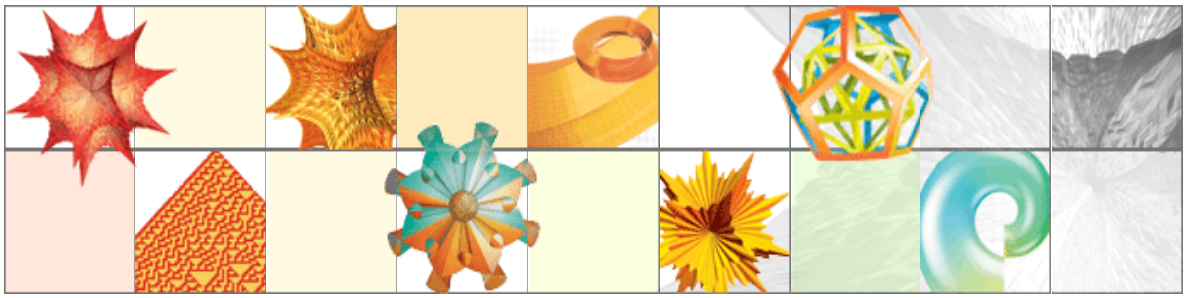
Mathematica Tutorial

Megan Frary

Boise State University



Version 1

Last updated November 2008

Table of Contents

1. Introduction

Welcome to Mathematica! I hope you will find the program to be a useful tool. I created this tutorial guide from a number of individual tutorials I had written for different classes. The information is organized by topic and should provide sufficient background for any Mathematica related assignments I will give.

In section 2, the Mathematica documentation center is explained. You'll want to become very familiar with it and use it often! In section 3, some of the peculiarities of Mathematica are explained. Version 6 of the software goes a long way toward making the program more user-friendly and helping users to identify errors in their syntax. The rest of the sections explain different capabilities of the software.

This tutorial gives a number of examples of Mathematica input that are distinguished by a different font:

```
Mathematica input looks like this
```

If you are trying to follow along in Mathematica (which is a great idea!), do not just cut and paste though. Some of the symbols won't be pasted correctly and you'll be frustrated. Just type the lines in yourself.

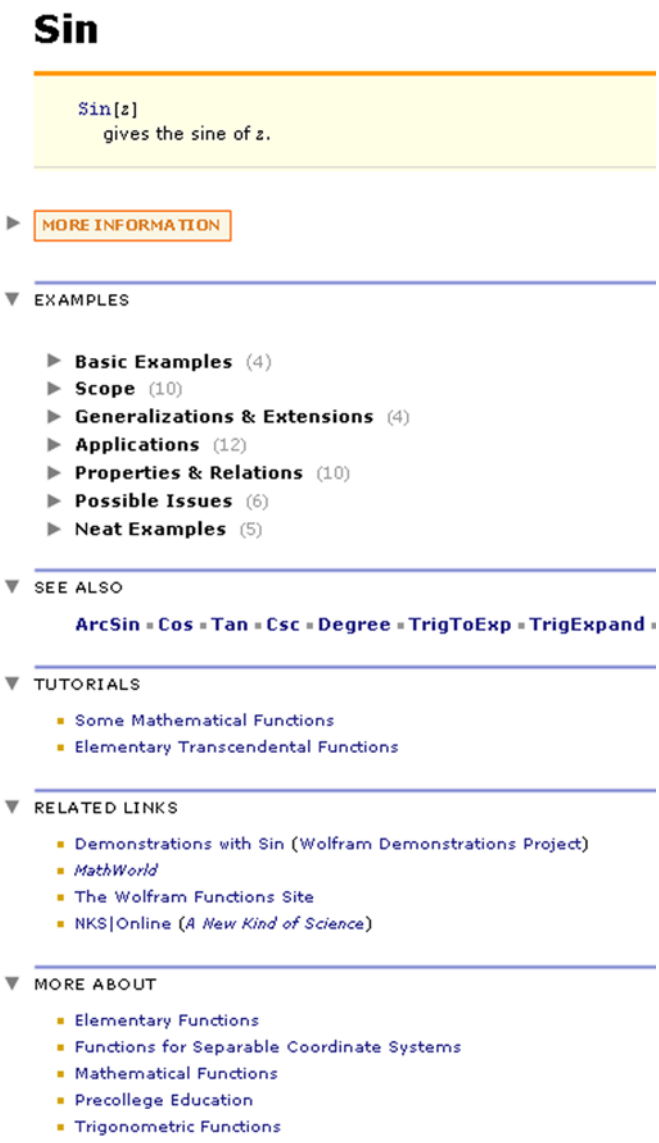
If you encounter errors in this guide or have suggestions for making it better, please let me know!

Megan Frary

August 2008

2. Mathematica Documentation Center

- This tutorial is intended to be a beginner's guide to Mathematica. However, the reader is strongly encouraged to refer to the documentation center that is part of Mathematica.
- Although the documentation center doesn't always explain beginning princples, it has very thorough documentation about each and every function.
- A sample page from the documentation center is shown here. All pages have the same format which includes, in the yellow bar at the top, the syntax for using the function.
- The orange more information box has useful information about the arguments of the function (for Sin it notes that the argument is to be in radians).
- Any of the sideways arrows in the Examples section can be expanded. The Basic Examples section will show you how to actually use the function. For many functions, there will be a section called Options where you can see options like plot styling.
- The links at the bottom are references to other useful information like related functions, tutorials and so on.



- You will do well to become familiar with all the documentation center has to offer.
- From the front page of the documentation center, you can access good overviews of the functions and capabilities Mathematica has.

3. Mathematica Peculiarities

3.1. Creating input

Mathematica is extremely case sensitive. If you are getting an error, check your capitalization.

Mathematica is very sensitive to the type of bracket you use. The syntax is as follows:

- [] are used in functions, e.g., Sin[x]
- () are used for grouping, e.g., (4+6)/2
- {} are used for lists, e.g., {1,2,3,4}

Comments can be included in cells by enclosing the comments in "(*" and "*)". For example:

```
2+5 (* gives the sum of 2+5 *)
```

Mathematica version 6 uses color to indicate errors and is much more user-friendly at diagnosing such errors. Functions that are misspelled (or don't exist) will show up in blue font while those that are correct will turn black. Mathematica also colors brackets that are missing their closing pair. Go to Help→ why the coloring for more information on what colors mean in input cells.

3.2. Evaluating cells

Use enter (not return) or shift+return to evaluate cells. Return adds a new line to the input.

Mathematica gives exact answers. You have to force it to give numerical values.

Putting a semicolon at the end of an input line suppresses the output and helps keep things neat.

3.3. Getting help

The documentation center (Help Menu) is incredibly useful. I refer to it all the time!

If you can't remember how to use a function or command, put the cursor on the function and press Shift+F1 to bring up documentation regarding that function.

4. Add-On applications

In some cases, Mathematica needs to be told to load a certain package. The syntax for loading a built-in package is:

```
Needs[packagename]
```

All package names end with the character `, which is not an apostrophe but is on the key to the left of the number 1.

Some packages we might use include:

```
Needs["LinearRegression`"]
```

for performing linear regressions and finding the R^2 value of a fit,

```
Needs["VectorAnalysis`"]
```

for vector calculus including Curl, Divergence, Gradient and Laplacian,

```
Needs["FourierTransform`"]
```

for (quite obviously) performing Fourier transforms,

```
Needs["PhysicalConstants`"]
```

for values of many important physical constants, and

```
Needs["Units`"]
```

for performing all kinds of unit conversions.

More information can be found at:

http://documents.wolfram.com/mathematica/Add-onsLinks/StandardPackages/index.en.html

5. Using the physical constants module

Begin by loading the necessary packages (see section 4 for more help):

```
Needs["PhysicalConstants`"]
Needs["Units`"]
```

5.1. Perform calculations using physical constants

Find the value of Boltzmann's constant, k_B:

```
BoltzmannConstant
```

A list of constants available can be found by searching the help file for "PhysicalConstants". Next, calculate the value of k_BT at room temperature:

```
BoltzmannConstant 300 Kelvin
```

Note that the space implies multiplication and that units are being used.

5.2. Convert units using the built in converters

Find the value of k_BT at room temperature in eV:

```
Convert[BoltzmannConstant 300 Kelvin,ElectronVolt]
```

A list of units which Mathematica knows can be found by searching help for "units".

5.3. Use units in a user-defined function

Functions can also take parameters and variables with units and return the answer with units.

```
Density[m_,v_]:=m/v
Density[12.4 Gram,(2×10^-2 Meter)^3]
```

Note that this doesn't work when trying to plot a function.

6. Using the ElementData, ChemicalData, and LatticeData modules

6.1. Accessing element data

Mathematica contains a lot of built-in information about the elements and many chemical compounds. The command

```
ElementData[]
```

will return a list of all elements Mathematica knows. To find information about an element, use either command:

```
ElementData[“name”, “property”]
ElementData[n, “property”]
```

Note that the property name must be in quotes, as must the element name if it is given. As with all Mathematica commands, capitalization is important! Some of the properties Mathematica has tabulated (though not for all elements) include bulk modulus, thermal conductivity, melting point, refractive index, lattice constants, space group name, half life, and year of discovery. The complete list can be found in the ElementData entry in the documentation center (click on More Information just under the yellow bars). These data can be turned into tables and plotted. For more information on plotting lists, see section 16.

6.2. Accessing chemical compound data

As before, the command

```
ChemicalData[]
```

will return a list of all compounds Mathematica knows. The command:

```
ChemicalData[“name”]
```

gives a structure diagram for the chemical. To find other information, use the command:

```
ChemicalData[“name”, “property”]
```

Note that the property name must be in quotes, as must the element name if it is given. There are surprisingly lots of chemicals and properties available. Refer to the documentation center.

6.3. Accessing lattice data

Mathematica also knows a lot about lattices (both 2D and 3D). The command

```
LatticeData[n]
```

will return a list of all n-dimensional lattices Mathematica knows. The command:

```
LatticeData[“name”, “property”]
```

will give the specified property for the lattice. Mathematica can calculate things like basis vectors, packing radii, and many other things you will probably never need to use.

7. Loading a new style sheet

7.1. Apply a style sheet

Go to the Format Menu, choose Style Sheet, choose Creative and then choose Pastel. You will use this style sheet for all Mathematica assignments.

7.2. Assign a style to a cell

Begin by typing your name into the Mathematica window. To apply the style "Title", click on the cell bracket to the right of the cell. Go to the Format menu, select Style, and select Title. Alternatively, you can select the cell bracket and push ALT+1. Using the same approach, you can apply the styles subtitle, subsubtitle, section, subsection, subsubsection, and text.

7.3. Minimize subsections

Using section and subsection headings, parts of your Mathematica notebook can be minimized while other parts are visible. In Mathematica assignments, each problem will be a separate section. Click below the subtitle cell and type "Practice". Assign the style Section. In the next cell, type "Addition". Assign the style subsection. Generate a few cells of addition problems. Create a new subsection called "Subtraction" and generate a few cells of subtraction problems. Notice that double clicking on the rightmost bracket will minimize the entire section "Math Practice". (Double clicking again will expand it). Double clicking on the second rightmost bracket will minimize the subsection.

8. Built-In Functions

The number of built-in function in Mathematica is astonishing. This section introduces a number of functions we will use most often. Information about other functions can be obtained in the documentation center. In Mathematica, most basic operations can be accomplished with the keyboard alone but can also be entered using the Basic Math input palette. If the palette is not displayed, go to Palettes→ BasicMathInput to revive it.

8.1. Basic operations

Addition is accomplished using the plus sign.

```
2+5
```

Subtraction is accomplished using the minus sign.

```
5-2
```

Multiplication is accomplished using an asterisk (keyboard), the "x" symbol in the Basic Math Input palette, or by putting a space between the symbols to be multiplied. Mathematica is very particular about this. For example, "2 a" means 2 times a, but "2a" is just a variable named 2a.

```
4*7
```

```
4 a
```

```
2 8
```

Division is accomplished using a slash (keyboard), the division symbol in the Basic Math Input palette, or the fraction input button in the Basic Math Input palette (top right button).

```
4/7
```

```
100/10
```

Numbers can be raised to powers by using the carrot symbol (keyboard) or the Basic Math Input palette (top left button).

```
10^-7
```

```
5^2
```

8.2. Commonly used functions

As Section 1 explained, Mathematica is very precise about brackets. The arguments of a function are given in [] brackets.

Mathematica gives exact values. To force it to give numerical values, the function N is used. Here, the function N[] can be applied to an argument or can be added after the fact as //N:

```
N[Pi]
```

```
Pi//N
```

To find the absolute value, use Abs[]:

```
Abs[-10]
```

To find the natural logarithm of a number, use Log[], the base b log of n is found by Log[b,n]:

```
Log[5]
Log[10,5]
```

To find e raised to a power, use Exp[]:

```
Exp[4]
```

All of the trigonometric functions are built in and take an argument in radians:

```
Sin[Pi]
Cos[Pi]
Tan[Pi]
```

All of the inverse trigonometric functions are also built in and return an answer in radians (pay careful attention to the capitalization):

```
ArcSin[Pi]
ArcCos[Pi]
ArcTan[Pi]
```

To find the square root of a number, use Sqrt[]:

```
Sqrt[49]
Sqrt[x]
```

To round a number to the nearest integer, use Round [x]; to round to the nearest multiple of a, use Round[x,a]:

```
Round[7.56]
Round[Sqrt[50],0.01]
```

To use the imaginary unit $(-1)^{1/2}$, you can enter it as a capital I or use the Basic Math Input palette. The value of e can likewise be represented entered as capital E, using the Basic Math Input palette, or using the function Exp[]. Finally, the π can be entered as Pi or using the Basic Math Input palette. Be sure to not use I or E as variables ever!

```
I^2
E^2
Sqrt[Pi]
Sqrt[π]
```

To find the factorial of a number, simply use !

```
10!
```

9. Using the print command

9.1. Print text

The function Print[expr] will print expr as output:

```
Print[“I’m a student at Boise State”]
```

9.2. Print a variable

Start by defining a = 4 and then print the value of a.

```
a = 4;
Print[a]
```

Compare using Print[a] with just entering a:

```
a
```

Print generates a line of text output as compared to mathematical output.

9.3. Create an output statement that calculates the value of a function

You can also use Print to report the values of functions mixed with text. For example:

```
Print[“The numerical value of Pi is “, Pi]
```

Warning: Do not try to copy and paste this text. Mathematica is particular about its quotation marks. Just type it in yourself. Notice how this allows you to insert a calculation into a sentence. Here’s another example:

```
Print["One centimeter is equal to ", 1/2.54," inches"]
```

10. Defining and working with functions

10.1. Assignment vs. replacement

In Mathematica, LHS = RHS is an immediate assignment in which RHS is evaluated at the time the assignment is made. For example, define the gas constant:

```
Rgas = 8.3145 (*J/mol/K*)
```

In contrast, LHS := RHS (notice the colon) is a delayed assignment in which RHS is evaluated each time LSF is called. This will be very handy for creating our own functions.

10.2. Create a function

Let's say the we want to define a function f[x] = x^2 + 6. We use the following:

```
f[x_]:=x^2+6
```

The underscore next to x is very important! It signals Mathematica that x is just a dummy variable and can be replaced by anything. Now we can use our function by giving a value for x:

```
f[2]
f[boise]
```

Functions can be defined with any number of variables:

```
density[mass_,volume_]:=mass/volume
density[10(*g*)/3(*cm^3*)]
```

10.3. Nesting functions inside other functions

You can use functions inside of other functions easily. For example, let's modify the density function to calculate volume ($V = \pi r^2 h$) for a cylinder. First define volume:

```
volume[height_, radius_]:= Pi radius^2 height
```

Notice that spaces are used to indicate multiplication. Now we can define density as a function of height and radius also, but calling the volume function:

```
density[mass_,height_,radius_]:= mass/volume[height,radius]
```

Notice that although density is a function of height and radius, those come into the function through volume. Also, because you are using the volume function (and not defining it here), do not put the underscores after the variables.

10.4. Conditional or piecewise functions

It is also possible to define piecewise or conditional functions in Mathematica by using /;

```
g[x_]:= x^2/;x≥0
g[x_]:= -x^2/;x<0
```

11. Use the Solve function

Solve[eqns,vars] solves an equation for the variables specified. It is very important to use a double equals sign (= =) in the equation. The double equals tells Mathematica that these two things are exactly equal (instead of assigning the RHS to the LHS).

11.1. Use the solve function by entering the function manually

Solve can be used to solve a polynomial:

```
Solve[x^2 + 3 x +10 == 0,x]
```

This will return the roots of the equation. In this case, they are imaginary and are given with square roots in them. The numerical values of the roots can be obtained using NSolve:

```
NSolve[x^2 + 3 x +10 == 0,x]
```

11.2. Using the solve function with user-defined functions

Solve can be used just as easily with user-defined functions. In Sec. 5 density was defined for a solid cylinder. To find the mass of a cylinder with known density and volume, use:

```
Solve[density[m,10(*cm*),1(*cm*)]= = 4 (*g/cm^3*),m]
```

This will return a value for mass which, according to our units, will be in grams.

11.3. Check the solutions

We can check the solution by plugging the value of mass into the density equation and veriftying it returns a density of 4.

```
density[40 Pi,10,1]
```

On a regular basis, there is no reason to check unless you are unsure of the answer.

11.4. Solve equations simultaneously

The solve function can also be used to solve equations simultaneously. Let's say you want to find the point where two lines intersect ($m_1 = 4$, $m_2 = -3$, $b_1 = 1$, $b_2 = 2$):

```
Solve[{y==4x+1,y==-3x+2},{x,y}]
```

Notice that the list of equations is given in curly brackets as is the list of variables.

12. For and Do Loops

12.1. Do Loops

A do loop can be called using Do[exp, {i, i_{min}, i_{max}, di}], which would look like this:

```
Do[Print[4i],{i,3,13,2}]
```

which would print out 4 times i for i = 3, 5, 7, 9, 11 and 13. If you don't use Print and just use 4i instead, Mathematica will execute the multiplication but won't provide any output.

12.2. For Loops

A for loop can be called using For[start, test, incr, body]. A For loop can replace a Do loop. For example, the statement below could be written with a For or Do loop:

```
For[i=0, i<5, i++, Print["Twice ", i, " is " 2 i]]
Do[Print["Twice ", i, " is " 2 i],{i,0,5,1}]
```

Both print out five lines that show the values of 2i. However, a For loop could also be used when the condition isn't just a number of outputs. For example,

```
For[k=1,Prime[k]<100,k++,Print[Prime[k]]]
```

will print all prime numbers less than 100 (we couldn't use a Do loop because we didn't know beforehand how many there would be. A For loop can execute more than one command. Simply separate them by a semicolon:

```
For[k=1,k^3<100,k++,Print[k];Print[k^2];Print[k^3]]
```

12.3. Other Loop Structures in Mathematica

Mathematica has great power at creating loops or nested structures. More information can be obtained by searching for "looping constructs" in the documentation center.

13. Creating lists and tables

In Mathematica, Tables and Lists are generated in the same way. The Mathematica book explains several ways to generate tables and lists automatically.

13.1. Create a list of x and y values by inputting values by hand

Here we will be concerned with creating tables in which we specify manually all of the entries. The approach is to give the table a name, then specify the entries, one row at a time. Each row should be enclosed in brackets. So, if we want a table that has these entries:

1 2 3
4 5 6
7 8 9

we use command:

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

Notice how each row is given in {} and separated by commas. The entire table is given in {}. Tthe format {{x1,y1},{x2,y2},…} is used to create a list of x-y pairs:

```
mylist={{1,2},{2,4},{3,6},{4,8},{5,10},{6,12}}
```

13.2. Create a list by importing data from a file.

A second method for creating a list is to import data from a file. The first step is to figure out where to store the file. Mathematica looks at certain file paths by default, so it's easiest to store the file of interest in the places Mathematica already looks. To find out the default paths, type

```
$Path
```

Make sure the file you're trying to import (we will use one called "practice.csv") is stored in one of the default folders. As long as you have the file stored where Mathematica will look, it is easy to import the file as:

```
Import["practice.csv"]
```

You can assign the imported data to a variable (in our case, a list) by using:

```
plist = Import["practice.csv"]
```

13.3. Creating an empty table to fill with values

Sometimes it will be useful to first create a table with certain dimensions, and then fill in the values later using a function. To create a list of zeros, use:

```
list1=Table[0,{10}]
```

This creates the table {0,0,0,0,0,0,0,0,0,0}. To create a table of x,y pairs, use:

```
list2=Table[0,{i,12},{j,2}]
```

This creates a table of 12 x,y pairs whose initial values are all zeroes. By default, i and j start at 1 and run to value given.

13.4. Creating a table from a function

The entries in a table can also be generated from a function. For example, the command

```
tableA=Table[10i+j,{i,4},{j,3}]
```

will generate a table that look like this:

```
{{11,12,13},{21,22,23},{31,32,33},{41,42,43}}
```

13.5. Creating a table from a user-defined function

A user-defined function can also be used to generate table entries. Define the function sumfunction to find the sum of a and b.

```
mysum[a_,b_]:=a+b
```

Next, generate a table that calculates the values of a+b for 0 < a, b, < 5:

```
mytable = {{mysum[1,1], mysum[1,2], mysum[1,3], mysum[1,4]},
{mysum[2,1], mysum[2,2], mysum[2,3], mysum[2,4]},
{mysum[3,1], mysum[3,2], mysum[3,3], mysum[3,4]},
{mysum[4,1], mysum[4,2], mysum[4,3], mysum[4,4]}}
```

Notice that the table could have been made more easily using the example above.

13.6. Manipulating data in the table

Let's say that we (for some reason) want to keep the same x values in the table but want to use the square root of the y-values. You can be creative about how you do this, but one way that I found worked is to first define a new variable (plist2) that is the same as the original list:

```
L2 = L
```

Now use a For loop to change the y value of each item in the table:

```
For[i=1,i<Length[L2]+1,i++,L2[[i,2]]=Sqrt[L2[[i,2]]]]
```

13.7. Extracting a value from a table or list

Let's say that you've generated a table of the first 10 primes:

```
myprimes=Table[Prime[k],{k,1,10}]
```

and want to find out the value of the 5th one. You can use listname[[x]] to extract element x:

```
myprimes[[5]]
```

If you have a table (like the one created in a previous section) and want to extract the element in the first row, second column, you can use listname[[row,column]] to extract it:

```
tableA[[1,2]]
```

13.8. Other list-related functions

Lists can be endlessly manipulated in Mathematica. Use the documentation center to find information on the following commands: First, Last, Part, Rest, Take, Drop, Append, Prepend, Insert, ReplacePart, and Join

14. Formatting lists and tables

14.1. Outputting a table as a table

Instead of having the table printed as a list, Mathematica can display it in a tabular form by using the TableForm function:

```
TableForm[mytable]
```

14.2. Making a nice looking table

Row and column headings can be added as well using the option:

```
TableHeadings→{{row headings},{column headings}}]
```

For example:

```
TableForm[mytable,TableHeadings→{{"a=1"," a=2"," a=3","
a=4"},{"b=1","b=2","b=3","b=4"}}]
```

The list of row headings (or the list of column headings) can be replaced by

```
None
```

if there are no headings are needed. The option TableSpacing can also be used to adjust the spacing between rows and columns. For example:

```
TableSpacing→{row spacing, column spacing}]
```

All together, we can make a nice table using:

```
TableForm[mytable,TableHeadings→{{"a=1"," a=2"," a=3","
a=4"},{"b=1","b=2","b=3","b=4"}},TableSpacing→{4,2}]
```

15. The Plot Function:

15.1. Plotting a function of a single variable

The plot function is called using

```
Plot[function to be plotted, {independent variable, min, max}]
```

For example, to plot sin(x) from -π to π, we would enter

```
Plot[Sin[x],{x,-π,π}]
```

The Plot function can also be used to plot several functions at once using

```
Plot[{first function, second function, third function}, {independent variable, min, max}]
```

Notice the {} around the list of functions. To plot sin(x), cos(x) and tan(x), we would enter

```
Plot[{Sin[x], Cos[x], Tan[x]},{x,-π,π}]
```

15.2. Plotting a function of many variables as a function of one variable

It will be common to have a function of many variables and want to plot it as a function of one:

```
concentration[h_,T_]:= exp(-h/T/8.3145)
```

We can make this plot with the regular Plot function by varying T and fixing h to a known value (or plot curves for different values). The command is:

```
Plot[concentration[100,T], {T,100,1000}]
```

15.3. Modifying the plot style

If you are following along in Mathematica, you can see that these plots aren't very well labeled. It's quite easy to add options to make a nicer plot. The function now looks like this

```
Plot[function to be plotted, {independent variable, min, max}, style options here]
```

Some commonly used style options include:

```
PlotRange→{-1.2, 1.2}
```

which fixes the range of the y-axis,

```
PlotRange→{{-3.14,3.14},{-1.2, 1.2}}
```

which fixes the range of both axes,

```
PlotStyle→{Pink,Thickness[0.01]}
```

which defines the color and thickness of a single curve,

```
PlotStyle→{{Pink,Thickness[0.01]}, {Blue,Thickness[0.01]}}
```

which defines the color and thickness of multiple curves (notice the nested lists)

```
Axes→False
```

which gets rid of the axes inside a plot,

```
Frame→True
```

which frames the plot,

```
PlotLabel→"Trig functions"
```

which labels the plot,

```
FrameLabel→{"Value of x", "Value of function"}
```

which labels the axes,

```
ImageSize→400
```

which resizes the plot,

```
AspectRatio→1
```

which specifies the aspect ratio, and

```
BaseStyle→{FontFamily→"Times", FontSize→14}
```

which specifies the font and size to use.

You can use the help section in Mathematica to understand what each command does if it is not obvious. Note that when you type "->" in Mathematica that it turns into a little arrow which is used to define the various style options.

Finally, it is useful to be able to put labels on our graphs. The easiest way to do this is to add to the end of the style options the following command:

```
Epilog→{Text[label,{x location, y location}]}
```

Multiple labels can be added like this:

```
Epilog→{Text["sin",{2.5,0.8}],Text["cos",{-1,0.8}]}
```

To make a nice (and well labeled plot), we use:

```
Plot[{Sin[x], Cos[x]}, {x, -π, π}, PlotRange→{-1.2, 1.2},
PlotStyle→ {{Pink, Thickness[0.01]}, {Blue,
Thickness[0.01]}}, Frame → True, FrameLabel→{"Value of x",
"Value of function"}, PlotLabel→"Trig functions",
BaseStyle→{FontFamily→"Times", FontSize→14},
ImageSize→400, Epilog→{Text["sin",{2.5,0.8}],Text["cos",{-
1,0.8}]}]
```

16. Plotting lists

Functions aren't the only things that Mathematica can plot (nor the only things you'll want to plot). Mathematica can also plot lists. Assume that the list "mylist" is a set of x and y values.

```
mylist={{1,2},{2,4},{3,6},{4,8},{5,10},{6,12}}
```

16.1. Plotting discrete data with ListPlot

To make a very basic list plot, use the command:

```
ListPlot[mylist]
```

You can also plot several lists on the same plot using:

```
ListPlot[{mylist, myotherlist }]
```

Notice that the lists to be plotted are inside {} and separated by a comma.

This is just a barebones plot though. You can make a nice looking ListPlot the same way you make a nice looking Plot.

NOTE: If you just give a list of single values (e.g., mylist = {1,2,3,4,5,6}), the resulting plot assumes that your x-values are uniformly spaced and labels them from 1 to x (where x is the number of entries in the list).

16.2. Make a nice list plot

Many of the same commands from Plot can be used to make a nice ListPlot:

```
Frame→True,
PlotRange→{{0,7},{0,13}},
FrameLabel→{"My list plot","y values"},
PlotLabel→"My List Plot",
BaseStyle→{FontFamily→"Times", FontSize→14},
ImageSize→400
```

However, some new commands are available with ListPlot: For the point size command, you can enter point sizes such as Small, Medium or Large, or enter a number that represents a percentage of the graph width. This results in the following plot:

```
PlotStyle→PointSize[Large]
```

will make large points,

```
PlotStyle→{Red,PointSize[Large]}
```

will make large red points, and

```
PlotStyle→{Red,PointSize[0.1]}
```

will make red points that are 0.1 times the width of the image.

```
Joined→True
```

will connect all of the points in the list (but doesn't show the points) and

```
Mesh→All
```

will show the points when joined is true. Note that Mesh has other options available.

```
PlotStyle→PointSize[Large]
```

Refer to ListPlot in the documentation center for even more options! To make a reasonably nice and well labeled list plot, we would use:

```
ListPlot[mylist,PlotStyle→{Red,PointSize[0.05]},
Frame→True,PlotRange→{{0,7},{0,13}},FrameLabel→{"x
values","y values"}, PlotLabel→"My List Plot",
BaseStyle→{FontFamily→"Times", FontSize→14},
ImageSize→400]
```

17. Making Logarithmic Plots

Some functions are best plotted with one axis using a logarithmic scale. Mathematica has options for both function and list plots in which one or both axes uses a log scale.

17.1. The LogPlot Command

The LogPlot command generates a plot of f[x] with a logarithmic y-axis:

```
LogPlot[function,{x,xmin,xmax}]
```

For example,

```
LogPlot[4^x,{x,0,10}];
```

LogPlot has the same options as Plot. Refer to section 15.2 for plot styling options. In addition, multiple functions can be plotted at once, just like with Plot.

17.2. The LogLinearPlot Command

To do a linear (y-axis) - logarithmic plot (x-axis) of a function, use:

```
LogLinearPlot[function,{x,xmin,xmax}]
```

17.3. The LogLogPlot Command

To do a log-log plot (x- and y-axes) of a function, use:

```
LogLogPlot[function,{x,xmin,xmax}]
```

17.4. The ListLogPlot Command

To do a logarithmic plot (y-axis) of a list, use:

```
ListLogPlot[list]
```

ListLogPlot has the same options as ListPlot. Refer to section 16.2 for plot styling options.

17.5. The ListLogLinearPlot Command

To do a linear (y-axis) - logarithmic plot (x-axis) of a list, use:

```
ListLogLinearPlot[function,{x,xmin,xmax}]
```

17.6. The ListLogLogPlot Command

To do a log-log plot (x- and y-axes) of a list, use:

```
ListLogLogPlot[function]
```

18. Using Contour Plots

The ContourPlot function in Mathematica v6 allows you to plot individual contours of a function of two variables (i.e., f[x,y] = 10) or an entire map of contours.

18.1. Making a discrete ContourPlot

Here we will find the set of x and y values that correspond to a given value of a function. Let's use the function:

```
trigfunction[x_,y_]:= Sin[xy] + Cos[y]
```

Now, plot all points where trigfunction = 0.5. In this case, we use the ContourPlot function and specify that we only want points = 0.5. Note the double equals sign in the command:

```
ContourPlot[trigfunction[x,y]==0.5,{x,-Pi,Pi},{y,-Pi,Pi}]
```

Multiple values can be plotted on the same chart, just as multiple functions can be plotted with the Plot command. In this case, the different equalities are listed in {}, as:

```
ContourPlot[{trigfunction[x,y]==0.2,trigfunction[x,y]==0.5,
trigfunction[x,y]==0.8},{x, -Pi,Pi},{y,-Pi,Pi}]
```

18.2. Make a nice looking discrete contour plot

You can use many of the options from Plot to make a nicer contour plot. The only options that are new are those that allow you to specify the color and thickness of the contour. For example:

```
ContourStyle→{Green,Thickness[0.01]}
```

Note that the documentation center in Mathematica gives many (tons, in fact) other options that can be used to make interesting and informational contour plots.

18.3. Make a continuous contour plot

Let's say you want to plot the values of a function over a range of x and y values. This is the more standard contour plot that Mathematica can make. Begin with the basic function:

```
ContourPlot[trigfunction[x,y],{x,-Pi,Pi},{y,-Pi,Pi}]
```

In this graph, the contours are not labeled and the color function is the default value.

18.4. Make a nice looking continuous contour plot

Again, most of the commands you've learned to use with Plot still work, but now you can define some other conditions:

```
ContourLabels→Automatic
```

which automatically shows the value of the function for each contour,

```
ColorFunction→"RedBlueTones"
```

changes the color scheme; color functions can be found in the Palettes menu under Color Schemes

```
Contours→20
```

will make 20 equally spaced contours, and

```
Contours→{-20,-15,-10,-8,-6,-4,-2,0,2,4,6,8,10,15,20}
```

will draw a contour line at each value specified.

19. Plotting in 3D

19.1. Making a 3D surface plot from a function of x and y

Let's consider the same function defined in section 18.1:

```
trigfunction[x_,y_]:= Sin[xy] + Cos[y]
```

To make a 3D plot of the value of the function, we use Plot3D:

```
Plot3D[trigfunction[x,y],{x,-2π,2π},{y,-2π,2π}]
```

Many of the same plot style options available with 2D graphs are available here. Refer to the documentation center for options to the Plot3D function.

19.2. Making a 3D surface plot from discrete data

ListPlot3D can operate in two ways. It can be used to generate a 3D surface plot representing an array of height values (where the indices of the entries in the array are the x, y value.

```
myarray = {{1,1,2,3},{2,3,4,4},{3,5,6,7},{4,5,8,9}}
ListPlot3D[myarray]
```

It can also generate a 3D surface plot from a list of {x, y, z} values where the z-values specify the height of the surface.

```
mylist = {{1,1,4},{2,4,7},{1,4,5},{2,1,8}}
ListPlot3D[mylist]
```

19.3. Making a 3D plot of discrete x, y, z data

Mathematica can also make 3D scatter plots using the ListPointPlot3D function. Again, ListPointPlot3D can operate using an array where the indices of the entries give the x, y values and the array value is the height.

```
ListPointPlot3D[myarray]
```

It can also take in a list of {x, y, z} values and plot the points. For example:

```
ListPointPlot3D[mylist]
```

Again, the options available to ListPlot are largely available here as well.

19.4. Making a 3D contour plot of a function of x, y and z

The function ContourPlot3D can generate a 3D contour plot of a function of x, y, and z:

```
ContourPlot3D[f,{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}]
```

The same function can also plot a single contour (like in 2D) by specifying the value g of the function f(x,y,z) that should be plotted:

```
ContourPlot3D[f==g,{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}]
```

Refer to the documentation center for information on changing the colors, opacity, and how to make cutaways to show the contours.

19.5. Plotting miscellaneous 3D shapes (Graphics3D)

Mathematica can also generate a lot of 3D shapes such as cylinders, spheres, cuboids, and polyhedra using the function Graphics3D. A very simple example of this is:

```
Graphics3D[Cylinder[]]
```

Refer to the documentation center for more information on each shape (by searching for its name). For example, the cylinder can be rotated, resized, colored differently. Also, look at PolyhedronData in the documentation center for more 3D shape stuff.

20. Manipulate

The ability easily create interactive applications with Manipulate is one of the coolest features in Mathematica version 6. Start by looking up Manipulate in the documentation center, then going through the Introduction to Manipulate Tutorial (link in the tutorials section).

20.1. Basics of Manipulate

Manipulate can be called in a number of ways, including:

```
Manipulate[expr,{u,umin,umax}]
```

which generates a version of expression with controls added to allow interactive manipulation of the value of u,

```
Manipulate[expr,{u,umin,umax,du}]
```

which allows the value of u to vary between u_{min} and u_{max} in steps of du,

```
Manipulate[expr,{{u,uinit},umin,umax,...}]
```

which takes the initial value of u to be u_{init},

```
Manipulate[expr,{{u,uinit,ulabel},umin,umax}]
```

which labels the controls for u with u_{lablel},

```
Manipulate[expr,{u,{u1,u2,u3...}}]
```

which allows u to take discrete values u_1, u_2…, and

```
Manipulate[expr,{u,umin,umax},{v,vmin,vmax}]
```

which provides controls for multiple variables u and v (but many more can be added!).

Most of the interactive tools that I've seen and developed rely on the manipulation of a plot. Here are a few examples:

```
Manipulate[Plot[Sin[b z],{z,-Pi,Pi}],{b,1,10}]
```

```
Manipulate[ListPlot[Table[RandomReal[],{i,k},{j,2}],
PlotStyle→{Red,PointSize[0.04]},PlotRange→{{0,1},{0,1}}],
{k,1,20,1}]
```

```
Manipulate[Plot[(x+a)^b,{x,-10,5}], {{a,2,"value of
a"},0,4}, {{b,1,"value of b"},0,4}]
```

20.2. Making a better looking interactive tool

As you can see from the examples above, it is very easy to make a neat tool. The method for labeling the sliders was already introduced. There are hundreds of other ways to make nice interactive tools. If the expression in Manipulate is Plot (or any plotting functions), all of the options available there (colors, labels, plot styles, etc.) can be included in the expression field of Manipulate. A few ways to better label the interactive window itself will be explained here. To begin, the value of the adjustable variable can be shown automatically using the command:

```
Manipulate[Plot[Sin[b z], {z,-Pi,Pi}], {b,1,10, Appearance
->"Labeled"}]
```

The controller type can be assigned to be radio buttons, setter (i.e., select a button), slider or popup menu using the following command:

```
Manipulate[Plot[Sin[b z],{z,-Pi,Pi}],{b,1,10,1},
ControlType->PopupMenu]

Manipulate[Plot[Sin[b z],{z,-Pi,Pi}],{b,1,10,1},
ControlType->Slider]

Manipulate[Plot[Sin[b z],{z,-Pi,Pi}],{b,1,10,1},
ControlType->Setter]

Manipulate[Plot[Sin[b z],{z,-Pi,Pi}],{b,1,10,1},
ControlType->RadioButton]
```

Notice that most of those options will only work when you specify a value of du.

The label style can also be adjusted so that the fonts are different sizes, styles, or colors. Some other commands that might help the creative user include Grid, Row, and Text. Here is a glimpse at how you might build up a complex interactive module:

```
Manipulate[Grid[{{Plot[Sin[b z],{z,-Pi,Pi},
ImageSize→300]},{},{Row[{Text@"The value of the function at
x = Pi is ",Round[Sin[b Pi],0.001],Text@"."}]}}],
{b,1,10,Appearance→"Labeled"}]
```

21. Fitting Data

One very cool feature of Mathematica is its ability to fit data to just about any function you can imagine. There are two functions to use: Fit and FindFit. Also, Mathematica can do a statistical analysis of the fit and return those results.

21.1. Fitting data with Fit

The way that the Fit function finds a least-squares fit to a list of data as a linear combination of the functions and variables. You give it the data, a list of functions to fit, and a list of variables that you are fitting. Let's say we have a list of day (x,y pairs) that we want to fit with a line. In other words, we want to fit to the functions 1 and x (to get an intercept and slope):

```
Fit[mylist,{1,x},x]
```

For more information on the Fit function, look in the documentation center in Mathematica. Notice that in between the {} we put the functions to use. The last argument is the variables. (Mathematica can fit to many variables, here we only use x). We could also assign the fit to a variable as:

```
pfit = Fit[mylist,{1,x},x]
```

We can see how our fit looks by using (this is why we defined pfit):

```
Plot[pfit,{x,0,20}]
```

To fit the data to more functions (like a polynomial), use:

```
Fit[mylist,{1,x,x^2},x]
```

We can see how our fit looks by using (this is why we defined pfit):

```
Plot[pfit,{x,0,20}]
```

21.2. Fitting the data with FindFit

FindFit finds numerical values of the parameters that make the expression give a best fit to the data as a function of variables. Strictly speaking, it looks like this:

```
FindFit[data, expression, parameters, variables]
```

Let's say that we have some data to fit with the function x^a, but we don't know the value of a. We can use FindFit like this:

```
FindFit[mylist, x^a, {a}, x]
```

FindFit works well when fitting a list of a known function (built-in or user-defined). For example, when trying to fit a list of data to a normal distribution function, the function is known but the mean and standard deviation are unknown. FindFit then works like this:

```
FindFit[otherlist,PDF[NormalDistribution[μ,σ],x],{μ,σ},x]
```

More information about statistical distributions is given in the next section.

21.3. Performing a linear regression

Start by loading the linear regression package

```
Needs["LinearRegression`"]
```

See section 4 for information about loading packages and the documentation center for complete notes about what the linear regression package can do. The Regress function works sort of like the Fit function but it also gives a complete analysis of the fit:

```
Regress[mylist,{1,x},x]
```

To just extract the R^2 value, use:

```
Regress[mylist,{1,x},x, RegressionReport→{RSquared}]
```

22. Statistics

Mathematica has a wide variety of statistics functions that can be used on discrete data (i.e., lists) or on statistical distributions (see section 23). Here we will focus on applying only a few of those functions to discrete data. For more information, search for "Statistics" in the documentation center.

22.1. Averaging

Mathematica can find all sorts of "averages" in a list of data including the mean, median, geometric mean, harmonic mean, and others. For example:

```
testlist = {1,1,2,2,3,3,4,4,5,5,6,7,8,9,10}
Mean[testlist]
Median[testlist]
```

To see how any of these functions works, you can also enter a symbolic list and get a symbolic answer:

```
Mean[{a,b,c,d}]
```

22.2. Variations in the data

To get an idea of the scatter associated with a list, Mathematica can find both the variance and standard deviation of a list:

```
Variance[testlist]
StandardDeviation[testlist]
```

Remember that the variance is just the standard deviation squared.

22.3. Other statistical measures of a list

When trying to analyze a data set, it might also be helpful to find other statistical measures. For example:

```
RootMeanSquare[testlist]
Norm[testlist]
```

RootMeanSquare gives the square root of the sum of the squares divided by the square root of the number of items in the list. Norm gives the square root of the sum of the squares of the absolute values.

Although we won't go into it here, Mathematica can perform ANOVA. Look it up in help!

23. Statistical Distributions

Mathematica contains a number of built-in statistical distributions, including the normal, chisquare, lognormal, and beta distributions (among others). It also has several discrete distributions including the Poisson and binomial distributions. For more information, look up statistical distributions in the documentation center.

23.1. Probability Density Functions

Most of the continuous distributions work in the same way, so this tutorial will just work with the normal distribution. Mathematica contains a built in function for the Normal Distribution:

```
NormalDistribution[m,σ]
```

You should note that this is just a distribution function and not a function which generates the probability at any point. In other words, entering NormalDistribution[5,2] doesn't get you anything! One thing you might like to know from a normal distribution is the probability associated with a certain value of x given values of m (mean) and σ (standard deviation). In case, the built in function PDF will work:

```
PDF[distribution,x]
```

which gives the probability density function for the distribution evaluated at x. We can easily generate our own function now that will give the probability at x for a given m and σ:

```
mypf[m_,σ_,x_]:=PDF[NormalDistribution[m,σ],x]
```

One advantage to this approach (besides it being shorter) is that we can now use x as a variable. For example, to plot the normal distribution, use:

```
Plot[mypf[5,2,x],{x,0,10}]
```

23.2. Cumulative Density functions

In some cases, a statistical distribution is more useful as a cumulative distribution function (as opposed to a probability density function). To generate the CDF of a statistical distribution, use:

```
CDF[distribution,x]
```

For example, the Weibull distribution is typically used in CDF form (not PDF form). To generate a function that returns the cumulative probability of failure P_f as a function of stress, use the cumulative distribution:

```
CDF[WeibullDistribution[m,σ],stress]
```

To see what the CDF looks like for a given set of parameters, use Plot:

```
Plot[CDF[WeibullDistribution[8,400],stress],{stress,0,1000}
```

We can also define a function that gives P_f for a certain set of Weibull parameters:

```
mypf[m_,σ_,stress_]:=CDF[WeibullDistribution[m,σ],stress]
```

24. Vectors and Matrices

24.1. Entering vectors

A vector is entered just like a one-dimensional list using {} and commas to separate entries. A vector can have any dimension. In materials science, we will often deal with 3-D vectors that represent directions in a Cartesian coordinate system.

24.2. Vector operations: Dot and Cross Products, Vector Angle

There are two ways to find the dot product of two vectors, using a "." symbol or the function Dot.

```
a = {1,1,2}; b = {-1,1,0};
a.b
Dot[a,b]
```

Both functions work equally well whether or not you define the vectors as variables first. For example, we could have written instead:

```
Dot[{1,1,2},{-1,1,0}]
```

To find the cross product of two vectors, use the function Cross:

```
Cross[a,b]
```

To find the angle between two vectors (in radians), use the function VectorAngle:

```
VectorAngle[a,b]
```

To find the magnitude of a vector, use the function Norm:

```
Norm[a]
```

Notice that if you want to see how one of these functions work, you can enter symbolic vectors such as {h1, k1, m1} and {h2, k2, m2}:

```
Dot[{h1,k1,m1},{h2,k2,m2}]
```

24.3. Entering matrices

A matrix can be entered in at least two ways in Mathematica. One method is to enter a m × n matrix is to do so as you would a m × n table. For example:

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

The second method is to insert a blank matrix with dimensions m×n and then fill it in. To create a blank matrix, go to Insert→Table/Matrix→New, then enter the number of rows and columns.

24.4. Matrix arithmetic

Simple arithmetic functions can be applied to matrices if they have the same dimension:

```
m1 + m2
```

```
m1 - m2
```

To multiply matrices using true matrix multiplication, use either a dot or the Dot function:

```
m1.m2
```

```
Dot[m1,m2]
```

Using a multiplication sign (asterisk) or space will result in simple multiplication of identical entries and not multiplication of the entire matrices.

24.5. Matrix operations: Inverse, Transpose, Det

Mathematica can find the transpose of matrix m using the function Transpose:

```
Transpose[m]
```

Mathematica can also find the inverse and determinant of a square matrix s:

```
Inverse[s]
```

```
Det[s]
```

As with the vector operations, the matrix operations can be used on a variable that represents a matrix or the matrix itself.

25. Linear Algebra and Eigenvalues

25.1. Solving a system of equations

Mathematica can easily solve systems of linear equations using one of two different methods. In the first method, the Solve function is used with the following syntax:

```
Solve[{list of equations},{variables to solve for}]
```

For example:

```
Solve[{3x-4y==-9,2x+6y==7},{x,y}]
Solve[{x-2y==a,2x+3y==b},{x,y}]
```

A second approach to solving a system of linear equations uses the function LinearSolve with the following syntax:

```
LinearSolve[matrix of prefactors,matrix of RHS]
```

For example, to solve the two systems given above, use:

```
LinearSolve[{{3,-4},{2,6}},{-9,7}]
LinearSolve[{{1,-2},{2,3}},{a,b}]
```

25.2. Eigensystems

Eigenvalues are a special set of scalars associated with a linear system of equations. If A is an m × m matrix, the eigenvectors X satisfy the equation $AX = \lambda X$ (where X is a vector and λ is a scalar. There are m pairs of eigenvalues and eigenvectors. In other words, for a 3×3 matrix, we can write $AX_1 = \lambda_1 X_1$, $AX_2 = \lambda_2 X_2$ and $AX_3 = \lambda_3 X_3$ (where X_i is a vector). The eigenvalues λ satisfy the characteristic equation $(A - \lambda I) X = 0$, where I is the identity matrix. For more information, see http://mathworld.wolfram.com/Eigenvalue.html. As an example:

```
Eigenvalues[{{a11,a12},{a21,a22}}]
Eigenvectors[{{a11,a12},{a21,a22}}]
B = {{5,6,7},{10,2,4},{1,3,9}}
Eigenvalues[B]
Eigenvectors[B]
```

Both the eigenvectors and eigenvalues can be obtained at the same time using Eigensystem:

```
Eigensystem[B]
```

26. Vector Calculus

Before performing any vector calculus, the VectorAnalysis package must be loaded:

```
Needs["VectorAnalysis`"]
```

The following functions are defined assuming that V is a scalar quantity and **F** is a vector.

26.1. Gradient

The gradient of the scalar function V is the vector given by: $\text{grad}V = \nabla V = \frac{\partial V}{\partial x}\hat{\mathbf{i}} + \frac{\partial V}{\partial y}\hat{\mathbf{j}} + \frac{\partial V}{\partial z}\hat{\mathbf{k}}$.

Physically, the function V represents the value of a quantity at all points in space. The gradient points in the direction of the maximum rate of increase in the quantity at a given point. The magnitude of the gradient of V is the rate of increase in V at a given point. In Mathematica, it is required that the x, y and z whose derivatives will be taken are entered as Xx, Yy, and Zz (I don't know why!). For example:

```
Grad[Xx^2 Zz + Xx Yy^2 + Yy Zz^2]
```

Notice that the Xx terms turn black, indicating that you've entered a recognized function.

26.2. Divergence

The divergence of the vector **F** is the scalar given by: $\text{div}\mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} + \frac{\partial F_3}{\partial z}$.

Physically, the function F represents the flux of a quantity as a function of position. The divergence is the rate at which the flux causes the density of the quantity flowing to change. Again, the symbols Xx, Yy and Zz must be used in Mathematica. For example:

```
Div[{Xx^2 Zz, Xx Yy^2, Yy Zz^2}]
```

Notice that the argument of the function is a vector.

26.3. Curl

The curl of the vector **F** is the vector given by: $\text{curl}\,\mathbf{F} = \nabla \times \mathbf{F} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_1 & F_2 & F_3 \end{vmatrix}$.

Physically, the function F represents a vector quantity (like an electric field). The curl is the amount of "rotation" of the contents at a point in space. If the curl = 0, the field is called irrotational. The curl shows up in Maxwell's equations for electromagnetism. For example:

```
Curl[{Xx^2 Zz, Xx Yy^2, Yy Zz^2}]
```

Again, it is important to remember to use Xx, Yy and Zz.

26.4. Laplacian

The Laplacian of the scalar function V is the scalar given by: $\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2}$.

Physically, the scalar function V represents the value of a quantity at all points in space. The Laplacian is the divergence of the gradient of the function V. The Laplacian shows up in diffusion, the Schrodinger equation and the wave equation. The Laplacian can also be written for different coordinate systems (e.g., cylindrical). In Mathematica, the Laplacian is called by:

```
Laplacian[Xx^2 Zz + Xx Yy^2 + Yy Zz^2]
```

Some of the information in this section came from http://mathworld.wolfram.com.

27. Complex Numbers

Mathematica can deal with both explicit complex numbers and symbolic complex variables.

27.1. Entering complex numbers

In Mathematica, the symbol "I" represents $(-1)^{1/2}$. The symbol can also be entered using the Input Palette. For example, to enter the number 4+3*i*, use:

```
4 + 3 I
```

Mathematica can also handle symbolic complex variables. For example:

```
Sin[x + I y]
```

Remember to put a space between the number (or symbol) and I.

27.2. Working with parts of complex numbers

In some cases, it is necessary to consider only the real (or imaginary) part of a complex number or function. For example, consider the function Exp[zI]. If you try to plot Exp[zI] as a function of z, nothing happens; this is because the function is complex and Mathematica doesn't know how to plot it. However, we can look at the real and imaginary parts separately:

```
Plot[Re[Exp[z I]],{z, -2π, 2π}]
```

```
Plot[Im[Exp[z I]],{z, -2π, 2π}]
```

The functions Re[] and Im[] can also be used to return just the real (or imaginary) parts of numbers or variables.

27.3. Moving between trigonometric and exponential notations

Recall that the function $Ae^{i\theta}$ is equal to $A\,\mathrm{Cos}\,\theta + i\,A\,\mathrm{Sin}\,\theta$. Mathematica can easily convert between the two expressions using the functions ExpToTrig and TrigToExp. For example:

```
ExpToTrig[A Exp[I θ]]
```

```
TrigToExp[A Cos[θ] + I A Sin[θ]]
```

It is important to note that the TrigToExp function doesn't work unless trigonometric functions are given (even when the mixed number can be written as an exponential). For example:

```
ExpToTrig[8 Exp[I π/4]]
```

gives a value of (4+4*i*)Sqrt[2]. However,

```
TrigToExp[(4+4i)Sqrt[2]]
```

does not return $8e^{i\pi/4}$.

27.4. Other properties of complex numbers

As the previous section described, complex numbers can be expressed in two equivalent ways: as a mixed number (x+y*i*) or as an exponential ($Ae^{i\theta}$). There is a simple relationship between the

two: x = A Cos θ and y = A Sin θ. Likewise, if the trigonometric form is given, $A = x^2 + y^2$ and $\theta = \tan^{-1}(y/z)$. The value of A corresponds to the absolute value (or magnitude) of the number and can be found using Abs:

```
Abs[4 Cos[Pi/3] + 4 I Sin[Pi/3]]
Abs[4 Exp[I Pi/3]]
```

Note that Abs only works on numeric quantities. Although it is straightforward enough to calculate the value of θ, the phase angle, manually, the function Arg can also be used:

Arg[3+4 I]

As with Abs, Arg only works on numeric quantities. Using Abs and Arg together would allow you to go from a mixed complex number to the exponential form.

27.5. Complex Conjugate

One final property of complex numbers that is of interest is the complex conjugate. The product of a complex number with its conjugate results in the square of the absolute value of the number. The function Conjugate works for both numerical and symbolic expressions:

```
Conjugate[3 + 4 I]
```

It is useful to note that Mathematica makes no assumptions about the variables in an expression, so you will sometimes get a funny looking answer back when you take the conjugate of a symbolic expression. Mathematica has not assumed that the variables are positive and/or real.

28. Multivariable Calculus

Mathematica can do more derivatives and integrals than you can keep straight in your head!

28.1. Partial derivatives

The partial derivative of a function is, for example, $\partial f / \partial x$. To find the partial derivative of a function with respect to a particular variable, the function is:

```
D[function, variable]
```

For example:

```
D[2x^b,x]
```

Derivatives can be nested inside one another:

```
D[D[2x^b,x],x]
```

Derivatives can also be nested together with differentiation with respect to different variables:

```
D[D[x^3 y^4],x],y]
```

Finally, if you've defined a function f[x_] (of one variable only), the derivative can be found as:

```
f'[x]
```

28.2. Total derivatives

The total derivative of a function is, for example, $df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy$. To do this in Mathematica, the function is Dt:

```
f[x_,y_]:= x^2 y
Dt[f[x,y]]
```

Taking the total derivative with respect to one variable (applying the chain rule), also uses Dt:

```
Dt[f[x,y],x]
```

Notice that we now specify the variable used in the derivative. In this case, other variables are assumed to change also with x (whereas they were assumed independent of x in the partial derivative case). Compare the results from Dt and D on the function *f* defined above.

28.3. Integration

Mathematica can do both definite and indefinite integrals, as well as multiple integrals. To do an indefinite integral, just specify the function to be integrated and the variable of integration:

```
Integrate[x^2,x]
```

Notice that Mathematica does not add a constant of integration (although one may exist). For a definite integral, specify the range of integration:

```
Integrate[x^2,{x,0,10}]
```

Finally, integrals can be nested within one another:

```
Integrate[Integrate[x^2y^3,x],y]
```

Integrals may be entered using the integrate command or the integral buttons on the input palette.

Mathematica also includes a number of special functions used in mathematical physics that define certain integrals. These functions are encountered when trying to integrate other functions. Examples include the gamma function, error function, zeta function, Legendre function, and Bessel function. Look at the tutorial on Special Functions for more information on these and other special functions.

29. Differential Equations

Mathematica can solve both linear and nonlinear ordinary differential equations. It can also solve some partial differential equations. If you do not specify enough initial or boundary conditions (the number required is equal to the number of derivatives in the equation), Mathematica will give solutions that involve an appropriate number of undetermined coefficients. The documentation center has an unbelievably thorough review of the topic of differential equations; simply enter tutorial/DSolveOverview into the search window to view it.

29.1. General solutions to differential equations

The function DSolve will attempt to solve the differential equation. The function is called using:

```
DSolve[equation, function to solve for, variables]
```

Here is an example of a linear ordinary differential equation for y(x):

```
DSolve[y'[x]+y[x]==1,y[x],x]
```

Notice that whenever the function y appears in the command that it is referred to as "y[x]". This is very important! In addition, the double equal sign is needed. In this example, the derivative is entered with the "prime" notation. The derivative could also have been written D[y[x],x]. The prime notation can be easier for entering multiple derivatives. Here is an example of a linear partial differential equation for the function y(x,z):

```
DSolve[D[y[x,z],x]+D[y[x,z],z]==1/(xz),y[x,z],{x,z}]
```

In both of the examples above, the solution will include at least one undetermined constant which Mathematica calls C[1] (with subsequent constants called C[2], C[3], etc.).

29.2. Solutions to differential equations with boundary conditions

To eliminate undetermined constants from the solution, it is necessary to specify boundary or initial conditions. Initial conditions are added using:

```
DSolve[{eqn, cond1, cond2}, function, variables]
```

Notice that the equation and conditions are enclosed in curly brackets, thus forming a list. The conditions need also to include the double equal sign and use square brackets. For example:

```
DSolve[{y'[x]+y[x]==1,y[0]==7},y[x],x]
```

30. Fourier Series

Mathematica can calculate either the coefficients for Fourier series or the entire series themselves using either the trigonometric or exponential representations. To begin, load the Fourier Series package:

```
Needs["FourierSeries`"]
```

A periodic function defined for $-\pi \le x \le \pi$ can be represented by a Fourier series as:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

or

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx)\right)$$

with

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, e^{-inx} dx\,,$$

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) dx\,,$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx\,,$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx\,.$$

One way to deal with a Fourier series is to determine the coefficients using the equations above and then write the series as a sum over *n*. One trick is needed to do this: Mathematica needs to be told that n is an integer. In the Integrate function, add a statement:

```
1/Pi Integrate[f[x] Cos[n x],{x,-Pi,Pi}, Assumptions→
Element[n,Integers]]
```

Mathematica can also determine the coefficients or the entire series. The only catch is that Mathematica takes the default periodicity to be $-½ \le x \le ½$. The function FourierParameters allows us to change to $-\pi \le x \le \pi$. To find the coefficients c_n, use:

```
FourierCoefficient[function, variable, indexing variable,
FourierParameters→{-1,1/(2π)}]
```

The indexing variable is the one that is being summed over (n in the examples above). The FourierParameters term sets the period to 2π. For example, to find the coefficients for x^2, use:

```
FourierCoefficient[x^2, x, n, FourierParameters→{-1,1/(2π)}]
```

To find the coefficients a_n and b_n, use:

```
FourierCosCoefficient[function, variable, indexing variable,
FourierParameters→{-1,1/(2π)}]
```

```
FourierSinCoefficient[function, variable, indexing variable,
FourierParameters→{-1,1/(2π)}]
```

To have Mathematica generate the entire exponential series for *n* terms, use the function FourierSeries:

```
FourierSeries[x^2, x, 10, FourierParameters→{-1,1/(2π)}]
```

Notice that *n* must be a real integer now. The function FourierTrigSeries gives the Sin and Cos representation:

```
FourierTrigSeries[x^2, x, 10, FourierParameters→{-1,1/(2π)}]
```

Mathematica can do a lot more with Fourier series and Fourier transforms.