# Introduction to Mathematica

*Mathematica* is documented in *The Mathematica Book*, Version 4 by Stephen Wolfram. Any serious user should have a copy of this book. Some on-line documentation can be found at

```
http://documents.wolfram.com/v4/
```

These notes provide a *very brief* introduction to the use of Mathematica.

**Gettting Started.** At the UNIX level simply type

```
mathematica
```

to start the Notebook version of Mathematica. Typing `math` starts the line version. In the Notebook version you can read in a file by pulling down `File` to `Open...` and finding the desired file. Another convenient way to read a file, say `file1`, into the current session of Mathematica is

```
<<file1
```

When working on programs in Mathematica I prefer to write it to a file (using an editor) and then read it into a Mathematica session. This way when mistakes occur (and they do!), then I can edit the file in another window and re-read the file into Mathematica. This is particularly easy using the Notebook version.

To exit Mathematica at the Notebook level pull down `File` and go to `Quit`. At the line level type `Exit[]`.

**Arithmetic Operations.** The arithmetic operations `+` and `-` are as usual addition and subtraction. Multiplication of $x$ times $y$ is `x*y` and the division of $x$ by $y$ is `x/y`. The operation of $x$ to the power $y$, $x^y$, is `x^y`. A common beginner's error is to think `x/y+2` is $\frac{x}{y+2}$ when in fact it is $\frac{x}{y}+2$. To get the desired result use parentheses: `x/(y+2)`. Similarly, `a^b*c` is $a^b\,c$ whereas `a^(b*c)` is $a^{bc}$.

**User Defined Functions.** In Mathematica the function $f(x) = x^2$ is defined by

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

On the left-hand side the $x$ appears as `x_` to denote that $x$ is a variable. After $f$ has been so defined, typing `f[4]` on a command line will result in the output 16. Typing `f[a+b]` will result in the output $(a+b)^2$.

To define, say, the function $g(x, y) = x^2 + 6y^4$ of *two* variables $x$ and $y$:

```
g[x_,y_]:=x^2+6*y^4;
```

Observe this example ended with a semicolon whereas the first example did not. Ending with a semicolon tells Mathematica not to show the output. This rule applies to all Mathematica inputs. (Sometimes the output is quite lengthy and one does not want to see it printed on the screen.)

**Built-In Functions.** Mathematica has many built-in functions that are familiar from calculus: $\sin x$, $\cos x$, $\exp x$, $\log x$, to name but a few. They are all called with the same syntax, e.g. `Sin[x]`, `Log[x]`, etc. A function we will use in Math 131 is

```
Random[]
```

which when called produces a number $x$, $0 < x < 1$, which is uniformly distributed on $(0, 1)$. This is what we mean when we say informally, "pick a number at random between 0 and 1."

Mathematica has many functions which carry out algebraic operations. For example, if we apply the Mathematica function `Expand` to our function `f[a+b]` defined above, e.g.

```
Expand[f[a+b]]
```

the output will be $a^2 + 2ab + b^2$.

If we input `Log[2]`, the output is $\log 2$. If we want a numerical approximation to $\log 2$ we use the Mathematica function `N`, e.g.

```
N[Log[2]]
```

results in the output 0.693147. If we want accuracy to, say, 15 decimal places we input

```
N[Log[2],15]
```

and we get the output 0.693147180559945.

**Mathematical Constants.** Mathematica has symbols for some common mathematical constants: `Pi` ($= \pi \approx 3.14159$), `E` ($= e \approx 2.71828$), `I` ($i = \sqrt{-1}$) and `Infinity` ($\infty$). For example,

```
Sin[Pi]
```

evaluates to 0 and

```
E^(2*Pi*I)
```

evaluates to 1.

**Incrementing Variables.** Mathematica has commands that increment variables which follow the programming language $C$. For example, $i++$ increments the value of $i$ by 1 and $i+=di$ adds $di$ to the value of $i$. These constructions are useful in the programming part of Mathematica.

**Conditionals.** Mathematica provides various ways to set up conditionals, which specify that particular expressions should be evaluated only if certain conditions hold. The most common such construction is the `If` statement which has the general form `If[`*test*, *then*, *else*`]` which evaluates *then* if *test* is `True`, and evaluates *else* if it is `False`. For example the input

```
If[7<8,x,y]
```

returns the output $x$.

The expression *test* is a Boolean variable that evaluates to either `True` or `False`. Note that in Boolean expressions when testing whether $x$ equals $y$ one writes `x==y` (note double equal sign).

A more interesting example is

```
If[Random[]<.5,1,-1]
```

Here the function `Random` is first called and if it returns a number less that 0.5 then the If statement returns the number 1 but if `Random` returns a number greater than or equal to 0.5 the If statement returns the number -1. This construction is useful for simulating the toss of a fair coin.

**Loops.** There are programming constructs that allow an expression to be evaluated several times. The three most common are `Do`, `While`, and `For`. These are explained beginning on page 328 of Wolfram's book. Here are some typical examples

```
Do[Print[f[j]],{j,1,3}]
```

If `f` is the function defined earlier ($f(x) = x^2$), then the ouput of the above statement is $1, 4, 9$. The general form is

`Do[`*expr*`,{`*i*`,`*imin*`,`*imax*`}]`

where *expr* is repetitively evaluated with $i$ varying from *imin* to *imax* in steps of 1. Writing $\{i, imin, imax, di\}$ instead gives steps of $di$.

In many situations in probability one doesn't know the number of total iterations since this may be random. For example, toss a coin until the first Tail appears. In these cases the `While` construction is particularly useful. The general form is

`While[`*test*, *body*`]`

evaluates *body* repetitively so long as *test* is `True`.

For example, if we initialize the expression `position` to zero:

```
position=0
```

then the expression

```
While[Random[]<.5,position++]
```

counts the number of times, say, Heads consecutively appears in a given simulation of coin tossing. (This information will be in the expression `position`).

The `For` loop has the form

```
For[i=1,i<=n,i++,expr]
```

which evaluates the expression *expr* for values of $i = 1, 2, 3, \ldots, n$.

**Modules.** In an earlier example we showed how the user defines functions. The construction used above is good for single line statements. Many times we wish to define a function which requires several Mathematica statements. Also, in the process we use expressions which are only necessary for the evaluation of the function. It is a good habit to keep these internal expressions local to the evaluation of the function we are defining. This can all be done in the `Module` command. The general form is

`Module[{`*a*`,`*b*`,...},` *proc*`]`

that is, a procedure with local variables $a$, $b$, ....

As an example, suppose we toss a coin $n$ times and we want to know the proportion of heads to the total number of tosses. Also, we want this proportion in decimal form. Here is a simple procedure to do this:

```
coin[n_]:=Module[{headcounter=0},
For[i=1,i<=n,i++,If[Random[]<.5,headcounter++]]; N[headcounter/n,10]]
```

The expression `headcounter` is a local variable that is initialized to the value 0. I ran this for 1000 tosses by inputing `coin[1000]` with the result 0.492. For 10,000 tosses I got the result 0.499.

**Graphics.** Here are two graphics commands frequently used: `Plot` and `ListPlot`. For example,

```
Plot[Sin[x],{x,0,2*Pi}]
```

is the command to plot $\sin x$ for $x$ in the domain $0 \leq x \leq 2\pi$. There are several options that allow the user to customize the resulting graph. Given a list $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\}\}$ of coordinates of points—let's call this list `PointList`—the command

```
ListPlot[PointList]
```

produces a plot of these points. If one wants consecutive points in the list joined by a straightline (as we frequently do in probability), then one writes

```
ListPlot[PointList,PlotJoined->True]
```

If instead of the list $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\}\}$, one uses $\{y_1, y_2, \ldots, y_n\}$ then the $x$-coordinates are defaulted to the values $1, 2, \ldots, n$.