x^2+y^2)}$$

over the rectangle $-3 \le x \le 1$ and $-2 \le y \le 2$.

In[163]:=

```
Plot3D[(x^2+2*y^2)*Exp[-(x^2+y^2)],{x,-3,1},{y,-2,2},
                AxesLabel->{"x","y","z"},
              ViewPoint->{-2.270,  0.910,  2.580}];
```



1.4.3 **ParametricPlot** and **ParametricPlot3D**

Parametrically defined functions such as $x = f(t)$, $y = g(t)$, $a \le t \le b$, in two-dimensions, and $x = f(t)$, $y = g(t)$, $z = h(t)$, $a \le t \le b$, in three-dimensions can be graphed using **ParametricPlot** and **ParametricPlot3D**, respectively. *Mathematica* provides the following information on these functions.

In[164]:=

```
?ParametricPlot
```

ParametricPlot[{fx, fy}, {t, tmin, tmax}] produces a parametric plot with x and y coordinates fx and fy generated as a function of t. ParametricPlot[{{fx, fy}, {gx, gy}, ...}, {t, tmin, tmax}] plots several parametric curves.

In[165]:=

```
?ParametricPlot3D
```

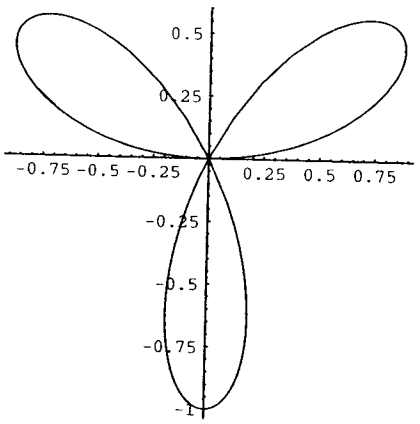
ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}] produces a three-dimensional space curve parameterized by a variable t which runs from tmin to tmax. ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}, {u, umin, umax}] produces a three-dimensional surface parametrized by t and u. ParametricPlot3D[{fx, fy, fz,

s}, ...] shades the plot according to the color specification s. ParametricPlot3D[{{fx, fy, fz}, {gx, gy, gz}, ...}, ...] plots several objects together.

Options for these functions are similar to **Plot** and **Plot3D**. Here are three examples.

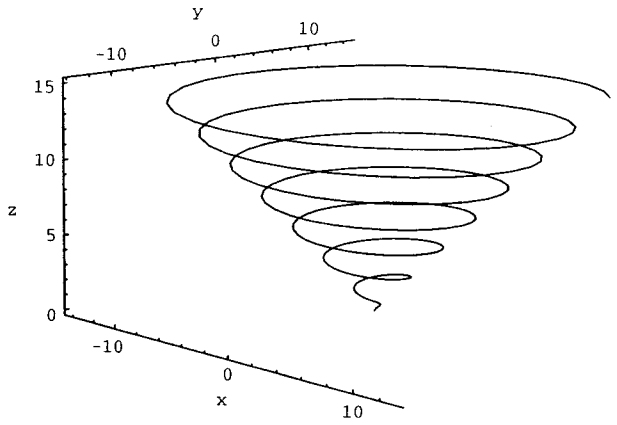
In[166]:=

```
ParametricPlot[{Sin[3*t]*Cos[t],Sin[t]*Sin[3*t]},
                {t,-Pi,Pi},AspectRatio->1];
```



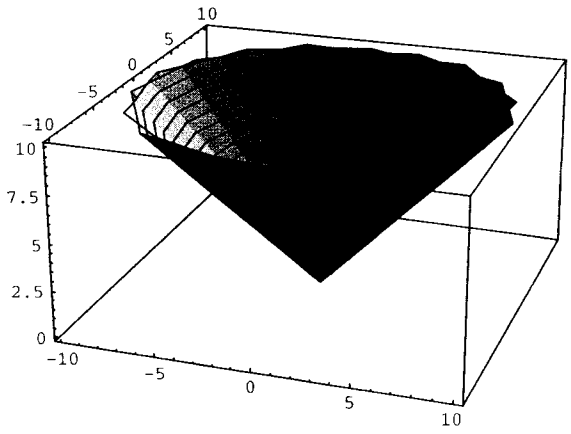
In[167]:=

```
ParametricPlot3D[{t*Sin[3*t],t*Cos[3*t],t},{t,0,15},
        PlotPoints->200,ViewPoint->{2.737,-3.416,0.906},
    Boxed->False,AxesLabel->{"x","y","z"}];
```



In[168]:=

```
ParametricPlot3D[{t*Cos[u],t*Sin[u],t},{t,0,10},
    {u,-2Pi,2Pi},ViewPoint->{1.368,-4.101,1.712}];
```



1.4.4 ContourPlot

The **ContourPlot** function produces a contour map of a surface defined by an equation $z = f(x,y)$ for various values of z. The following is the *Mathematica* information on **ContourPlot**.

In[169]:=
```
?ContourPlot
```

ContourPlot[f, {x, xmin, xmax}, {y, ymin, ymax}] generates a contour plot of f as a function of x and y.

This function has several options.

In[170]:=
```
Options[ContourPlot]
```

Out[170]=
```
{AspectRatio → 1, Axes → False, AxesLabel → None,
  AxesOrigin → Automatic, AxesStyle → Automatic,
  Background → Automatic, ColorFunction → Automatic,
  ColorOutput → Automatic, Compiled → True,
  ContourLines → True, Contours → 10, ContourShading → True,
  ContourSmoothing → True, ContourStyle → Automatic,
  DefaultColor → Automatic, Epilog → { }, Frame → True,
  FrameLabel → None, FrameStyle → Automatic,
  FrameTicks → Automatic, PlotLabel → None, PlotPoints → 15,
  PlotRange → Automatic, PlotRegion → Automatic, Prolog → { },
  RotateLabel → True, Ticks → Automatic,
  DefaultFont :> $DefaultFont,
  DisplayFunction :> $DisplayFunction}
```

Let us check the meaning of **Contours** and **ContourShading**.

In[171]:=
```
?Contours
```

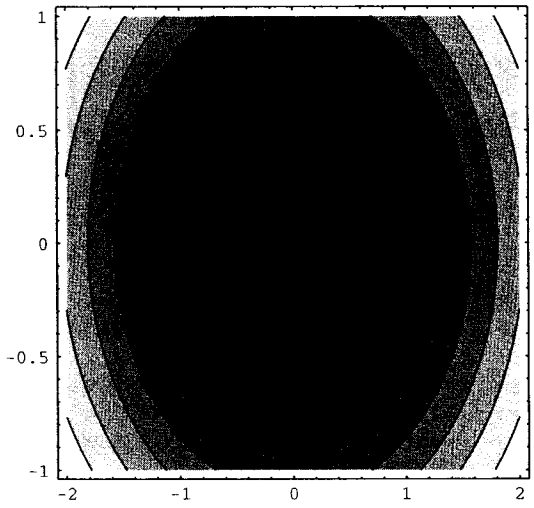
Contours is an option for ContourGraphics specifying the contours to use. Contours → n chooses n equally spaced contours between the minimum and maximum z values. Contours → {z1, z2, ... } specifies the explicit z values to use for contours.

In[172]:=
```
?ContourShading
```

ContourShading is an option for contour plots. With ContourShading → False, regions between contour lines are left blank. With ContourShading → True, regions are colored based on the setting for the option ColorFunction.

As an example, let us plot the contour of $f(x,y) = \sqrt{x^2+2y^2}$. The contour plots are by default shaded in such a way that regions with higher z values are shaded more lightly than those lower z values.

In[173]:=
```
Clear[f,x,y];
f[x_,y_]:= Sqrt[x^2 + 2*y^2];
           ContourPlot[f[x,y],{x,-2,2},{y,-1,1},
           PlotPoints->25];
```



1.4.5 Graphics primitives and the **Show** function

Graphics in *Mathematica* are composed of graphics primitives. In two dimensions, the graphics are constructed using some basic graphics objects: points, lines, polygons, circles, and text. These graphics primitives are constructed using the *Mathematica* functions **Point**, **Line**, **Polygon**, **Circle**, and **Text**. Below are descriptions of these primitives.

In[176]:=
```
?Point
```

Point[coords] is a graphics primitive that represents a point.

```
In[177]:=
    ?Line

    Line[{pt1, pt2, ...}] is a graphics primitive which represents a line
      joining a sequence of points.

In[178]:=
    ?Polygon

    Polygon[{pt1, pt2, ...}] is a graphics primitive that represents a
      filled polygon.

In[179]:=
    ?Circle

    Circle[{x, y}, r] is a two-dimensional graphics primitive that represents
      a circle of radius r centered at the point {x, y}. Circle[{x, y}, {rx, ry}]
      yields an ellipse with semi-axes rx and ry. Circle[{x, y}, r,
      {theta1, theta2}] represents a circular arc.

In[180]:=
    ?Text

    Text[expr, coords] is a graphics primitive that represents text
      corresponding to the printed form of expr, centered at the
      point specified by coords.
```

Graphics directives such as **Thickness**, **RGBColor**, **PointSize**, **Dashing**, etc., control the ways these basic objects are constructed. Let us check two of these directives.

```
In[181]:=
    ?Thickness

    Thickness[r] is a graphics directive which specifies that lines which
      follow are to be drawn with a thickness r. The thickness r is given
      as a fraction of the total width of the graph.

In[182]:=
    ?RGBColor

    RGBColor[red, green, blue] is a graphics directive which specifies
      that graphical objects which follow are to be displayed, if possible,
      in the color given.
```

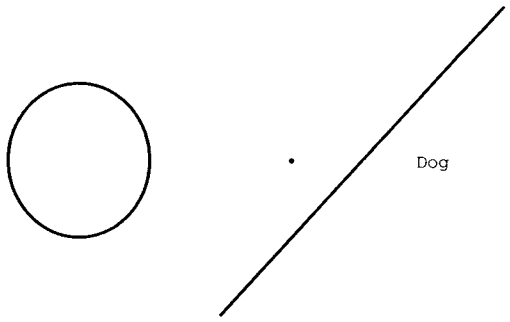
The graphics primitives and their directives are assembled by another *Mathematica* function. In two-dimensions, this function is **Graphics** and in three-dimensions it is **Graphics3D.** As an example, we construct a two-dimensional graphic that contains a point at (1/4,1/2), a thick line between the points (0,0) and (1,1), a circle of radius 1/4 centered at (–1/2,1/2), and the text "Dog" centered at the point (3/4,1/2).

```
In[183]:=
    picture = Graphics[{Point[{1/4,1/2}],Thickness[0.01],
                        Line [{{0,0}, {1,1}}],
                       Circle[{-1/2,1/2},1/4],
                       Text["Dog",{3/4,1/2}]}];
```

Graphics constructs the graphics named **picture**, but it does not display it. To display the graphics, *Mathematica* uses the **Show** function.

In[184]:=

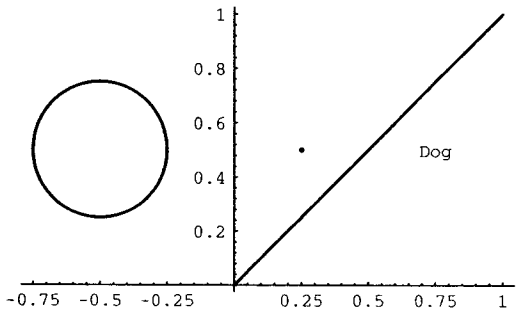
```
Show[picture];
```



Notice that the directive **Thickness[0.01]** is in effect for both the line and the circle. The **Show** function has several options. We will display the above graphic using two of these options.

In[185]:=

```
Show[picture,Axes->True,AspectRatio->Automatic];
```



Options can also be placed in the **Graphics** and **Graphics3D** function. For example, here are the options for Graphics.

In[186]:=

```
Options[Graphics]
```

Out[186]=

$$\{\text{AspectRatio} \to \frac{1}{\text{GoldenRatio}}, \text{Axes} \to \text{False}, \text{AxesLabel} \to \text{None},$$
$$\text{AxesOrigin} \to \text{Automatic}, \text{AxesStyle} \to \text{Automatic},$$
$$\text{Background} \to \text{Automatic}, \text{ColorOutput} \to \text{Automatic},$$
$$\text{DefaultColor} \to \text{Automatic}, \text{Epilog} \to \{\}, \text{Frame} \to \text{False},$$
$$\text{FrameLabel} \to \text{None}, \text{FrameStyle} \to \text{Automatic},$$
$$\text{FrameTicks} \to \text{Automatic}, \text{GridLines} \to \text{None}, \text{PlotLabel} \to \text{None},$$
$$\text{PlotRange} \to \text{Automatic}, \text{PlotRegion} \to \text{Automatic}, \text{Prolog} \to \{\},$$
$$\text{RotateLabel} \to \text{True}, \text{Ticks} \to \text{Automatic},$$

```
DefaultFont :> $DefaultFont,
DisplayFunction :> $DisplayFunction}
```

□ 1.5 Modules and Packages □

Often, we would like to group a number of *Mathematica* statements together to perform a specific task. One easy way to do this is to use the **Module** function. Here is *Mathematica*'s description of this function.

In[187]:=

```
?Module
```

Module[{x, y, ...}, expr] specifies that occurrences of the symbols x, y, ... in expr should be treated as local. Module[{x = x0, ...}, expr] defines initial values for x, ...

The first argument of **Module** is a list of symbols that are used internally in the the module. The second argument contains the instructions that *Mathematica* is to follow in the module. We can thus define a function that requires several steps. For example, the following function generates a circle that is centered at (a,b) with radius R and places a label at the point. We will call this function **labeledcircle**. The first argument **pt** is a list that represents the point (a,b), the second argument **rad** is the radius R, and the third argument **label** is a label.

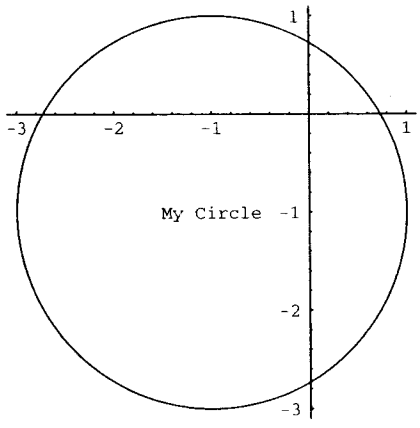
In[188]:=

```
Clear[labelcircle,pt];
labelcircle[pt_,rad_,label_]:=
      Module[{round,txt},round=Circle[pt,rad];
                          txt=Text[label,pt];
            Graphics[{round,txt}]]
```

Thus, the following statement creates a graphics object, named **picture**, which is a circle centered at $\{-1,1\}$ with radius 2 and a label, "My Circle." To display the output of this function, we use the **Show** function with a couple of options.

In[190]:=

```
picture = labelcircle[{-1,-1},2,"My Circle"];
Show[picture,AspectRatio->Automatic,Axes->True];
```



A module is a convenient way to organize a group of *Mathematica* statements. Many of the special functions that we use in this book have function definitions that are defined as modules. We have collected these function definitions into a package called **MathEcon**. This package is defined in a *Mathematica* notebook (file) called **MathEcon.m**. The package can be loaded into the kernel by using the **Needs** function. However, before the **MathEcon** package can be loaded, a path must be established to inform *Mathematica* of the location of the package. Suppose the notebook **MathEcon.m** is stored in a folder named **MathFunctions** on the hard-disk drive called **mydisk**. The following statement establishes a path to the package.

```
AppendTo[$Path,"mydisk:MathFunctions"];
```

Once the path is established, we can load the **MathEcon.m** package into the *Mathematica* kernel using the **Needs** function.

```
Needs["MathEcon`"]
```

The functions that are defined in **MathEcon.m**, are now available to use. We ask *Mathematica* to give us a description of the package.

In[192]:=

```
?MathEcon
```

MathEcon is a package of functions that is used with the book:

Mathematics and Mathematica for Economists
by Cliff Huang and Philip Crooke.

It contains the following functions:

fibprime directed directed3d
vector2d vector3d addition2d
subtraction2d projection2d projection3d
rotation2d rank rowop
colop submatrix minor
cofactor aug colrep
signQ signQL borderB
gradf hessian conhess
arc2d tangentline fjohn
separableode linearode tangentfield
wronskian wis trajectory

As an example, we have created a function called **fibprime** that will find all of the Fibonacci numbers that are prime and less than or equal to a given integer. The Fibonacci numbers z_k are those integers that satisfy the the difference equation

$$z_k = z_{k-1} + z_{k-2},\ k = 0,1,2,\ldots$$

with $z_0 = z_1 = 1$. Let us look up the format of this function.

In[193]:=

```
?fibprime
```

fibprime[n] calculates the prime Fibonacci numbers that are less than or equal to an integer n. For example,

fibprime[37].

For example, the first 6 Fibonacci numbers are {1, 1, 2, 3, 5, 8}. Having loaded the file **MathEcon.m**, we can use this function to find the prime Fibonacci numbers less than or equal to a given integer. Let us find the prime Fibonacci numbers that are less than or equal to 100.

In[194]:=
```
fibprime[100]
```

Out[194]=
```
{2, 3, 5, 13, 89}
```

To distinguish between the functions defined in the **MathEcon** package and the built-in functions of *Mathematica*, all the **MathEcon** functions start with a lower case letter. Once these functions have been loaded, they are available in the **Function Browser** of the **Help** menu.

Mathematica comes with many other specialized packages that the user can read into the kernel. For example, the normal and other continuous probability distributions are contained in the **Statistics`ContinuousDistribution`** package. Let us first load the package into the kernel.

In[195]:=
```
Needs["Statistics`ContinuousDistributions`"]
```

We evaluate the probability density function, **PDF**, of the variable x having the normal distribution with zero mean and unit standard deviation, **NormalDistribution[0,1]**.

In[196]:=
```
normal=PDF[NormalDistribution[0,1],x]
```

Out[196]=

$$\frac{1}{E^{x^2/2}\,\text{Sqrt}[2\text{Pi}]}$$

The probability density function is plotted for $-3 \le x \le 3$.

In[197]:=
```
Plot[normal,{x,-3,3},AxesLabel->{"x","f(x)"}];
```