

Some Very Basic Mathematica Usage

Shift-Enter. Shift-Enter. Have I told you about Shift-Enter?

The most important thing to know is how to evaluate a cell. When you've typed in some stuff you want *Mathematica* to do for you, hit Shift-Enter, or the Enter on the numeric keypad. Just regular "enter" won't do it. Whoever made this user interface decision is responsible for an enormous amount of profanity that has been directed at the computer screens of novice *Mathematica* users.

The first lab for the IT multivariable calculus course (2374) has a good introduction to *Mathematica*. You can get it from <http://www.math.umn.edu/math2374/>. Download Lab 1A. It's quite useful. If you're looking for stuff on doing calculus with *Mathematica*, you should definitely check out those labs. They have lots of stuff, and if you can figure out Jon's code, you can do quite a lot in *Mathematica*.

Also, this program is very narcissistic and italicizes its name automatically when you type it.

Like capital letters and square brackets? You'll LOVE Mathematica.

Functions and constants generally use capital letters and square brackets:

```
In[1] := Sin[ $\pi$ /e]
```

```
In[2] := PartitionsP[13]
```

```
In[3] := HermiteH[3, x]
```

Lots of functions are of the form Function[blah , {options}]. For instance:

```
In[4] := Integrate[Sin[x^2], {x, 0,  $\pi$ }]
```

```
In[5] := Sum[1/n, {n, 1, 5}]
```

```
In[6] := Product[(3j + 1)!/(5 + j)!, {j, 0, 4}]
```

Defining your own functions

Everyone wants to define their own functions and variables. Variables are easy:

```
In[7] := a = 12
```

```
In[8] := b = x * y
```

```
In[9] := stegasaurus = Plot[x * Sin[x - 1], {x, 0, 1}]
```

As you see, you can use entire words for variables. *Mathematica* distinguishes between words-as-variables and letters put next to each other to mean "multiplication" with spaces. Sometimes spaces are hard to see, so it's best to use asterisks to eliminate confusion.

```
In[10] := a b
```

```
In[11] := ab
```

For functions, the key is to use underscores on the left-hand side of the function. For example,

```
In[12] := f[x_, y_, z_] = x + y + z + 2
```

On the left side, you use "x_" to signify one of the variables or parameters. On the right-hand side, just use that variable as normal. When you don't use an underscore, *Mathematica* assumes you're defining the function for a *particular* value. If the variable doesn't have a particular value, weird things happen.

```
In[13] := NumASMs[n_] = Product[(3j + 1)! / (n + j)!, {j, 0, n - 1}]
```

Recursive functions

The example that absolutely everybody uses every single time recursive functions are brought up. First, give it the initial conditions:

```
In[14] := f[1] = 1
```

And now the general definition:

```
In[15] := f[n_] := n * f[n - 1]
```

Okay, let's see if it works:

```
In[16] := f[5]
```

The := is key. It tells *Mathematica* to not try and evaluate the right-hand side until you actually ask for the left-hand side. When we actually ask for f[5], it keeps recursing until it hits f[1], which it knows equals 1.

If you're doing heavy-duty computations, you may want your recursive function to remember values it has computed. The way you tell *Mathematica* to do that is like so:

```
In[17] := g[1] = 1
```

```
g[x_] := g[x] = x g[x - 1]
```

Note that I did both those commands in one cell by hitting Enter after the first one, and only hitting Shift-Enter when I wanted the whole thing evaluated.

We can ask for a value and then use ? to see what *Mathematica* thinks of this function.

```
In[18] := g[4]
```

```
In[19] := ?g
```

Let's do general orthogonal polynomials. We know they always satisfy a three-term recurrence, for suitable values of b and l :

```
In[20] := p[-1, x_] = 0
```

```
p[0, x_] = 1
```

```
p[n_, x_] := p[n, x] = (x - b[n-1]) p[n-1, x] - l[n-1] p[n-2, x]
```

As defined this way, we just need to define the b and l functions and we'll get a particular OPS. For instance, if $b[n] = 0$ and $l[n-1] = n$, you get Hermite polynomials.

Other useful functions

`Factor`, `Expand`, and `Simplify` do mostly what you expect. There's two ways to use them:

```
In[21] := Factor[x^2 + 2xy + y^2]
```

```
In[22] := x^2 + 2xy + y^2 //Factor
```

The `//Command` syntax means "apply Command to all the stuff before `//`". You can't use this for commands with more than one parameter.

There's also `FullSimplify`. `FullSimplify` works much harder than `Simplify` and sometimes yields much better results. Sometimes not.

`Expand`...expands an expression. `Collect` gathers up the powers of the variable you specify:

```
In[23] := JacobiP[3, a, b, x]
```

```
In[24] := Expand[JacobiP[3, a, b, x]]
```

```
In[25] := Collect[Expand[JacobiP[3, a, b, x]], a]
```

Expanding functions in a series is great when you're working with generating functions. The syntax for the `{x, 0, 6}` is `{variable, expand_about_this_point, degree}`

```
In[26] := Series[(1 - Sqrt[1 - 4x])/(2x), {x, 0, 6}]
```

If the 'H' key on your keyboard doesn't work, you can find Hermite polynomials using the generating function. Here's the fifth Hermite polynomial:

```
In[27] := 5! * Coefficient[Series[Exp[xt + t^2/2], {t, 0, 5}], t^5] //Expand
```

The `Coefficient` command just picks out the specified coefficient – t^5 in this case.

If you're checking to see if your formula agrees with a known result, you can use

Table to work out a bunch of cases. Say we have a formula for Hermite-polynomials-as-matchings, which we derived combinatorially:

```
In[28]:= hermitehmatching[n_, x_] = Sum[Binomial[n, 2k] (2k-1)!! x^(n-2k), {k, 0, Floor[n/2]}]
```

and we want to compare that with a specialization of the usual Hermites:

```
In[29]:= hermitehmatching2[n_, x_] =  
2^(-n/2) HermiteH[n, x * Sqrt[-1/2]] *  
Which[FractionalPart[n/4] == 0, 1,  
FractionalPart[n/4] == 0.25, -Sqrt[-1],  
FractionalPart[n/4] == 0.5, -1,  
FractionalPart[n/4] == 0.75, Sqrt[-1]]
```

We can divide one by the other, factor it, and see if we get one. Let's do the first ten to see if they match (ha!) up:

```
In[30]:= Table[hermitehmatching[n, x]/hermitehmatching2[n, x]//Factor, {n, 0, 10}]
```

Making notes to yourself

Both Mathematica and Maple fancy themselves as complete math-aware word processors. They have this notebook idea going on, which when used properly is quite nice (look at some of the 2374 labs sometime). It's nice to make notes to yourself, which you can do with the Format menu: if you've got some text, go to Format -> Style -> Text and it turn it into text like this right here.

```
In[31]:= Regular prose gets formatted funny when Mathematica thinks you're typing some  
kind of formula.
```

There's also collapsible sections and subsections, which aren't too hard to figure out.

Jane, stop this crazy thing (dealing with the *Mathematica* kernel)

If you've been working for a while and want to start over, you can reset *Mathematica* without quitting. Go to Kernel -> Quit Kernel -> Local. It will ask if you really want to do that. Hit yes if you want all your definitions and variables to be forgotten.

You can clear the value of a particular variable or function with Clear:

```
In[32]:= a = 5
```

```
In[33]:= a
```

```
In[34]:= Clear[a]
```

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

If you evaluated a cell and it's taking too long, you can stop *Mathematica* by going to Kernel -> Abort Evaluation.

Finally, if you want to clear out all the gobbledygook that has been output to the screen, use Kernel -> Delete All Output. It only removes output; variable definitions are remembered. This is useful when emailing files, since it reduces the size of the notebook.

Congratulations if you recognized the Jetsons reference of the title.

Getting help

The help browser is fairly nice. Go to Help -> Help Browser and you'll get a window with a search box. Type in what you're looking for and it will look up things in its database. Together with Google, it's not too difficult to figure things out in *Mathematica*.

Now let's switch to Maple and see what happens when Canadians are running the show

Dan Drake
drake@math.umn.edu