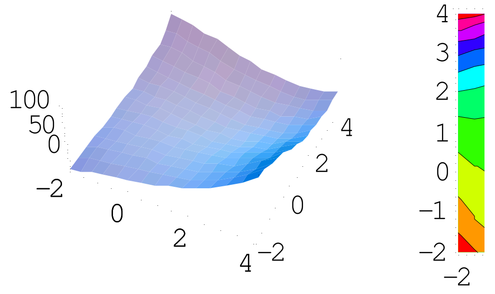
Introduction to Mathematica
A Tutorial

Jennifer Voitle 2000



Contents

Introduction

Mathematica is a system for doing mathematics on the computer.It can do numerics,symbolics,graphics and is also a programming language.Mathematica has infinite precision.It can plot functions of a single variable; make contour,surface and density plots with special shading and lighting effects (in plots,the functions may be defined analytically or in tabular form); perform algebraic manipulations on polynomials; solve linear and nonlinear equations and systems of equations; compute symbolic and numeric values of total and partial derivatives,and do animation with Mathematica-generated graphics or with graphics imported from other applications.It has extensive curve-fitting capabilities by regression polynomials,interpolating polynomials,splines and series approximations.It can work with Laplace and Fourier transforms and solve systems of differential equations,both numerically and symbolically.It has hundreds of built-in functions such as the Error,Gamma,Bessel and Legendre functions as well as some rather exotic ones.It can work with both real and complex numbers.Mathematica is also a rich programming language,containing all of the standard constructs of normal languages such as FORTRAN or C but with all of the symbolic power of Mathematica.Mathematica functions can be compiled and called from Excel,C++ or Visual Basic (to name a few).Thus,Mathematica can be taught to do anything that other languages can do such as neural network analysis,image processing and Finite Element Analysis.Beyond this,the notebook interface allows creation of truly dynamic documents not possible in any other system.In a notebook,all of the supporting code,illustrations,hyperlinks and text to support a paper are all self-contained.In addition, Mathematica documents can be converted to HTML.Although note as of the latest release (v 4.0) there are problems with the HTML conversion process as you will see if you try to actually upload it to a website.

Getting Started It is very important to follow Mathematica's naming conventions. Mathematica is case-sensitive. All reserved words begin with uppercase letters, with all other letters lowercase. When two words are concatenated, an uppercase letter begins each word. These are the only times when uppercase letters are used. A list of some of the more common function names is provided at the end of this document. Once you understand Mathematica's syntax, using it becomes almost intuitive. For example, the command for plotting is Plot, and FindRoot is the function that solves nonlinear equations by the Newton-Raphson or Secant techniques. Mathematica essentially works by pattern-matching, and at the heart of it is this rule:

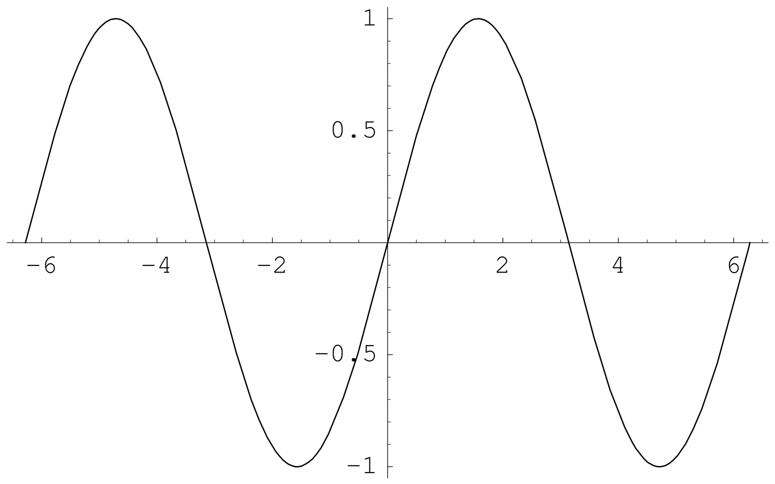
Given an object, apply transformation rules to it until there are no longer any changes.

Executing a Command Typing commands into Mathematica is very similar to typing with a word processor. Try typing a command such as the following:

```
Plot[Sin[x],{x,-2Pi,2Pi}];
```

Type it exactly as shown above.Note that a times sign is not necessary between the 2 and the symbol Pi; the multiplication will be performed automatically.If the command Plot[Sin[x],{x,-2Pi,2Pi}]; just beeps and prints the expression back at you,it means that a syntax error has occurred:Mathematica is unable to understand what you mean because perhaps you have typed plot instead of Plot,or used round brackets () rather than square.After typing the command,press the Enter key (or press Shift-Enter.) This will execute your command.Pressing the return key alone is insufficient as a command may be several lines long,such as when writing a program.You will press Enter or Shift-Enter each time you wish to ask Mathematica a question.The resulting plot should look like the following:

```
Plot[Sin[x], {x, -2 π, 2 π}]
```



To learn more about Mathematica, it is suggested that you work through this manual. Please be aware that some of the plots may take a long time to execute.
NOTE: In the following, the boldface commands such as Plot[Sin[x],{x,-2Pi,2Pi}]; are what you type; anything else is a remark or output.

Fundamental Arithmetic and Algebraic Operations

Mathematica can do all types of arithmetic and algebraic manipulations. The following symbols are used for the operations of addition, subtraction, multiplication and so on:

Operation	Mathematica Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^
Matrix Multiplication	.

Mathematical Constants Mathematica has hundreds of built-in constants. For example, e is given by E or the function Exp[x]; pi is given by Pi.

Scientific Notation To represent the number mE0n, where m and n are any values, one would write m 10^n in *Mathematica*. For example, 5.667 E-08 (which is 0.00000005667) would be written 5.667 10^-08. (What do you get if you actually write 5.667 E-08? Why?)

Brackets *Mathematica* has four types of brackets:

Symbol	Explanation
()	Parentheses are used for grouping and ordering operations.
{}	Curly brackets are used for elements of lists and arrays
[]	Square brackets are used for function arguments
[[]]	Double square brackets are used for elements of matrices and lists

Equal Signs Mathematica has several ways of testing equality, but we will mainly be interested in only the following:

Symbol	Meaning
=	x = y means set x to the value of y immediately
==	x == y means test whether x and y are equal. True or False is re
:=	x := y means set delayed. x is not set equal to y immediately, but t is remembered and when x is called, the definition y is substituted.
!=	x != y tests x not equal to y. True or False is returned

Logical Operators The logical And, Or are carried out via the && and || operators respectively. x And y would be written x && y while x Or y would be evaluated x || y. (The || symbol is made by pressing the shift-\ key twice.)

User-defined Variables Variable names can be any length desired. However, they cannot conflict with Mathematica keywords. Since Mathematica keywords always begin with an uppercase letter, conflict may be avoided by always beginning user-defined variables with a lowercase letter. (It is possible to override built-in definitions, but this is usually unnecessary.) Variable names must begin with a letter or $ sign and may combine any combination of numeric and upper and lowercase alphabetic characters. No spaces may appear in variable names as a space between symbols usually implies multiplication. You also may not use the letters C, D, E, I, N or O as these are reserved. note however that the lowercase equivalents c, d, e, i, n and o are valid choices. Variable names may not include any of the following: .,%^+/*-_#()[]{} | ? to name a few.

Some examples and nonexamples:
(1) x2 is a legal variable name while 2x is not. Since a multiplication symbol is not required between operands and variables aer not allowed to begin with numerals, 2x will be evaluated as twice the current value of x. Should you attempt to set 2x to something, such as 10, you will get an error:

```
2 x = 10
Set::write : Tag Times in 2 x is Protected.
10
```

(2) The variable MaxIterations is allowed while neither Max_Iterations nor Max.Iterations are valid.

(3) The variables Tan, Sin, Cos, Sinh, Log, Max and the like are not allowed since they are reserved keywords. However, valid variable names may include reserved keywords, such as myTan and Tangent.

Mathematica's Built In Symbols To see a list of all of Mathematica's intrinsic keywords, type ?* To see a list of all key words starting wtih a certain letter, or containing partial phrases, use * as a wildcard character. For example, all of the keywords that start with Z are

```
?Z*
```

```
ZeroTest       ZeroWidthTimes Zeta           ZTransform
```

More detailed information is available by using the double question mark:

```
?? ZeroTest
```

```
ZeroTest is an option for LinearSolve and other linear algebra functions,
  which gives a function to be applied to combinations of matrix
  elements to determine whether they should be considered equal to zero.

Attributes[ZeroTest] = {Protected}
```

Command-completion is very helpful in situations where you don't remember the full name of the function you want, or don't want to type out a long name. In the Windows environment, type the first few letters of the function you want and press Ctrl-k. Make your selection from the pop-up list that appears.

Now, let's start exploring!

Numerics

Evaluate the product of the two integers 127 and 9721:

```
127 × 9721
```

```
1 234 567
```

A more complicated expression:

```
2 + (3 + 5) 8 / 7 - (2 - Pi) ^2 / (81 - 4) + 7 / (14 / 3 - 6 / 3) - 2
```

$$\frac{659}{56} - \frac{1}{77} (2 - \pi)^2$$

Note that the answer is left partially evaluated. To convert it to a decimal value, wrap N[ ] around it. You can use N[%]. % always refers to the most recently evaluated Mathematica expression, wherever it may actually be, and %% is the next-to last most recently evaluated expression, and so on. So, N[%] means "give the numerical equivalent of the most recently evaluated expression."

```
N[%]
```

```
11.7509
```

Complex Numbers As an example, simplify 4 + i -(3 - 5i)

```
4 + I - (3 - 5 I)
```

```
1 + 6 ⅈ
```

Now for some division:

```
(4 + I) / (3 - 5 I)
```

$$\frac{7}{34} + \frac{23\,i}{34}$$

Representation of Numbers Let's get Pi to oh, 800 digits:

```
N[Pi, 801]
```

```
3.14159265358979323846264338327950288419716939937510582097494459230̇7⋱
 8164062862089986280348253421170679821480865132823066470938446095505⋱
 8223172535940812848111745028410270193852110555964462294895493038196⋱
 4428810975665933446128475648233786783165271201909145648566923460348⋱
 6104543266482133936072602491412737245870066063155881748815209209628⋱
 2925409171536436789259036001133053054882046652138414695194151160943⋱
 3057270365759591953092186117381932611793105118548074462379962749567⋱
 3518857527248912279381830119491298336733624406566430860213949463952⋱
 2473719070217986094370277053921717629317675238467481846766940513200⋱
 0568127145263560827785771342757789609173637178721468440901224953430⋱
 1465495853710507922796892589235420199561121290219608640344181598136⋱
 297747713099605187072113499999983729780499510597317328160963186 0
```

(Is that really 800 digits?)

Matrix Operations

In the following, we show how to invert, evaluate the determinant, transpose, multiply and compute the eigenvalues of a matrix.
First, define a matrix A that we can work with:

```
A = {{1, 0, -5}, {7, 3, 0.6}, {-0.9, 11, 0.725}}
```

```
{{1, 0, -5}, {7, 3, 0.6}, {-0.9, 11, 0.725}}
```

```
MatrixForm[A]
```

$$\begin{pmatrix} 1 & 0 & -5 \\ 7 & 3 & 0.6 \\ -0.9 & 11 & 0.725 \end{pmatrix}$$

This looks much prettier. The determinant is:

```
Determinant[A]
```

```
Determinant[{{1, 0, -5}, {7, 3, 0.6}, {-0.9, 11, 0.725}}]
```

Fooled you - this is one time that we have a shortcut. The function to compute the determinant is named Det:

```
Det[A]
```

```
-402.925
```

The n by n identity matrix is Identity[n], so for n = 3,

```
IdentityMatrix[3] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Except for roundoff errors, A multiplied by it's inverse should return the identity matrix:

```
A.Inverse[A] // MatrixForm
```

$$\begin{pmatrix} 1. & 0. & 6.93889\times 10^{-18} \\ 0. & 1. & 9.54098\times 10^{-18} \\ 0. & -1.38778\times 10^{-17} & 1. \end{pmatrix}$$

Most of these entries are so close to zero that we can round. The Chop function is used for this purpose:

```
Chop[%] // MatrixForm
```

$$\begin{pmatrix} 1. & 0 & 0 \\ 0 & 1. & 0 \\ 0 & 0 & 1. \end{pmatrix}$$

The transpose of a matrix is obtained by interchanging the rows and columns of a matrix, so a[i,j] = a[j,i]:

```
Transpose[A]
```

```
{{1, 7, -0.9}, {0, 3, 11}, {-5, 0.6, 0.725}}
```

```
Eigenvalues[A]
```

```
{5.50702 + 5.80866 i, 5.50702 - 5.80866 i, -6.28904}
```

```
Eigenvectors[A]
```

```
{{-0.298534 + 0.384751 i, 0.286874 + 0.409611 i, 0.716077 + 0. i},
 {-0.298534 - 0.384751 i, 0.286874 - 0.409611 i, 0.716077 + 0. i},
 {0.510054, -0.432392, 0.74356}}
```

Symbolic Computations

Mathematica can work with symbols. Let us multiply two polynomials together:

```
(x - 1) (x^4 + x^3 + x^2 + x + 1)
```

$(-1+x)\left(1+x+x^2+x^3+x^4\right)$

Again, Mathematica leaves the result unevaluated, since this may be what the user wants. To force the expansion, use Expand:

```
Expand[%]
```

$-1+x^5$

Factor recovers the original expression.

```
Factor[%]
```

$(-1+x)\left(1+x+x^2+x^3+x^4\right)$

The next command performs the polynomial division (x^5 - 1)/(x - 1), which should be x^4 + x^3 + x^2 + x + 1:

```
(x^5 - 1) / (x - 1)
```

$$\frac{-1+x^5}{-1+x}$$

```
Expand[%]
```

$$-\frac{1}{-1+x}+\frac{x^5}{-1+x}$$

Hmmmm ... not quite the form we wanted, although still correct. The function PolynomialQuotient is intended for situations such as this.

```
PolynomialQuotient[(x^5 - 1), (x - 1), x]
```

$1+x+x^2+x^3+x^4$

For fun, let's take the product of (a + b i) to the tenth power. We can use the product operator. Ask for help on the syntax:

```
?? Product
```

```
Product[f, {i, imax}] evaluates the product of the expressions f as evaluated
  for each i 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 product over multiple indices.

Attributes[Product] = {HoldAll, Protected, ReadProtected}
```

```
Product[a + b I, {i, 10}]
```

$(a + ⅈ b)^{10}$

```
Expand[%]
```

$a^{10} + 10\,ⅈ\,a^9 b - 45\,a^8 b^2 - 120\,ⅈ\,a^7 b^3 + 210\,a^6 b^4 + 252\,ⅈ\,a^5 b^5 - 210\,a^4 b^6 - 120\,ⅈ\,a^3 b^7 + 45\,a^2 b^8 + 10\,ⅈ\,a\,b^9 - b^{10}$

Can Mathematica recover the original expression? Let's see ...

```
Factor[%]
```

$(a + ⅈ b)^{10}$

Sequences and Series

It is said that Gauss' father made him compute long sums as punishment. One such problem was to compute the sum 1 + 2 + 3 + ... + 100. Gauss discovered a neat trick for this. We can discover this too by looking at a few patterns - this is one of the powers of Mathematica! For this problem, let's use the Sum function:

$$\sum_{i=1}^{100} i$$

```
5050
```

Alternatively, we can type out the command:

```
Sum[i, {i, 100}]
```

```
5050
```

Generate a sequence of partial sums by using the Table command. We will just get the first 10 sums:

```
Table[Sum[i, {i, j}], {j, 1, 10}]
```

```
{1, 3, 6, 10, 15, 21, 28, 36, 45, 55}
```

You should be able to see the pattern now that would allow you to quickly evaluate the limit of such series.
Now, what is the series 1 + 1/x + 1/x^2 + 1/x^3 + ... + 1/x^n? Carry it out for x = 10:

```
Sum[1 / x^i, {i, 0, 10}]
```

$$1 + \frac{1}{x^{10}} + \frac{1}{x^9} + \frac{1}{x^8} + \frac{1}{x^7} + \frac{1}{x^6} + \frac{1}{x^5} + \frac{1}{x^4} + \frac{1}{x^3} + \frac{1}{x^2} + \frac{1}{x}$$

What is this if x = 0.1? We don't have to redo our work. Just evaluate like this:

```
% /. x → 0.1
```

1.11111×10^{10}

In the above, the /. means replace, and the x → 0.1 means use 0.1 anywhere x appears. So the above command literally means "Take the most recent expression and evaluate it using 0.1 for x." The arrow symbol is created by pressing the minus key followed by the greater-than key. The /. is created by pressing the backslash key followed by the period.
Note that products can also be easily evaluated. For example,

$$\prod_{i=0}^{10} x^i$$

x^{55}

Solution of Equations

Linear System of Equations

To warm up, let's solve the symbolic equation a x + b = y for x.

```
Solve[a x + b == y, x]
```

$$\left\{\left\{x \to -\frac{b-y}{a}\right\}\right\}$$

Note the use of the double equals sign above. This will be discussed more later, but for now, see what happens if you just use a single equals sign. Do not be surprised when it does not work.
Mathematica can also solve two equations in two unknowns. For example, the intersection of the lines 2 x - 3 y = 6 and 8 x + 4/10 y = 11 is

```
Solve[{2 x - 3 y == 6, 8 x + .4 y == 11}, {x, y}]
```

```
{{x → 1.42742, y → -1.04839}}
```

We can also use *Mathematica*'s LinearSolve function if we use matrix form:

```
A = {{2, -3}, {8, .4}};
b = {6, 11};
LinearSolve[A, b]
```

```
{1.42742, -1.04839}
```

In this case we could just as well have solved using the matrix inverse:

```
Inverse[A].b
```

```
{1.42742, -1.04839}
```

Check the solution. We require the product of A and the vector above to give the right hand side b, which it does:

```
A.%
```

```
{6., 11.}
```

```
% == b
```

```
True
```

```
Clear[b]
{x, y, z}.{a, b, c}
```

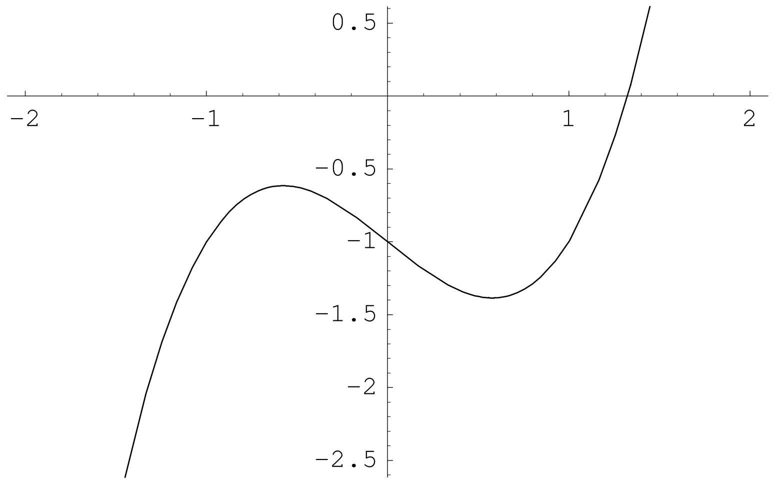
```
a x + b y + c z
```

Other vector operations are available, and more complex functions are in the package Vector-Analysis. See the *Mathematica* book or load the package for more information.

Solution of Nonlinear Equations

We first plot and then find the zeros of the function f(x) = x^3-x-1

```
Plot[x^3 - x - 1, {x, -2, 2}]
```



```
Solve[x^3 - x - 1 == 0, x]
```

$$\left\{\left\{x \to \frac{1}{3}\left(\frac{27}{2}-\frac{3\sqrt{69}}{2}\right)^{1/3}+\frac{\left(\frac{1}{2}\left(9+\sqrt{69}\right)\right)^{1/3}}{3^{2/3}}\right\},\right.$$

$$\left\{x \to -\frac{1}{6}\left(1+i\sqrt{3}\right)\left(\frac{27}{2}-\frac{3\sqrt{69}}{2}\right)^{1/3}-\frac{\left(1-i\sqrt{3}\right)\left(\frac{1}{2}\left(9+\sqrt{69}\right)\right)^{1/3}}{2\ 3^{2/3}}\right\},$$

$$\left.\left\{x \to -\frac{1}{6}\left(1-i\sqrt{3}\right)\left(\frac{27}{2}-\frac{3\sqrt{69}}{2}\right)^{1/3}-\frac{\left(1+i\sqrt{3}\right)\left(\frac{1}{2}\left(9+\sqrt{69}\right)\right)^{1/3}}{2\ 3^{2/3}}\right\}\right\}$$

```
FindRoot[x^3 - x - 1, {x, 1}]
```

```
{x → 1.32472}
```

Nonlinear Systems of Equations

The Newton-Raphson method is used to approximate the solution of the system below. We use the intial guess x = 0, y = 1 and x = 2:

f[x,y,z] = sin(x y) - 3x/z + log(z^2)

g[x,y,z] = tan(x^2y)+ 8 x y - 11 z

h[x,y,z] = cos(y) + x+sqrt(z^y)

```
Clear[f, g, h]
f[x_, y_, z_] = Sin[x y] - 3 x / z + Log[z^2];
g[x_, y_, z_] = Tan[x^2 y] + 8 x y - 11 z;
h[x_, y_, z_] = Cos[y] + Pi x + Sqrt[z^ y];

solution = FindRoot[{f[x, y, z] == 0, g[x, y, z] == 0 , h[x, y, z] == 0},
  {x, 0}, {y, 1}, {z, 2}, MaxIterations → 30]
```

```
{x → 0.0592735, y → 17.0133, z → 0.738848}
```

Check the results by substitution:

```
{f[x, y, z] , g[x, y, z] , h[x, y, z] } /. %
```

$\{-1.67967\times10^{-8}, -4.81571\times10^{-8}, 7.99074\times10^{-8}\}$

This is zero as far as the default machine precision is concerned.

Limit of a Function

What is the limit of the function sin(x)/x as x approaches zero? There are many ways to answer this. One could print out a table of values of {x,sin(x)/x} as we allow x to approach zero; plot the function, or we could just take the limit directly.

```
Print["x\tSin[x]/x"]
Do[
  Print[x, "\t", Sin[x] / x], {x, 0.1, 0.001, -.01}]
```

```
x    Sin[x]/x

0.1    0.998334

0.09    0.998651

0.08    0.998934

0.07    0.999184

0.06    0.9994

0.05    0.999583

0.04    0.999733

0.03    0.99985

0.02    0.999933

0.01    0.999983
```

```
Limit[Sin[x] / x, x → 0]
```

```
1
```

```
D[Log[x], x]
```

$\frac{1}{x}$

Mathematica also understands the prime symbol, so we could have written

```
Log'[x]
```

$\frac{1}{x}$

What is the derivative of the function f(x) = a^x with respect to x?

```
D[a^x, x]
```

a^x Log[a]

Partial derivatives are handled in the same way, but care must be taken in specification of the independent variables. Here is an ugly function:

$\partial_y \left(\text{Log}\left[\frac{x}{y}\right] \text{ArcTan}\left[\sqrt{1 + x\, y}\,\right]\right)$

$-\frac{\text{ArcTan}\left[\sqrt{1+x\,y}\right]}{y} + \frac{x\,\text{Log}\left[\frac{x}{y}\right]}{2\sqrt{1+x\,y}\,(2+x\,y)}$

```
D[Log[x / y] ArcTan[Sqrt[1 + x y]], y]
```

$-\frac{\text{ArcTan}\left[\sqrt{1+x\,y}\right]}{y} + \frac{x\,\text{Log}\left[\frac{x}{y}\right]}{2\sqrt{1+x\,y}\,(2+x\,y)}$

Maximum of a Function Which number is larger: $e^{\pi} e^{\pi}$ or $\pi^{e}\ \pi^{e}$? (Reference: *The Mathematical Gazette*, 1991). To determine which of the values is larger, we could just evaluate them and take the Max:

```
Max[{Exp[Pi], Pi^E}]
```

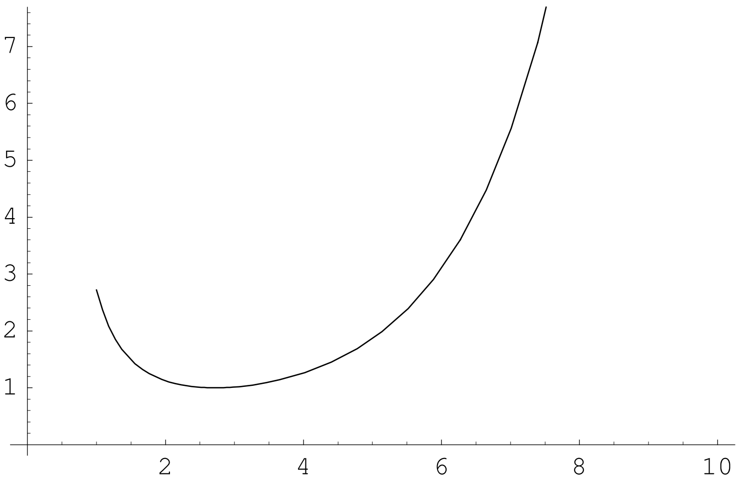
e^{π}

Alternatively, we could define the function y = exp(x)/x^e. If x = 1, then y = e; if x = e, then y = 1; if x = 10, then y = 42.1369. Clearly y has at least a local minimum on (1,10).

```
Clear[y]
```

$$y[x\_] := \frac{e^x}{x^e};$$

```
Plot[y[x], {x, 1, 10}]
```



```
FindMinimum[y[x], {x, 2}]

{1., {x → 2.71828}}
```

This is x=e.Then exp(x)/x^e>1 for all positive x,except for x=e.Hence e^pi>pi^e,since pi is not equal to e.

Taylor Series Expansions

Taylor Series Expansions are obtained with the Series function. To expand the (assumed n + 1 times differentiable) function f about the point a, with integer n number of terms, the command is Series[f[x],{x,a,n}]. For example, let us take n = 5:

```
Clear[f]
Series[f[x], {x, a, 5}]
```

$$f[a] + f'[a]\,(x-a) + \frac{1}{2} f''[a]\,(x-a)^2 + \frac{1}{6} f^{(3)}[a]\,(x-a)^3 + \frac{1}{24} f^{(4)}[a]\,(x-a)^4 + \frac{1}{120} f^{(5)}[a]\,(x-a)^5 + O[x-a]^6$$

Now obtain the Taylor Series expansion for Log[x], centered about the point x = 1 (so we are really doing a Maclaurin expansion), and take four terms.

```
Series[Log[x], {x, 1, 4}]
```

$$(x-1)-\frac{1}{2}(x-1)^2+\frac{1}{3}(x-1)^3-\frac{1}{4}(x-1)^4+O[x-1]^5$$

Note the O[x-1]^5 term at the end of the result. This represents the truncation error of the series. To do any calculations with the result, we have to drop this term. This is done by way of the Normal function:

```
Normal[%]
```

$$-1-\frac{1}{2}(-1+x)^2+\frac{1}{3}(-1+x)^3-\frac{1}{4}(-1+x)^4+x$$

It is interesting to use other intervals and different degree polynomials in the Taylor series expansion to see the change in accuracy. One could also use different expansion points. Since log(x) is not a polynomial, no approximation will match it exactly on the entire real line, but it is possible to match it as closely as desired on restricted intervals.
Problem: A sharp high-school student showed me the following formula he had read in a textbook, and asked how it could possibly be correct: how was it possible to expand something in terms of fractional powers? Use *Mathematica* to answer the student's question.

$$(1+\theta)^{1/3}\cong 1+\frac{\theta}{3}-\frac{\theta^2}{9}+\frac{5\theta^3}{81}$$

Answer: The right-hand side must be the Maclaurin series expansion of the function. Check this with Series. The result:

```
Series[(1 + θ) ^ (1 / 3), {θ, 0, 5}]
```

$$1+\frac{\theta}{3}-\frac{\theta^2}{9}+\frac{5\theta^3}{81}-\frac{10\theta^4}{243}+\frac{22\theta^5}{729}+O[\theta]^6$$

What is the nth term?

Integration

Mathematica is capable of computing definite and indefinite integrals, both analytically and symbolically. The integration routines may require a great deal of memory depending on the complexity of the function. The syntax for integration is

```
?? Integrate
```

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

Attributes[Integrate] = {Protected, ReadProtected}

Options[Integrate] =
 {Assumptions → {}, GenerateConditions → Automatic, PrincipalValue → False}
```

Now, how does Numerical Integration work??

```
?? NIntegrate
```

```
NIntegrate[f, {x, xmin, xmax}] gives a numerical approximation
  to the integral of f with respect to x from xmin to xmax.

Attributes[NIntegrate] = {HoldAll, Protected}

Options[NIntegrate] = {AccuracyGoal → ∞, Compiled → True, GaussPoints → Automatic,
  MaxPoints → Automatic, MaxRecursion → 6, Method → Automatic, MinRecursion → 0,
  PrecisionGoal → Automatic, SingularityDepth → 4, WorkingPrecision → 16}
```

Now, compute the integral of $\int \frac{2\,x\,\mathrm{Cos}[x^2]}{\mathrm{Sin}[x^2]}\,dx$

```
Integrate[2 x Cos[x^2] / Sin[x^2], x]
```

$\mathrm{Log}[\mathrm{Sin}[x^2]]$

```
Integrate[Exp[-x^2 / (2 tau)], x]
```

$\sqrt{\frac{\pi}{2}}\,\sqrt{\mathrm{tau}}\,\mathrm{Erf}\left[\frac{x}{\sqrt{2}\,\sqrt{\mathrm{tau}}}\right]$

Multiple Integration Double,triple and higher dimension integrals are easily handled.The outer limits appear first in the argument list. The syntax is

```
Integrate[integrand,{var1,lowerlimit,upperlimit},
             {var2,lowerlimit,upperlimit},...] or
  NIntegrate[integrand,{var1,lowerlimit,upperlimit},
             {var2,lowerlimit,upperlimit},...]
```

So the integral:

$$\int_{x=0}^{1}\int_{y=0}^{1}\sin(x+y)\,dy\,dx$$

is evaluated as:

```
Integrate[Sin[x + y], {x, 0, 1}, {y, 0, 1}]
```

$$\text{Sin}[1]-\text{Cos}[1]\,\text{Sin}[1]+2\,\text{Sin}\left[\frac{1}{2}\right]^2\,\text{Sin}[1]$$

```
N[%]
```

```
0.773645
```

With a more complicated integrand and/or more complicated limits, integration becomes more difficult. Attempting the integral of tan(x y/(x^2 + y^2)) as x and y run from pi/4 to pi/3 gives:

```
Integrate[Tan[x y / (x^2 + y^2)], {x, Pi / 4, Pi / 3}, {y, Pi / 4, Pi / 3}]
```

$$\int_{\frac{\pi}{4}}^{\frac{\pi}{3}}\int_{\frac{\pi}{4}}^{\frac{\pi}{3}}\text{Tan}\left[\frac{x\,y}{x^2+y^2}\right]dy\,dx$$

The answer above indicates that *Mathematica* was unable to find the solution of this integral. In such cases we must rely on numerical approximation. The syntax is the same except that we use NIntegrate rather than Integrate. A slick shortcut is to just use N of the preceding:

```
N[%]
```

```
0.0371428
```

It worked!

Solution of Ordinary Differential Equations

Differential equations are solved by the functions DSolve or NDSolve, and both can also handle systems of ODEs. The syntax is:

`DSolve[lhs == rhs, depvar, indvar]` *solves the ODE lhs = rhs for the dependent variable depvar in terms of the independent variable indvar*.

`NDSolve[lhs == rhs, depvar, {indvar,left, right}]` *gives a numerical approximation of the ODE lhs = rhs for the dependent variable depvar in terms of the independent variable indvar on [left,right].*

Example: Solve the ODE y''[x] + 2 y'[x] + 5 y = 0 , y[0] = 0, y'[0] = 1, both analytically and numerically. For the numerical solution, let x run from 0 to 10. Plot.

```
Clear[x, y]
DSolve[y''[x] + 2 y'[x] + 5 y[x] == 0, y[x], x]
```

$\{\{y[x] \to e^{-x} C[2] \, Cos[2x] - e^{-x} C[1] \, Sin[2x]\}\}$

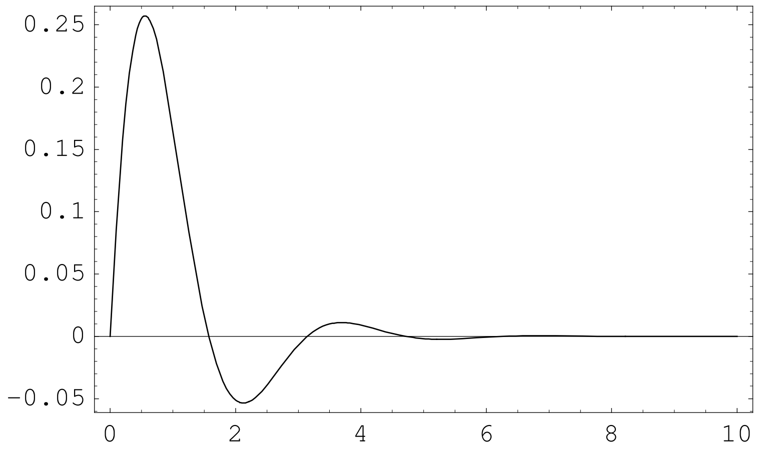
To use the initial conditions, call DSolve in this way:

```
DSolve[{y''[x] + 2 y'[x] + 5 y[x] == 0, y'[0] == 1, y[0] == 0}, y[x], x]
```

$\left\{\left\{y[x] \to \frac{1}{2} e^{-x} Sin[2x]\right\}\right\}$

Plot the solution, using a frame to make it fancy:

```
Plot[%[[1, 1, 2]], {x, 0, 10}, PlotRange → All, Frame → True]
```



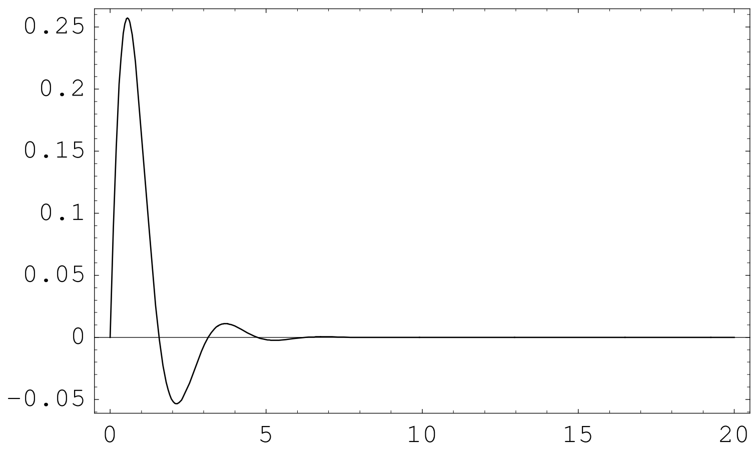
The numerical solution Notice that the only difference in the syntax between `DSolve` and `NDSolve` is in specification of the independent variable: here, we provide a domain.

```
NDSolve[{y''[x] + 2 y'[x] + 5 y[x] == 0, y'[0] == 1, y[0] == 0},
 y[x], {x, 0, 20}]
```

```
{{y[x] → InterpolatingFunction[{{0., 20.}}, <>][x]}}
```

`NDSolve` returns the solution as an InterpolatingFunction object. Use Evaluate to plot the solution:

```
Plot[Evaluate[y[x] /. %], {x, 0, 20}, PlotRange → All, Frame → True]
```



Function Interpolation and Approximation

Mathematica has several intrinsic functions to perform interpolation and approximation of data. These include capabilities to perform interpolation, regression and spline fits of data.

Function Approximation

The syntax for Lagrange Interpolation is:

```
InterpolatingPolynomial[{data_List}, var]
```

As an example, we build up a table of six {x,Exp[x]} pairs on the interval [0,4] as the data_List.

```
Table[{x, ℯ^x}, {x, 0, 4, 0.8}]
```

```
{{0, 1}, {0.8, 2.22554}, {1.6, 4.95303},
 {2.4, 11.0232}, {3.2, 24.5325}, {4., 54.5982}}
```

```
InterpolatingPolynomial[%, x]
```

```
1 + (1.53193 +
    (1.1734 + (0.599187 + (0.229477 + 0.0703085 (-3.2 + x)) (-2.4 + x))
        (-1.6 + x)) (-0.8 + x)) x
```

Regression

Regression is performed via the Fit function. The syntax is:

Fit[data_List,{1,var, ... , var^n},var]

where var is the symbol of the desired variable and n is the desired degree of fit. For the preceding {x,Exp[x]} data, a fifth-degree regression fit is constructed as

```
Fit[Table[{x, ⅇ^x}, {x, 0, 4, 0.8}], {1, x, x^2, x^3, x^4, x^5}, x]
```

$1. + 1.34637\,x - 0.449026\,x^2 + 1.07261\,x^3 - 0.332991\,x^4 + 0.0703085\,x^5$

As a shortcut, the variable list $\{1,x,x^2,x^3,x^4,x^5\}$ may be defined by use of `Table`. In fact, a convenient regression function is

```
Clear[regression]
regression[data_List, n_, var_] :=
 Fit[data, Table[var^i, {i, 0, n}], var]
```

```
regression[Table[{x, ⅇ^x}, {x, 0, 4, 0.8}], 5, x]
```

$1. + 1.34637\,x - 0.449026\,x^2 + 1.07261\,x^3 - 0.332991\,x^4 + 0.0703085\,x^5$

Of course, this matches the Interpolating polynomial of degree 5 found previously since the polynomial of degree n interpolating n+1 points is unique.

Multivariate Regression

`Fit` may also be used for multivariate functions. For example,

```
Fit[data_List,{1,x,y,x y},{x,y}]
```

will fit z[x,y] = a x + b y + c x y + d to the function defined by data_List.
Using the function 2 x + 4 y - 10, with five points in each direction on the domain 0 £ x £ 1, 0 ≤ y ≤ 1 we obtain the data as

```
RegressionData =
 Flatten[Table[{x, y, 2 x + 4 y - 10}, {x, 0, 1, 0.25},
    {y, 0, 1, 0.25}], 1]
```

```
{{0, 0, -10}, {0, 0.25, -9.}, {0, 0.5, -8.}, {0, 0.75, -7.},
 {0, 1., -6.}, {0.25, 0, -9.5}, {0.25, 0.25, -8.5}, {0.25, 0.5, -7.5},
 {0.25, 0.75, -6.5}, {0.25, 1., -5.5}, {0.5, 0, -9.},
 {0.5, 0.25, -8.}, {0.5, 0.5, -7.}, {0.5, 0.75, -6.}, {0.5, 1., -5.},
 {0.75, 0, -8.5}, {0.75, 0.25, -7.5}, {0.75, 0.5, -6.5},
 {0.75, 0.75, -5.5}, {0.75, 1., -4.5}, {1., 0, -8.},
 {1., 0.25, -7.}, {1., 0.5, -6.}, {1., 0.75, -5.}, {1., 1., -4.}}
```

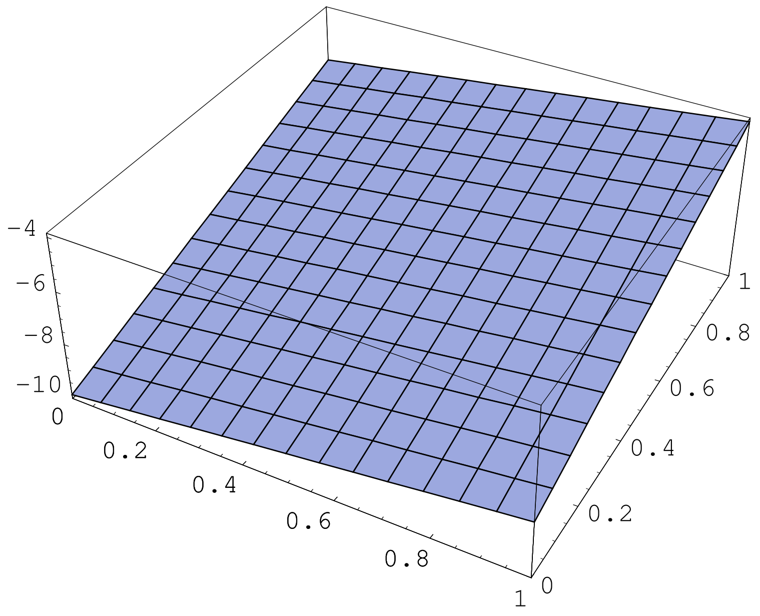
Now the regression fit is obtained as

```
Fit[RegressionData, {1, x, y, x y}, {x, y}]
```

$-10. + 2.\ x + 4.\ y - 3.10862 \times 10^{-15}\ x\ y$

which, as expected, matches the given function. The regression function may be plotted with Plot3D or Contour-Plot. The discrete RegressionData may be plotted via ListPlot3D or ListContourPlot.

```
Plot3D[%, {x, 0, 1}, {y, 0, 1}]
```



Graphics

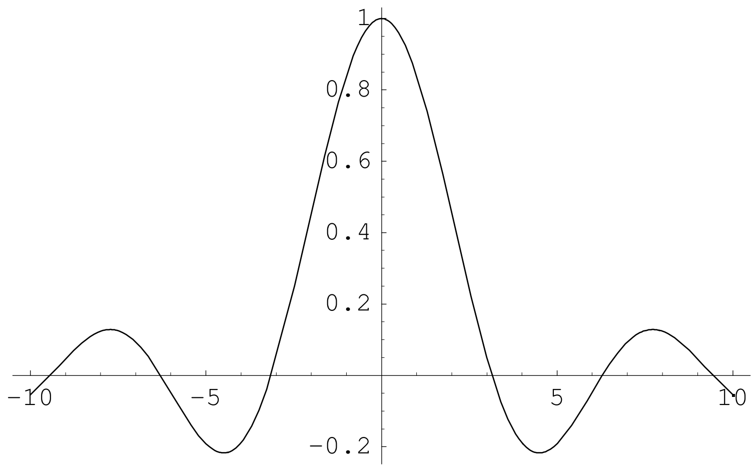
Mathematica has hundreds of built-in Graphics objects. You can create everything from a simple line plot to a stellated icosahedron. If what you want is not built in, *Mathematica*'s programming capablity means that you can easily write code to do your own graphics. There are primitives in *Mathematica* for drawing such objects as Circles, Points, Lines and Disks as well.

Plotting

`Plot[function,{var,left,right}]`
will plot the specified function of a single variable as var runs from left to right.
`Plot3D[function,{var1,left,right},{var2,left,right}]`
will plot the specified function of two variables var1, var2 on the specified domain.
`ContourPlot[function,{var1,left,right},{var2,left,right}]`
will make a contour plot of the specified function of two variables var1, var2 on the specified domain.
`ListPlot[list]`
will plot a set of points given as list. Option: PlotJoined->True will connect the points with lines. The option Prolog->PointSize[num] will use points of the specified size.
`ListPlot3D[list]`
will plot a set of 2-D data given as list.
`ListContourPlot[list]`
will make a contour plot of the tabular data given as list.

Examples

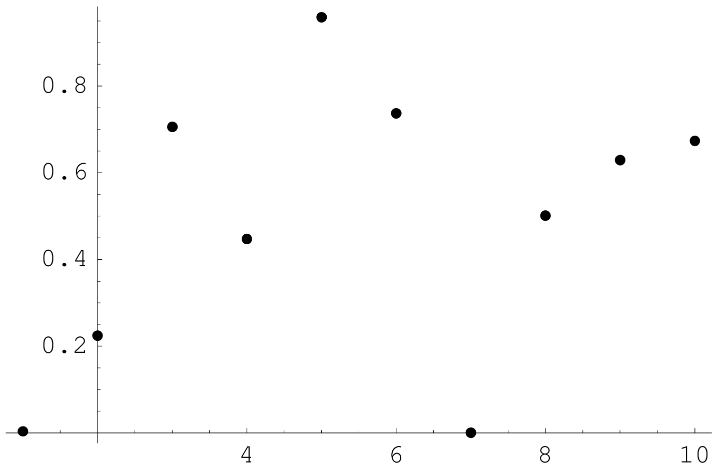
Plot$\left[\frac{\text{Sin}[x]}{x}, \{x, -10, 10\}\right]$



```
data = Table[{i, RandomReal[]}, {i, 10}]
```

```
{{1, 0.00373923}, {2, 0.224572}, {3, 0.706306},
 {4, 0.447634}, {5, 0.958863}, {6, 0.73721},
 {7, 0.000896704}, {8, 0.501372}, {9, 0.6294}, {10, 0.673787}}
```

```
dataplot = ListPlot[data, Prolog → PointSize[0.015`]]
```



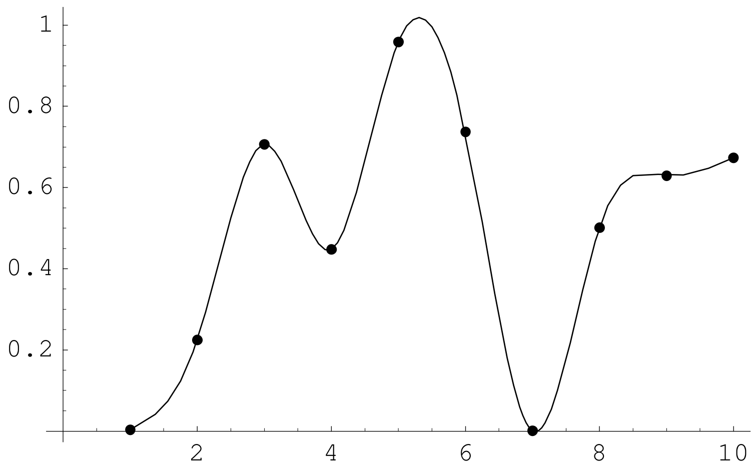
Splines

Mathematica has a package Graphics`Spline for construction of cubic, Bezier and composite Bezier curves. Load the package first and then construct and plot a cubic spline through the random points defined below:

```
Needs["Splines`"]
```

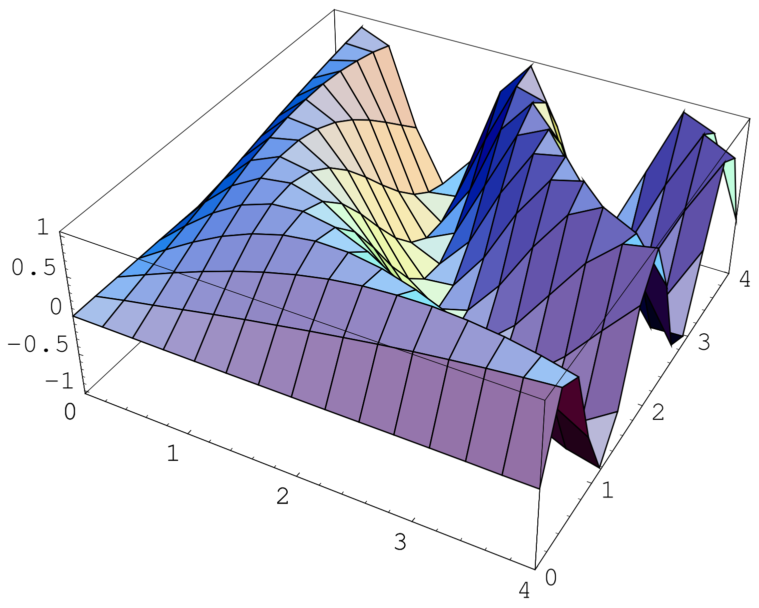
This shows the spline interpolation and the data points:

```
Show[dataplot, Graphics[Spline[data, Cubic]]]
```

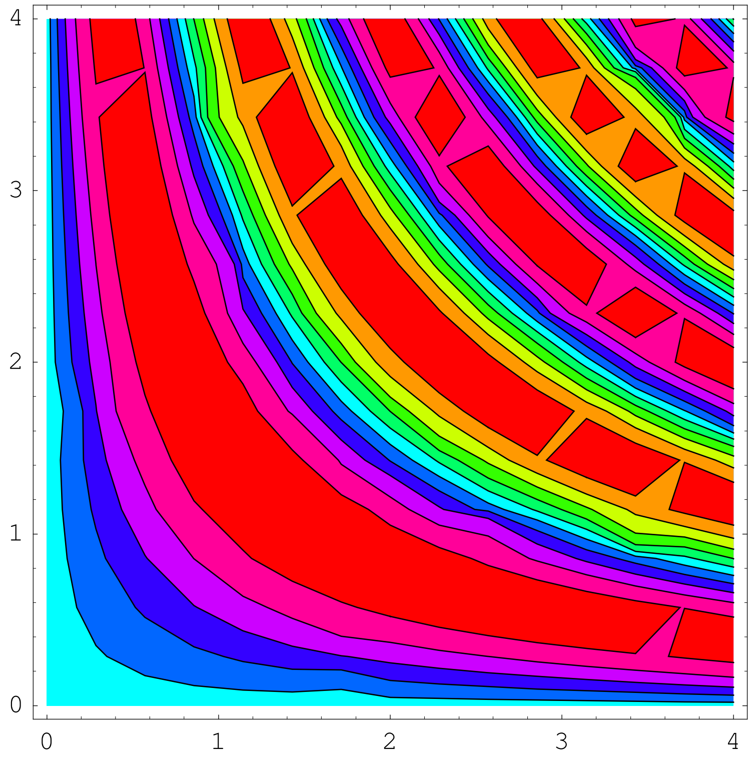


Surface Plots

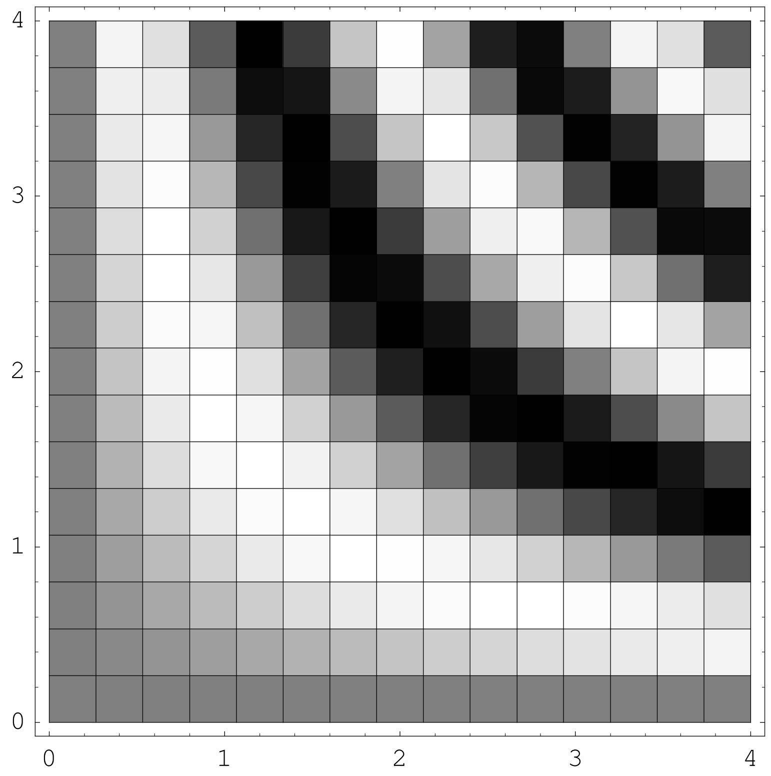
```
Plot3D[Sin[x y], {x, 0, 4}, {y, 0, 4}]
```



```
ContourPlot[Sin[x y], {x, 0, 4}, {y, 0, 4}, ColorFunction → Hue]
```



```
DensityPlot[Sin[x y], {x, 0, 4}, {y, 0, 4}]
```

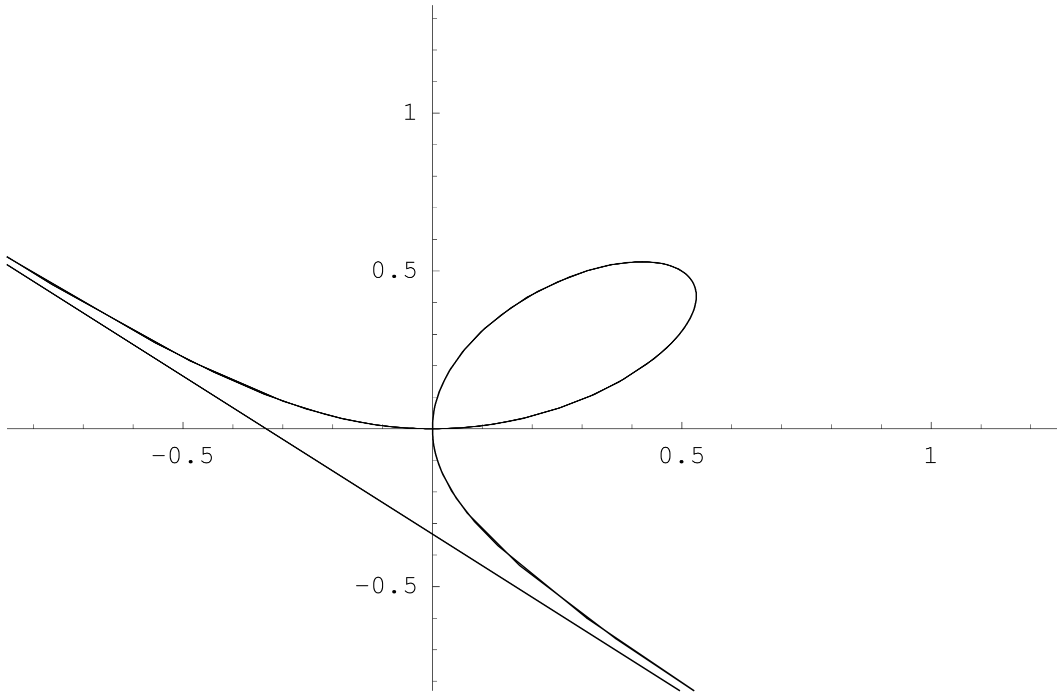


Parametric Plots

`Mathematica` can perform parametric plots of single- or multi-variate functions with the commands ParametricPlot, ParametricPlot3D. Examples follow.

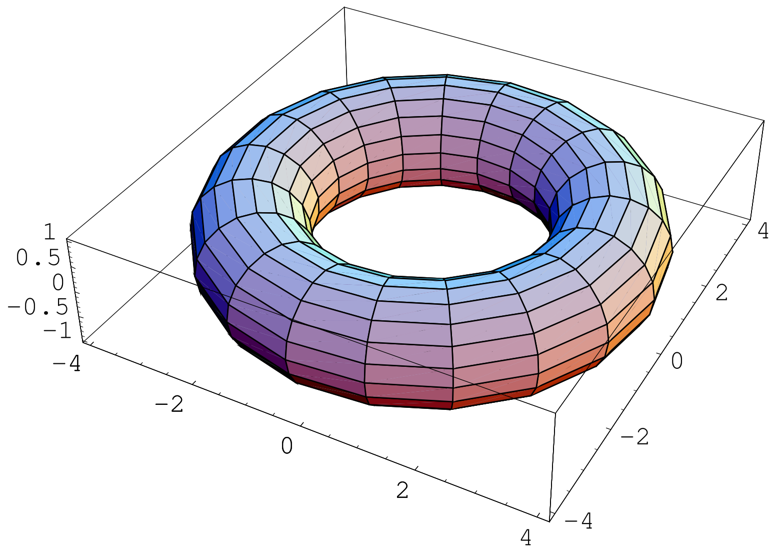
The Folium of Descartes is defined by the equation
f(x,y) = x^3 + y^3 - 6 x y = 0. Since y cannot be written explicitly as a function of x, perhaps parametric plotting will work.
Set x = r Cos[t], y = r Sin[t]. (Note tha*t Mathemati*ca could perform these transformations for us!) Then the function r[t] is as shown below.

```
Clear[r]
r[t_] = Sin[t] Cos[t] / (Sin[t]^3 + Cos[t]^3);
ParametricPlot[{r[t] Cos[t], r[t] Sin[t]}, {t, 0, 2 π}]
```



The following example is from the Mathematica book. The torus is created by varying t to produce a circle. The circle is rotated about the z axis by varying u.

```
ParametricPlot3D[{Cos[t] (3 + Cos[u]), Sin[t] (3 + Cos[u]), Sin[u]},
 {t, 0, 2 π}, {u, 0, 2 π}]
```

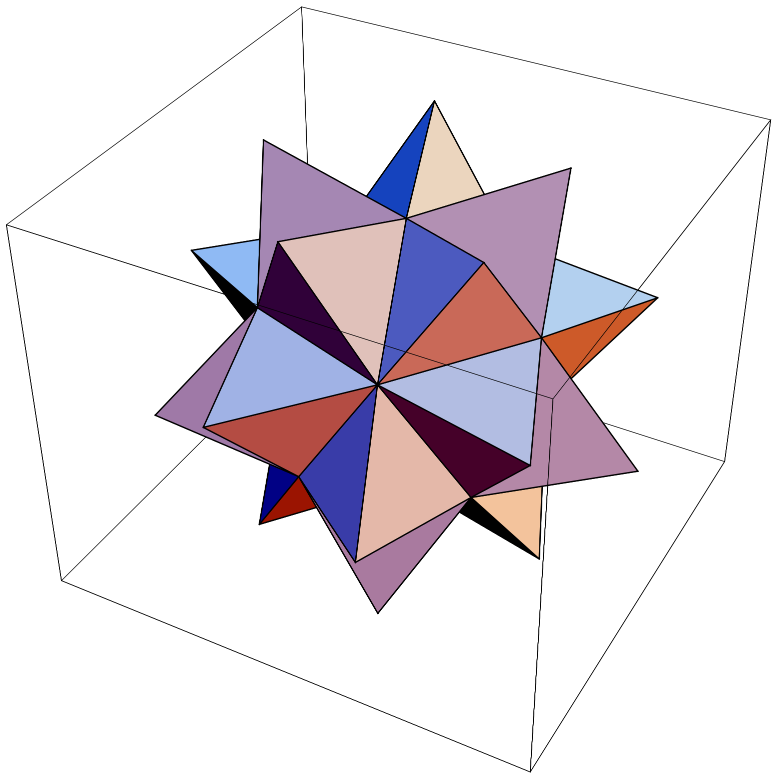


Mathematica has several built-in graphics objects. Load the Graphics`Master package so that all of the example functions will be available:

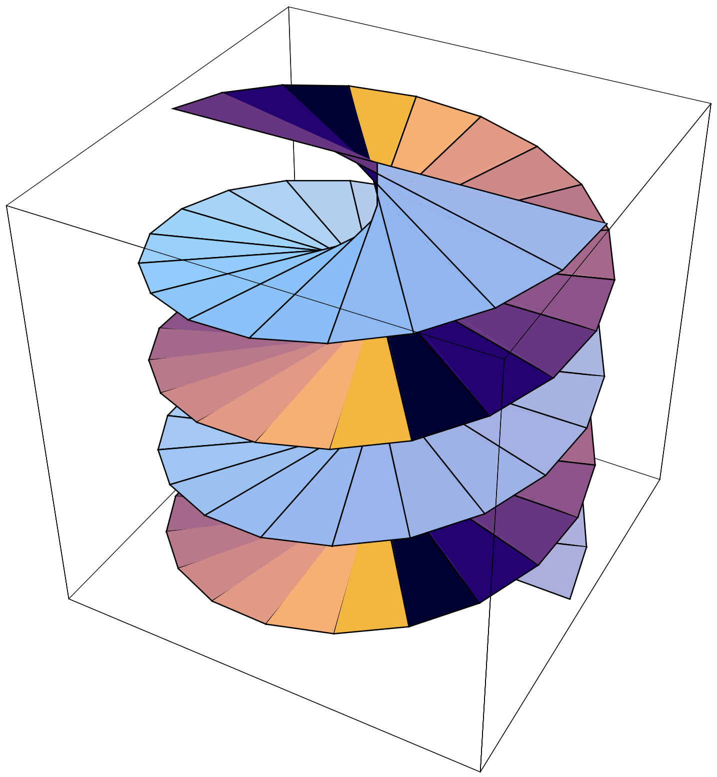
```
Needs["Graphics`Master`"]
```

We shall stellate an icosahedron (put star-like points on the faces of the icosahedron):

```
Show[Graphics3D[Stellate[Icosahedron[]]]]
```



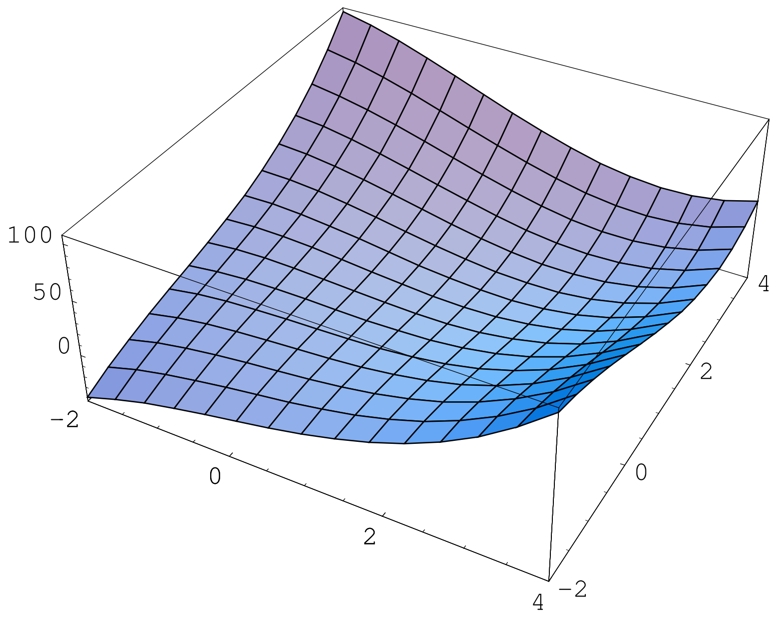
```
Show[Graphics3D[DoubleHelix[]]]
```



Using Show to Combine Graphics Objects

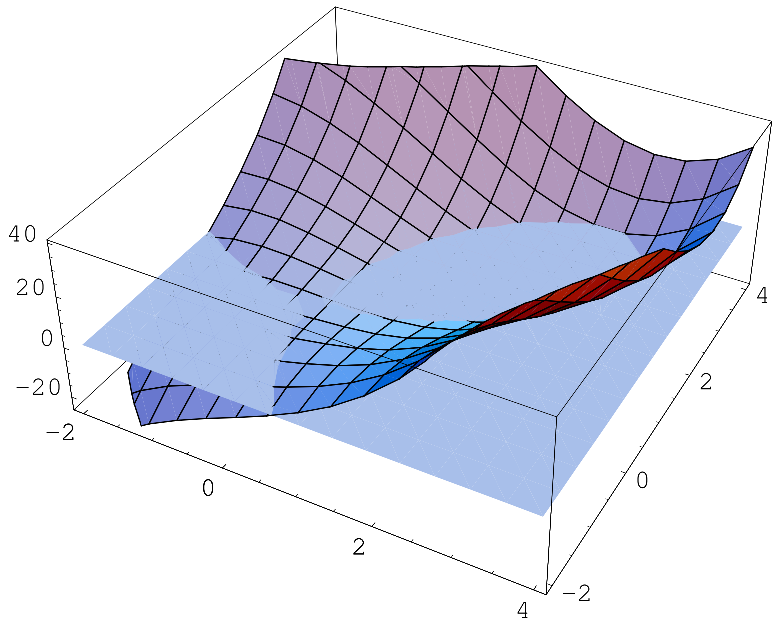
The Show command can be used to display a number of graphics objects on the same set of axes. As an example, to plot the intersection of the curvex^3 + y^3 - 6 x y and the plane z[x,y] = 0 as x and y run from -2 to 4, we can execute each plot and use Show to combine them (in plot2, the optional command DisplayFunction->Identity is used to suppress graphical output, since this graph is not very interesting.)

```
plot1 = Plot3D[x^3 + y^3 - 6 x y, {x, -2, 4}, {y, -2, 4}]
```



```
plot2 = Plot3D[0, {x, -2, 4}, {y, -2, 4},
   Mesh → False, DisplayFunction → Identity];

Show[plot1, plot2]
```

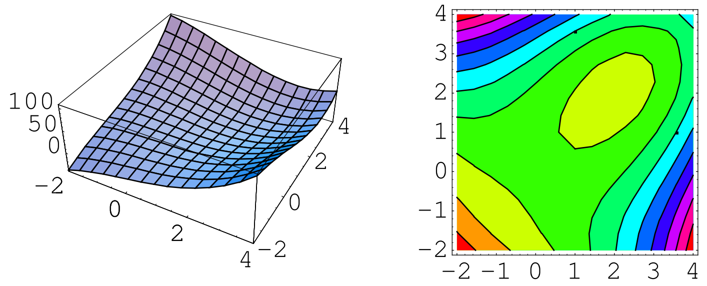


Using GraphicsArray to Combine Graphics Objects

Another nice feature in v. 2.0 is `GraphicsArray`, with which one can show an array of graphics objects side by side. As an example, surface and contour plots of the function x^3 + y^3 - 6 x y are shown below. The optional command `DisplayFunction->Identity` is used to suppress the plot (to save room.)

```
plot3 = ContourPlot[x^3 + y^3 - 6 x y, {x, -2, 4}, {y, -2, 4},
   DisplayFunction → Identity, ColorFunction → Hue];

Show[GraphicsGrid[{{plot1, plot3}}]]
```



The Solution Set of an Implicit Equation

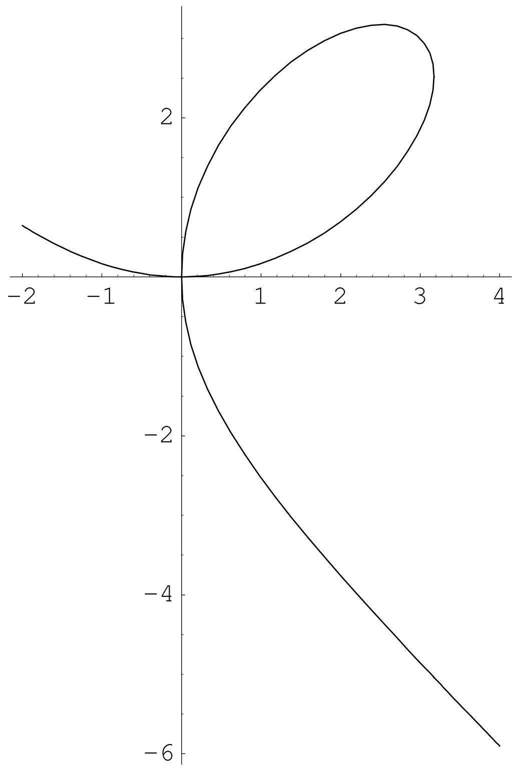
Reference: *Mathematica* Technical Report, "Guide to Standard *Mathematica* Packages", Wolfram Research, p. 97.
Implicit plots are generated from the package Graphics`ImplicitPlot`. The syntax is

```
ImplicitPlot[equation, {x, xmin, xmax}]
```

As an example, the solution of x^3 + y^3 - 6 x y = 0 will be plotted (this is the Folium of Descartes). First, load in the package:

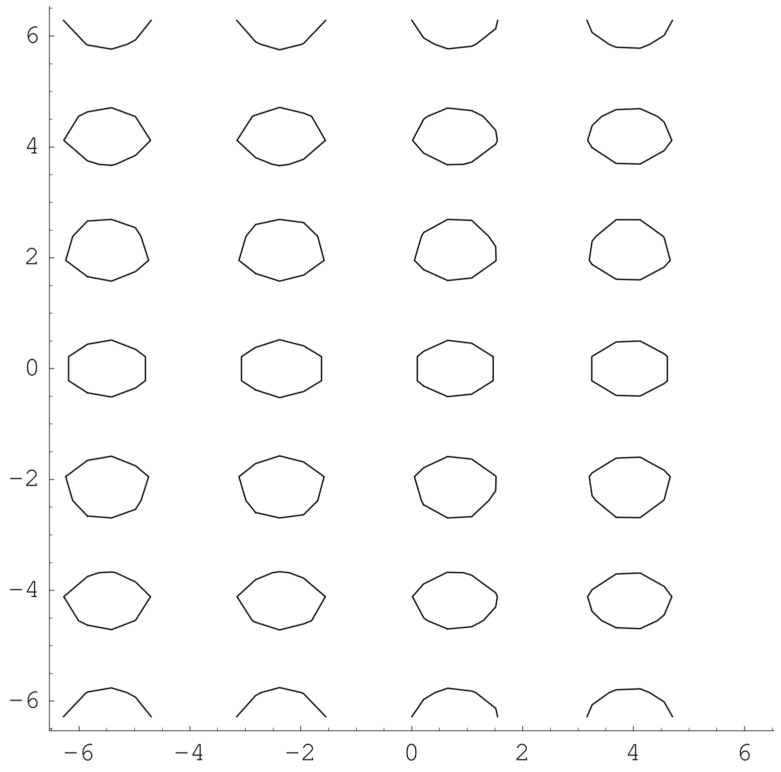
```
Needs["Graphics`ImplicitPlot`"]
```

```
ImplicitPlot[x^3 + y^3 - 6 x y == 0, {x, -2, 4}];
```



Another interesting example, from the "Guide to Standard `Mathematica` Packages", p. 98:

```
ContourPlot[Sin[2 x] + Cos[3 y] == 1,
  {x, -2 π, 2 π}, {y, -2 π, 2 π}, PlotPoints → 30];
```



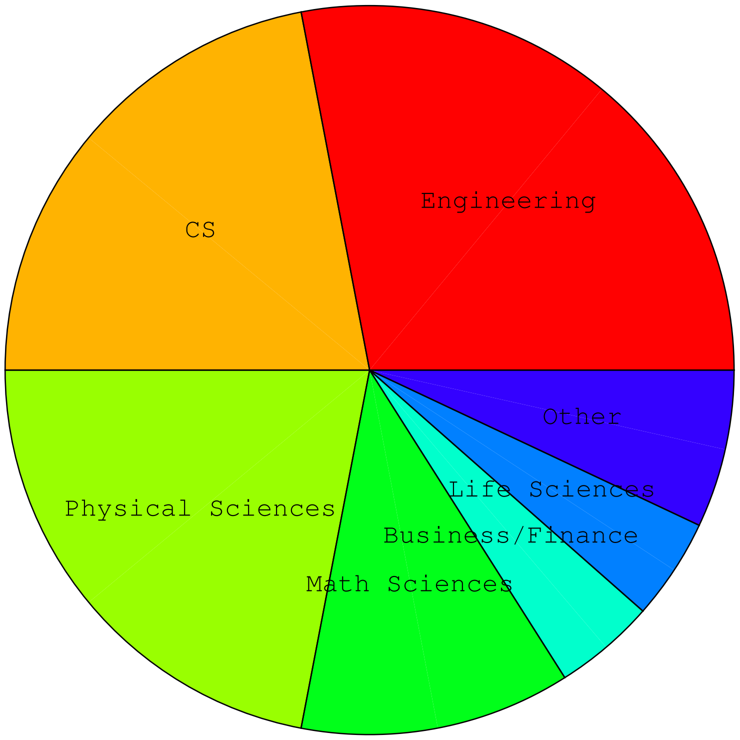
Pie and Bar Charts

The package Graphics`Graphics includes routines for creating pie and bar charts of data. As an example, charts of *Mathematica* user data are presented:

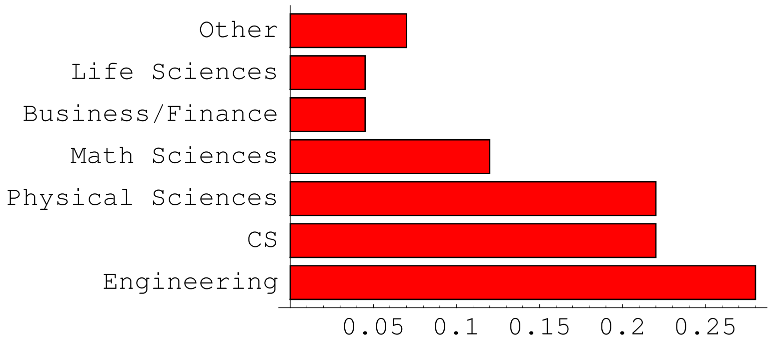
```
Needs["BarCharts`"]; Needs["Histograms`"]; Needs["PieCharts`"]
```

```
UserData =
  {{.28, "Engineering"}, {.22, "CS"}, {.22, "Physical Sciences"},
   {.12, "Math Sciences"}, {.045, "Business/Finance"},
   {.045, "Life Sciences"}, {.07, "Other"}};
```

```
PieChart[UserData]
```



```
BarChart[UserData, BarOrientation → Horizontal]
```



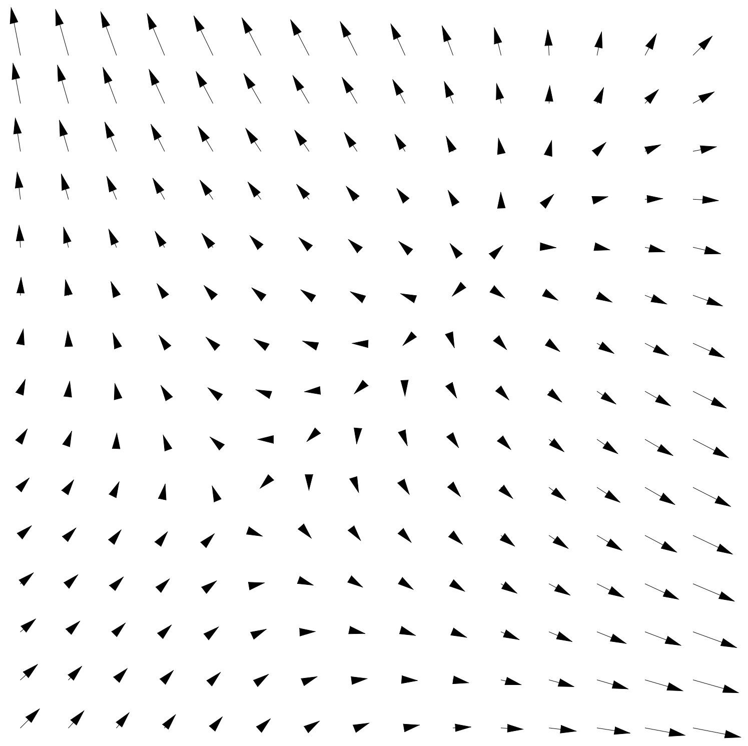
Vector Fields

The package Graphics`PlotField contains capabilities for plotting vector, gradient, Polya and Hamiltonian fields. The package Graphics`PlotField3D does the same for 3D objects.

Reference: Technical Report, "Guide to Standard *Mathematica* Packages", Wolfram Research, p. 114.

```
Needs["VectorFieldPlots`"]
```

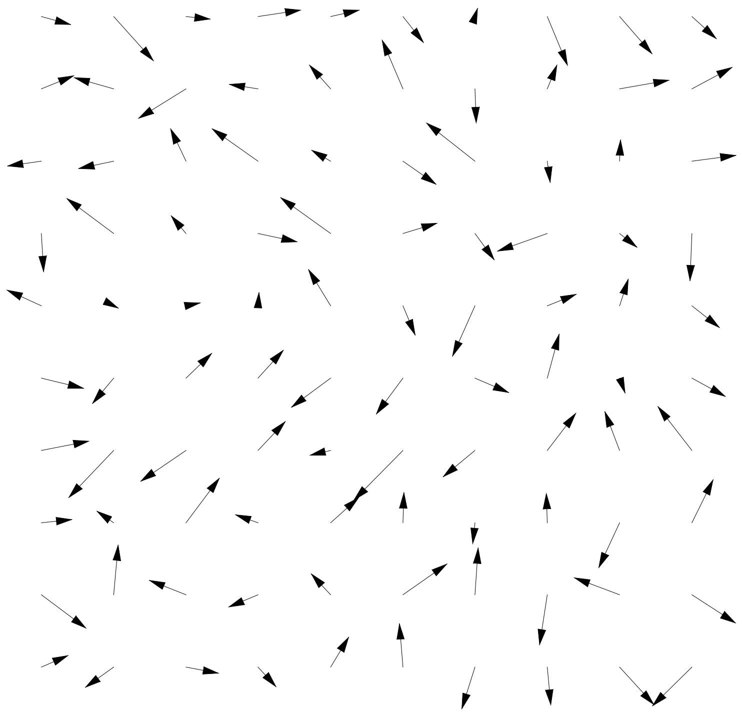
```
(Needs["VectorFieldPlots`"]; VectorFieldPlots`GradientFieldPlot[
   x^3 + y^3 - 6 x y, {x, -2, 4}, {y, -2, 4}]);
```



If your data are tabular rather than described by a function, use ListPlotVectorField instead. The following example is from the Technical Report, p. 115:

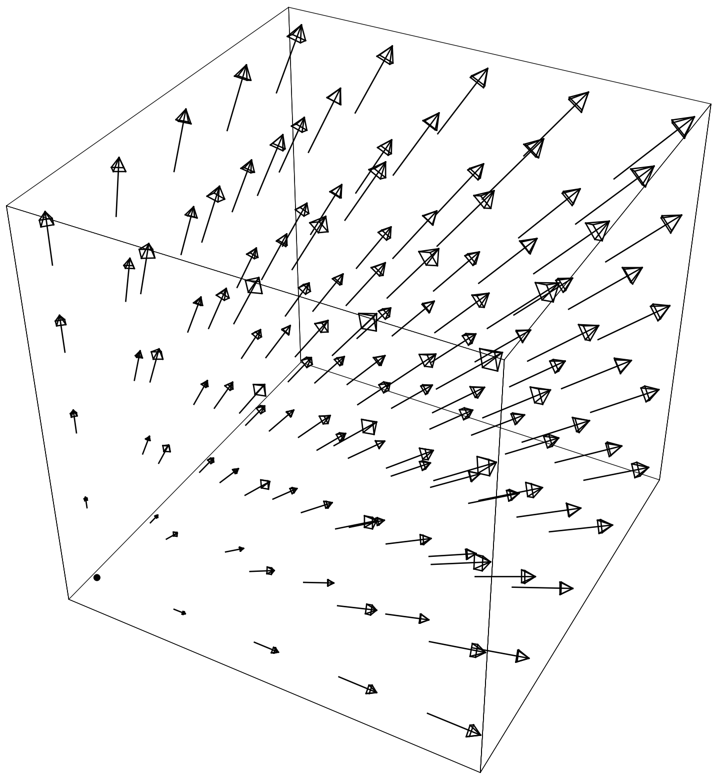
```
varray = Table[RandomReal[{-0.7`, 0.7`}, 2], {i, 10}, {j, 10}];
```

```
(Needs["VectorFieldPlots`"];
  VectorFieldPlots`ListVectorFieldPlot[varray]);
```



```
Needs["VectorFieldPlots`"]
```

```
(Needs["VectorFieldPlots`"];
  VectorFieldPlots`VectorFieldPlot3D[{x, y, z}, {x, 0, 2},
   {y, 0, 2}, {z, 0, 2}, PlotPoints → 5, VectorHeads → True]);
```



Introductory Programming in Mathematica

If *Mathematica* does not have the function you need, you can always create your own. In addition, *Mathematica* is a high-level programming language. In the following section, a brief introduction to programming in *Mathematica* will be presented.

Sorting Numbers

Use the `Sort` command. The argument is a list of numbers. To reverse the order, use the command `Reverse`. The minimum of the list is obtained with `Min`.

```
Sort[{1, 5, 10, -19, 41}]
```

```
{-19, 1, 5, 10, 41}
```

```
Reverse[%]
```

```
{41, 10, 5, 1, -19}
```

```
Min[{1, 5, 10, -19, 41}]

-19
```

Functions

The definition of the function f(x) = x^3 - x - 1 in *Mathematica* is written as `f[x_] = x^3 - x - 1.`
This is equivalent to DEF FNf(x) = x^3 - x - 1 in BASIC.
The underscore _ is required for pattern matching. This means that any argument of f will match. As a bad example, see what happens when the underscore is omitted in the definition of f and an attempt is made to evaluate f[1]:

```
Clear[f]
f[x] = x^3 - x - 1
f[1]

-1 - x + x^3

f[1]
```

The pattern does not match since x ≠ 1. *Mathematica* has no definition of f[1], and so f[1] is left unevaluated. The command f[x] = x^3 - x - 1 only gives a definition for the literal expression f[x]. Thus if we type f[x], x^3 - x- 1 will be returned. But no other argument of f will match. To make this function work as desired, use the underscore immediately after the function argument:

```
Clear[f]
f[x_] = x^3 - x - 1
f[1]

-1 - x + x^3

-1
```

Now the pattern works as desired.

A note about equal signs

Sometimes it is necessary to use := rather than = . But be careful!
:= means SET DELAYED while = means SET.
Thus LHS := RHS will set LHS equal to RHS only when LHS is called, not when the definition is made. Conversely, LHS = RHS will make the assignment immediately.

The := is used when you want LHS to define the "program" RHS.

Two cases of where use of = is necessary

Case 1 Differentiation

```
Clear[f]; f[x_] := x^3 - x - 1; fprime[x_] := ∂x f[x]
fprime[1]
```

```
– General::ivar : 1 is not a valid variable.

∂1 (-1)
```

The intent here was to set fprime[x] = f'[x] = 3x^2 - 1. However, the use of := in the definition of fprime[x] caused the definition to be applied only when fprime[1] was executed. *Mathematica* looks for the definition of fprime[x] and sets x = 1. Thus it attempts to evaluate D[f[1],1] =D[1,-1] rather than making the definition first (fprime[x] = 3x^2 - 1 and then setting x = 1 and evaluating. This is equivalent to the expression D[f[x],x] /. x-> 1)
The definition would have worked had we used = instead, or had we used f'[x].

```
Clear[f, fprime]
f[x_] := x^3 - x - 1; fprime[x_] = ∂x f[x]
fprime[1]
```

```
-1 + 3 x^2

2
```

Case 2 When % is used

Since % refers to the most recent output, one must be very careful when using % in assignments where SET DELAYED := is used. Consider the following:

```
Clear[f]
x^3 - x - 1
```

```
-1 - x + x^3
```

```
f[x_] := %
f[1]
```

The output is not at all what was desired! But why not? It is because f[1] finds the rule for f and then attempts to evaluate it at 1. However, by our definition of f as the most recent output, f[1] looks for the output of the line immediately preceding it, or the LITERAL OUTPUT x^3-x-1. This then is the output for f[1], and in fact, f of any

argument will give this result.
Here is the rule that *Mathematica* has set up for f:

```
Information["f", LongForm → False]

Global`f

f[x_] := %

Clear[f]

-1 - x + x^3

f[x_] = %
f[1]

-1 - x + x^3

-1
```

The assignment worked in the desired way. Explain why.
(Hint: look up the rule that *Mathematica* now has for f.)

Another case where := is necessary rather than = is in our definition of factorial. Try this example again but define factorial[n_] = n factorial[n-1]

Remember, on the Macintosh, Command-. will abort when *Mathematica* gets into an infinite loop. On the IBM, use the Kernel menu and choose Quit Kernel. You will still be able to save your notebook, but will have to restart the kernel in order to do any more calculations.

To understand what is happening, remember that the way*Mathematica* works is to evaluate an expression until it no longer changes! This is a recursive defintion for factorial[n] . . . it appears on both sides of the equation. *Mathematica* trys to "solve" for factorial[n] but finds that it is equal (by our definition) to n times factorial[n]! Thus there is no solution and no end to the evaluation. Use of a := makes the function work perfectly.

Pattern Matching with Function Arguments

Section 2.3 of the *Mathematica* book details how to test the arguments of a function to ascertain that they meet certain criteria. For example, to define array elements b[i] such that b[i] = *odd* if i is odd and b[i] = *even* for even i, use the `EvenQ` and `OddQ` functions. Additionally, the Integer? after i_ below tests that the argument of i is an integer.

```
b[i_Integer?EvenQ] := even; b[i_Integer?OddQ] := odd
```

```
b[3]
```

```
odd
```

```
b[18]
```

```
even
```

```
b[5.4]
```

```
b[5.4]
```

We can also test whether the arguments are lists, integers, text strings, etc. Built-in functinos such as PrimeQ, NumberQ, and NameQ will return true or false depending on the results of the test. Let's test whether the number 113 is prime:

```
PrimeQ[113]
```

```
True
```

Conditionals

As described in Section 2.5.8 of the *Mathematica* book, *Mathematica* has the following conditional structures:

```
Which
   /;
   If
   Switch
```

The If conditional has the structure

```
If[test, then, else]
```

An implementation of If can be seen in the following example, where the function TestandSort is defined. This function takes a list of numbers as its input, compares them to the sorted list, and if these lists are equal, evaluates to true. Otherwise, the numbers are sorted and returned.

```
TestandSort[x_List] :=
 If[x == Sort[x], Print["Numbers are in ascending order."],
  Print["The sorted numbers are: "]; Sort[x]]
```

```
TestandSort[{-16, 117, 49}]
```

```
The sorted numbers are:
```

```
{-16, 49, 117}
```

Another implementation of If is the following function, which tests whether an integer is odd or even:

```
Clear[TestNumber]
```

```
TestNumber[i_Integer] :=
 If[EvenQ[i], Print["The integer ", i, " is even."],
  Print["The integer ", i, " is odd."]]
```

```
TestNumber[6]
```

```
The integer 6 is even.
```

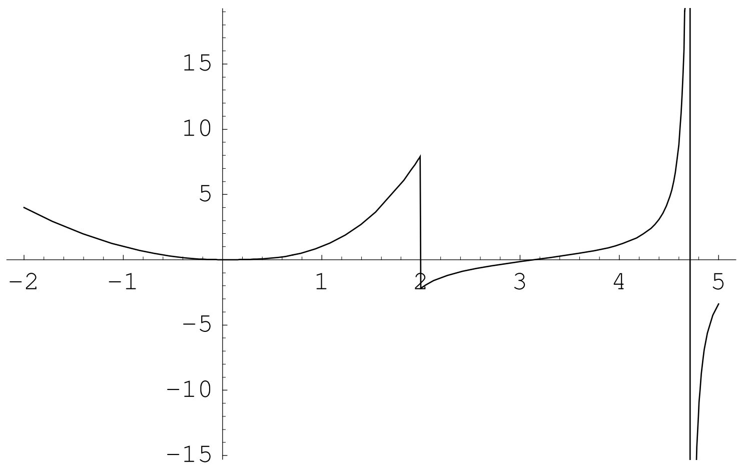
```
TestNumber[11]
```

```
The integer 11 is odd.
```

One example of a case where `Which` is very useful is in defining piecewise continuous functions for plotting of splines.

```
Clear[f]
f[x_] := Which[x < 0, x^2, x ≥ 0 && x < 2, x^3, True, Tan[x]]
```

```
Plot[f[x], {x, -2, 5}]
```

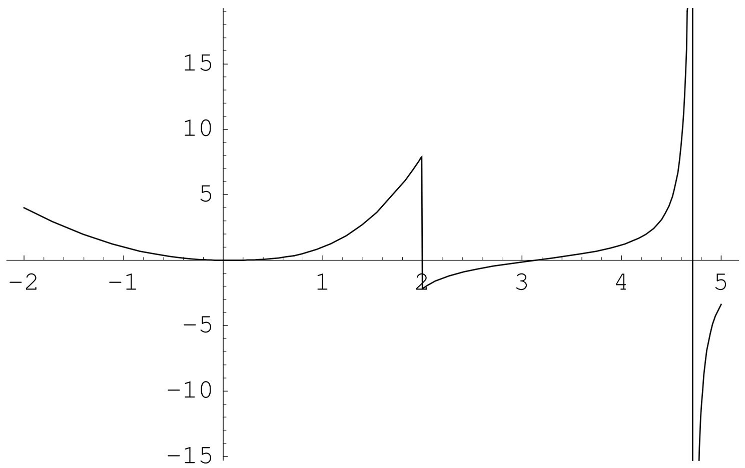


The /; conditional has the syntax

`lhs := rhs /; test`

which sets `lhs` equal to `rhs` if `test` is true.

```
Clear[f]
f[x_] := x^2 /; x < 0
f[x_] := x^3 /; x ≥ 0 && x < 2
f[x_] := Tan[x] /; x ≥ 2
Plot[f[x], {x, -2, 5}]
```



This is the same plot as before which shows that this technique is equivalent to using `Which`.

Exercise: Redo the graphing of the piecewise function f[x] using `If`.

Looping

Constructs for looping include `For`, `Do` and `While.` To illustrate these procedures the product ∏(x-i) for i running from 0 to 10 is printed using `For`, `Do` and `While`, respectively.

Demonstration of the For structure

```
myproduct = 1;  (* initial value *)
For[i = 0, i ≤ 10, i++, myproduct = myproduct (x - i)]
myproduct
```

```
(-10 + x) (-9 + x) (-8 + x) (-7 + x)
 (-6 + x) (-5 + x) (-4 + x) (-3 + x) (-2 + x) (-1 + x) x
```

Demonstration of the Do structure

```
product = 1; Clear[i]
Do[product = product (x - i), {i, 0, 10}]
product
```

```
(-10 + x) (-9 + x) (-8 + x) (-7 + x)
 (-6 + x) (-5 + x) (-4 + x) (-3 + x) (-2 + x) (-1 + x) x
```

Demonstration of the While structure

```
product = 1;  (* initialization *)
i = 0;
While[i ≤ 10, product = product (x - i); i ++]
product
```

```
(-10 + x) (-9 + x) (-8 + x) (-7 + x)
 (-6 + x) (-5 + x) (-4 + x) (-3 + x) (-2 + x) (-1 + x) x
```

Working with Lists

As you gain proficiency with `Mathematica`, you will doubtless find that your operations are more efficient when you work with lists. As a simple example, the following function Mean computes the average of a set of numbers. It is then used to compute the mean of the numbers 1,2, ... , 10.

```
                  Plus @@ x
Mean[x_List] := ----------
                 Length[x]
```

```
data = Table[i, {i, 10}]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
Mean[data]
```

$$\frac{11}{2}$$

Defining Your Own Rules

The Factorial Function

```
myfactorial[1] = 1;
myfactorial[n_] := n myfactorial[n - 1]
```

```
myfactorial[10]
```

```
3 628 800
```

Of course, *Mathematica* has a built in factorial function, Factorial[x]. Alternatively the symbol ! may be used:

```
10 !
```

```
3 628 800
```

Fibonacci Numbers The Fibonacci Numbers are the sequence {1,1,2,3,5,8,13, ... }. The nth number is the sum of the two preceding numbers.
A function, fibonacci[i], can be set up to compute these numbers and a table of results printed out.

```
myfibonacci[0] = myfibonacci[1] = 1;
myfibonacci[i_] := myfibonacci[i - 1] + myfibonacci[i - 2]
```

```
Table[{i, myfibonacci[i]}, {i, 0, 10}]
```

```
{{0, 1}, {1, 1}, {2, 2}, {3, 3}, {4, 5},
 {5, 8}, {6, 13}, {7, 21}, {8, 34}, {9, 55}, {10, 89}}
```

Newton's Method Another useful case where this technique could be applied is in Newton's Method for finding the root of a function, wherein the i-th estimate of the root x[i] is computed by the scheme $x_i = x_{i-1}$-f(x_{i-1})/f'(x_{i-1})
An initial guess x[0] for the root must be provided. As an example, we shall take some iterations on the root of f[x] = x^3 - x - 1 = 0, with an initial guess x[0] = 1.5.

```
Clear[f, x]
f[x_] = x^3 - x - 1;
x[0] = 1.5;
x[i_] := x[i - 1] - f[x[i - 1]] / f'[x[i - 1]]
Table[{i, x[i]}, {i, 0, 5}]
```

```
{{0, 1.5}, {1, 1.34783}, {2, 1.3252},
 {3, 1.32472}, {4, 1.32472}, {5, 1.32472}}
```

Of course, the built-in function FindRoot performs this but does not print out the iterations. Try FindRoot[f[x]==0,{x,1.5}] to see the estimate of the root obtained by Mathematica.

Teaching *Mathematica* New Tricks

If a desired function is not available in `Mathematica`, you can easily program it. As an example, the following routine defines a function called NewtonDD that performs interpolation by Newton's Divided Differences. The output will be identical to that of the intrinsic function `InterpolatingPolynomial`, but the new function is valuable for students to check if they are constructing their polynomials correctly.

Interpolation by Newton's Divided Differences

Procedure NewtonDD takes as input a list of n+1 {x,y} pairs and returns an interpolating polynomial of degree (at most) n in the variable *var*.

```
NewtonDD[data_List, var_Symbol] := Block[{i, n, poly, x, f},
  Do[x[i - 1] = data[[i, 1]]; f[i - 1] = data[[i, 2]], {i, Length[data]}];
  f[i_, 0] := f[i]; f[n_, i_] := (f[n, i - 1] - f[n - 1, i - 1]) / (x[n] - x[n - i]) /; i > 0;
  Sum[f[i, i] Product[(var - x[j - 1]), {j, 1, i}], {i, 0, Length[data] - 1}]]
```

$$\sum_{i=0}^{\text{Length[data]-1}} f[i,i] \prod_{j=1}^{i} (var - x[j-1])$$

Sample Calculations

Construct an interpolating polynomial through the data points (-6,-60),(-4,-9),(-3,0),(-1,0),(0,-3),(2,0),(3,12) by Newton's Divided Differences. Compare to the answer obtained by *Mathematica*'s intrinsic function InterpolatingPolynomial.

```
NewtonDD[
 {{-6, -60}, {-4, -9}, {-3, 0}, {-1, 0}, {0, -3}, {2, 0}, {3, 12}}, x]
```

$$-60+\frac{51\,(6+x)}{2}-\frac{11}{2}\,(4+x)\,(6+x)+\frac{1}{2}\,(3+x)\,(4+x)\,(6+x)$$

```
mypoly = Expand[%]
```

$$-3-\frac{5\,x}{2}+x^2+\frac{x^3}{2}$$

```
?? InterpolatingPolynomial
```

```
InterpolatingPolynomial[data, var] gives a polynomial in the variable var
  which provides an exact fit to a list of data. The data can have the
  forms {{x1, f1}, {x2, f2}, ... } or {f1, f2, ... }, where in the second
  case, the xi are taken to have values 1, 2, ... . The fi can be replaced
  by {fi, dfi, ddfi, ... }, specifying derivatives at the points xi.

Attributes[InterpolatingPolynomial] = {Protected}
```

```
InterpolatingPolynomial[
 {{-6, -60}, {-4, -9}, {-3, 0}, {-1, 0}, {0, -3}, {2, 0}, {3, 12}}, x]
```

$$-60+(6+x)\left(\frac{51}{2}+(4+x)\left(-\frac{11}{2}+\frac{3+x}{2}\right)\right)$$

```
intpoly = Expand[%]
```

$$-3-\frac{5\,x}{2}+x^2+\frac{x^3}{2}$$

```
mypoly === intpoly
```

```
True
```

LU-Factorization of Matrices

The code to perform LU factorization of matrices, which is a technique via which we may solve A.x = b by factorization of A into two triangular matrices L and U is shown below as a programming example.

```
Clear[LUFactorization, ConstructLandU]

ConstructLandU[n_] := Block[{i, j}, Unknowns = {};
  u[i_, i_] = 1; Do[Do[l[i, j] = 0, {i, n}, {j, i + 1, n}];
   Do[u[i, j] = 0, {i, 2, n}, {j, 1, i - 1}];
   U = Table[u[i, j], {i, n}, {j, n}]; L = Table[l[i, j], {i, n}, {j, n}];
   {n}]; Do[Unknowns = Append[Unknowns, l[i, j]], {i, n}, {j, i}];
  Do[Unknowns = Append[Unknowns, u[i, j]], {i, n}, {j, i + 1, n}]]
LUFactorization[a_List, b_List] :=
 Block[{n}, EquationList = {}; n = Length[a];
  ConstructLandU[n]; Do[EquationList = Append[EquationList,
     Flatten[L.U][[i]] == Flatten[a][[i]]], {i, Length[Flatten[L.U]]}];
  solution = Flatten[Solve[EquationList, Unknowns]];
  U = U /. solution; L = L /. solution; y = LinearSolve[L, b];
  x = LinearSolve[U, y]; Print["The solution x = ", x]]
```

Let us solve the following system (this arises in 2-point Gaussian Quadrature) by LU-factorization and compare to the known solution of {1,1}:

```
LUFactorization[{{1, 1}, {-0.57735, 0.57735}}, {2, 0}]
```

```
The solution x = {1., 1.}
```

To check hand calculations, L, U and y can be printed out. For example, L is

```
MatrixForm[L]
```

$$\begin{pmatrix} 1. & 0 \\ -0.57735 & 1.1547 \end{pmatrix}$$

Online Help From Mathematica

To obtain information about a command, say `FindRoot`, just type `?FindRoot`. For further information, use two `??` as in `??FindRoot`. Note that the file info.m must be loaded. To load, use File, Open, Packages, info.m or configure the Startup Settings in the Edit menu.
For a list of all symbols beginning (say) with Ja, type ?Ja*

```
Information["FindRoot", LongForm → False]
```

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

Information about the % symbol

```
Information["%", LongForm → False]
```

```
%n or Out[n] is a global object that is assigned to be the value produced on
  the nth output line. % gives the last result generated. %% gives the
  result before last. %% ... % (k times) gives the kth previous result.
```

To use the online help, pull down the apple on the top left of the screen and select About *Mathematica*... then click the HELP button.

During a session if a beep occurs and you want an explanation, select the Why the Beep?... from the apple menu. The most common errors are syntax errors, such as using PLOT for Plot or pi for Pi; or having unmatched parentheses. Also watch out when you define functions.
You should always Clear a function before defining it. Often "old" definitions may be remembered otherwise, leading to error.

How to find out what version of Mathematica you're running:
Type $Version or $VersionNumber.

```
$Version
```

```
4.0 for Microsoft Windows (July 26, 1999)
```