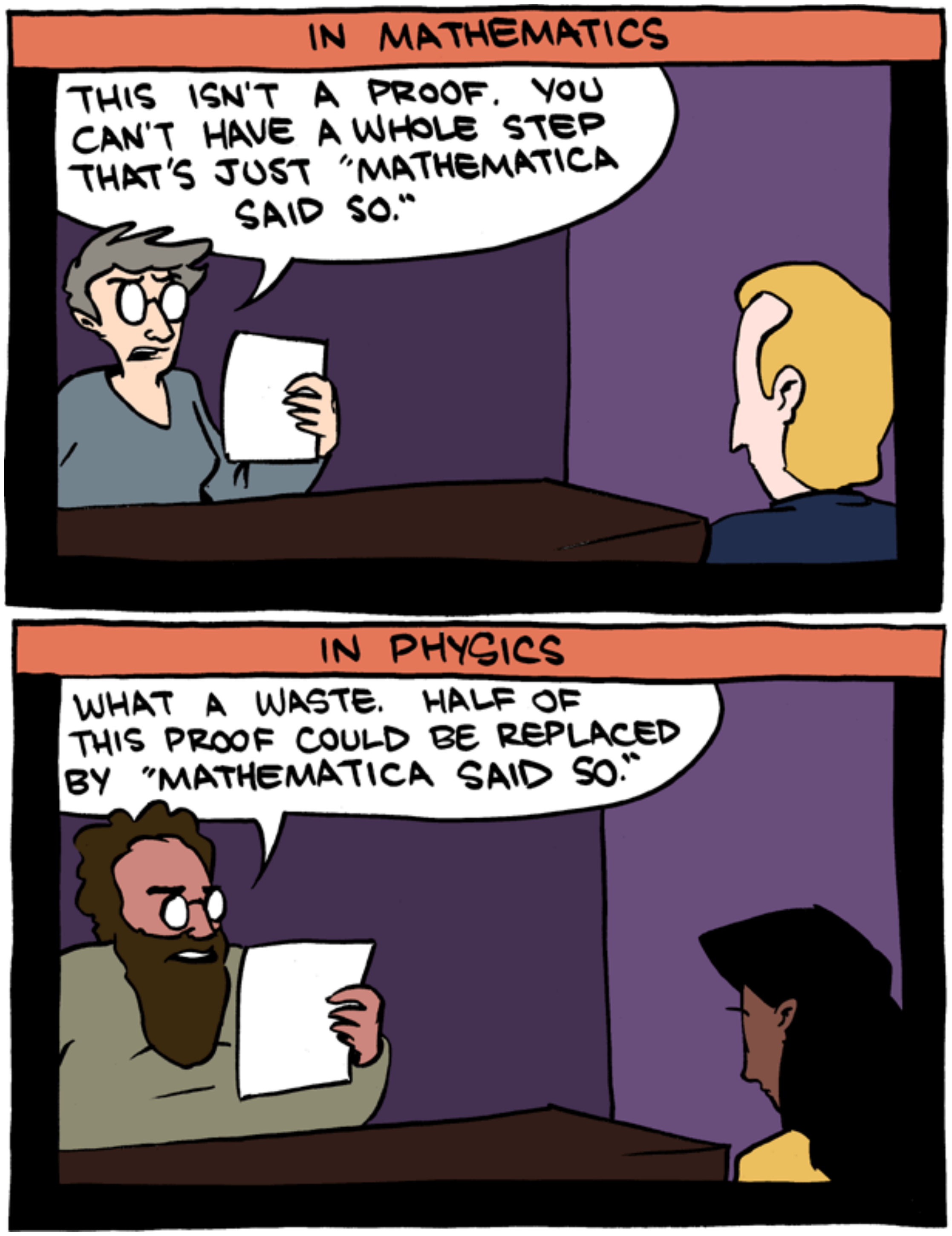
How to Mathematica

A practical guide for using Mathematica in problem sets

Ben Bartlett - Ruddock '17

2017 Ruddock Upperclassman Workshop: 24 April, 9pm

Contents:


IN MATHEMATICS
THIS ISN'T A PROOF. YOU CAN'T HAVE A WHOLE STEP THAT'S JUST "MATHEMATICA SAID SO."
IN PHYSICS
WHAT A WASTE. HALF OF THIS PROOF COULD BE REPLACED BY "MATHEMATICA SAID SO."

0. Wolfram documentation

Mathematica has incredibly well-documented functionality. Typing any command in an input cell and hovering over the text with your mouse will bring up a dropdown menu linking to the documentation for that command. The Wolfram Language Documentation Center also has a series of excellent video tutorials that cover a lot of introductory topics to using Mathematica.

I. Formatting a notebook

Mathematica is frequently used as a scratchpad for calculations, producing messy output that needs to be inserted into a more cleanly typed problem set. However, Mathematica documents can formatted with text, images, and other media to produce well-organized content that is suitable to turn in as a problem set!

Cell styles

Using Format -> Style, (Alt + number on Windows or Cmd + number on Mac) you can change the style of a cell to make a title, subtitle, section, subsection, or text. There are a lot of options for this, but the only ones really worth remembering are (replace Alt with Cmd for Mac):

- Title: Alt + 1 (Cmd + 1 on Mac)
- Chapter: Alt + 2
- Section: Alt + 4
- Subsection: Alt + 5
- Text: Alt + 7

Cell styles as folders

Mathematica also groups cells by descending section order. This allows you to collapse entire sections that you don't need to look at. If you hover over the extended right bracket "]" spanning this section, it will turn blue. Double click it to collapse the section, and double click again to expand it again.

Try it yourself

Try changing the style of this cell to a subsection!

Math input

One of the first things that turns new people off from Mathematica is the ugly way that expressions can look when you write them down. For example,

```
In[1]:= 2 * Sqrt[2] / 9801 *
         Sum[(4 alpha) ! * (1103 + 26 390 alpha) / ((alpha !) ^4 * 396^(4 alpha)), {alpha, 0, Infinity}]
```

Out[1]= $\frac{1}{\pi}$

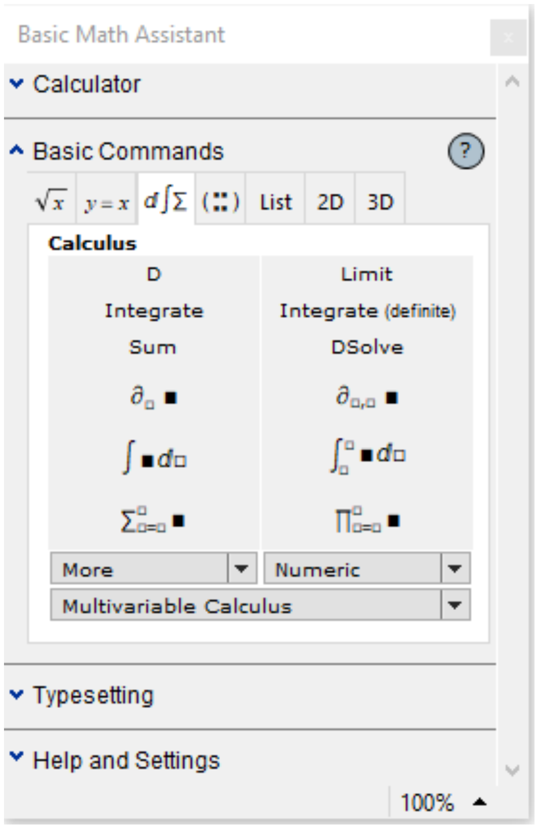
is not the most easily readable expression. However, using the correct keyboard input, you can write this much more nicely:

In[2]:= $$\frac{2\sqrt{2}}{9801}\sum_{\alpha=0}^{\infty}\frac{(4\,\alpha)\,!\,\left(1103+26\,390\,\alpha\right)}{(\alpha\,!)^4\,396^{4\,\alpha}}$$

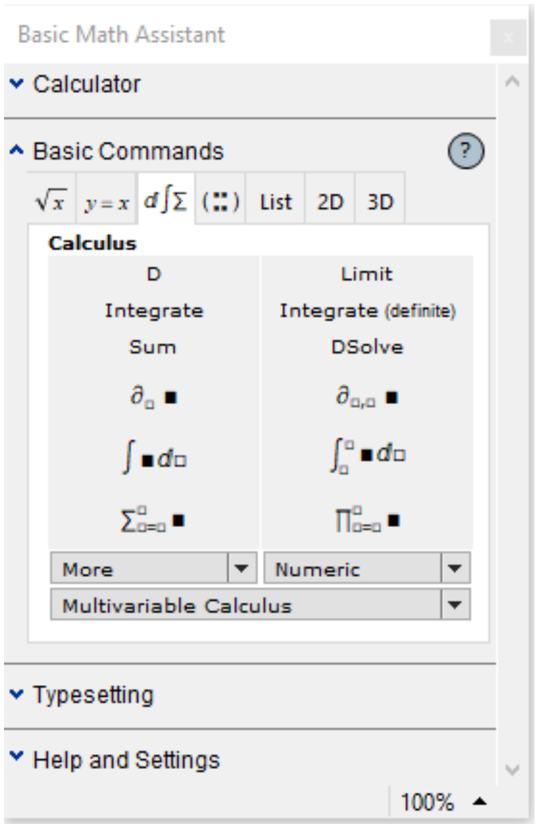
Out[2]= $$\frac{1}{\pi}$$

If you’re just starting out using Mathematica, you might find the math palettes (Palettes -> Basic Math Assistant) useful.

In[3]:=


Basic Math Assistant
Calculator
Basic Commands
?
√x
y = x
d∫Σ
(⁘)
List
2D
3D
Calculus
D
Limit
Integrate
Integrate (definite)
Sum
DSolve
More
Numeric
Multivariable Calculus
Typesetting
Help and Settings
100%

Out[3]=


Basic Math Assistant
Calculator
Basic Commands
?
√x
y = x
d∫Σ
(⁘)
List
2D
3D
Calculus
D
Limit
Integrate
Integrate (definite)
Sum
DSolve
More
Numeric
Multivariable Calculus
Typesetting
Help and Settings
100%

Hovering over any of the buttons should display the keyboard shortcut to enter that expression in Mathematica. Learn these! They'll save a lot of time in the future.

There is a lot of functionality here, but as usual with this guide, I'll cover the ones most useful to know.

Suppressing output

When you evaluate a cell with Shift+enter, you'll see an input and an output. Sometimes, for example if you're defining a block of variables, you don't need this output. In this case, you can suppress the output with a semicolon.

```
In[4]:= myVariable1 = 5
Out[4]= 5

In[5]:= myVariable2 = 10;
```

Greek characters

Any Greek character can be input into a document (in text mode or input mode) by typing esc+letter+esc. For example, esc+p+esc inputs π, esc+G+esc gives Γ, and esc+b+esc is β. This alone can easily reduce a lot of the ugliness associated with Mathematica syntax in a notebook. A few other useful similar shortcuts:

- esc + ii + esc: ⅈ (imaginary number)
- esc + ee + esc: ⅇ (number e)
- esc + inf + esc: ∞

Simple operator input

Many common operators can be written in Mathematica using Ctrl+key (on both Windows and Mac - not Cmd). These are typically pretty easy to remember:

- Ctrl + / gives a fraction: $\frac{1}{42}$
  - Aside: multiplication is done by *, but can be implicitly used by simply putting a space between two variables
- Ctrl + 2 gives a square root: $\sqrt{2}$
- Ctrl + 6 gives an exponent: 2^2
- Ctrl + 9 opens an inline math cell (only in text mode!): $2x+5y^2$
- Ctrl + - creates a subscript: x_i
  - As a general rule, I wouldn't recommend naming any of your variables with subscripts, since Mathematica has inconsistent handling of variables with subscripts in them. I always convert x_i to "xi" in my code.

As usual, there are many others, but these are some of the most commonly used ones.

More complex operator input

Mathematica also has a lot of “template operators” that typeset nicely in a notebook. These are typically entered by esc + (some sensible shortcut) + t + esc. For example:

- esc + intt + esc is an indefinite integral: $\int \blacksquare \, d\square$
- esc + dintt + esc is a definite integral: $\int_{\blacksquare}^{\square} \square \, d\square$
- esc + sumt + esc is a sum: $\sum_{\blacksquare=\square}^{\square} \square$
- esc + prodt + esc is a product: $\prod_{\blacksquare=\square}^{\square} \square$

These can be nested too! For example,

In[6]:= $\sum_{n=0}^{\infty}\left(2^{-n} * \prod_{k=0}^{\infty} 1\right)$

Out[6]= 2

Some operators aren’t templated, but are entered in a similar way:

- esc + dd + esc is a differential symbol: d
  - This one is tricky - this isn’t the same as D[expr, var]. For that, you need to use the ∂ symbol
- esc + pd + esc is a partial differential symbol: ∂
  - You can use this in place of D[expr, var]:

In[7]:= $\partial_x x^3$

Out[7]= $3\, x^2$

Try it yourself

Try entering the Ramanujan approximation of π in a new cell below.

2. Prerequisites: variables, functions, rule replacements, prefixes/suffixes, and lists

Mathematica has most data structures of other programming languages, but for use as a calculation tool for non-CS problem sets, there’s only a few that you have to know.

Variables

Variables in Mathematica can have two “types” (note that Mathematica is completely untyped, so I use this word in the loosest sense): symbolic or defined. Symbolic variables will always be blue or green in the document (they’re green if they’re used for evaluating something, like in plotting), while defined variables, which can be either expressions of symbolic variables or numbers, are black. You can clear defined variables using Clear[var1, var2, var3...]

Using results from previous cells

You can use previously computed results either by storing them to a variable (usually preferred) or by using the % operator, which gets the last Out[] result computed in the notebook. Note: this is the last value computed, not necessarily the previous cell!

```
In[8]:= 2 * 2
Out[8]= 4

In[9]:= % * 3
Out[9]= 12
```

Using %%...% gets the (number of %'s) previous cell:

```
In[10]:= %%
Out[10]= 4

In[11]:= %%%
Out[11]= 4
```

And using %n or Out[n] uses the n'th output:

```
In[12]:= %8
Out[12]= 4
```

Functions

Built-in Mathematica functions always start with a capital character. You can also define your own functions. Mathematica admits two types of "functions". You can define variables to be "expressions" of symbolic variables with an = symbol, or you can define formal functions that use lazy evaluation using a transformation rule on a symbol (x_, for example), and a := symbol. For example:

```
In[13]:= myExpression = ξ^2 + Exp[ξ];
         ξ = 1;
         myExpression
Out[15]= 1 + ⅇ

In[16]:= myFunction[arbitraryVariableName_] := arbitraryVariableName^2 + Exp[arbitraryVariableName];
         myFunction[1]
Out[17]= 1 + ⅇ
```

Pattern matching in functions

You can use existing symbols to create inferred functional patterns. For example, if I wanted to define the tensor product of two vectors (typically denoted by KroneckerProduct[v1, v2]) using the standard ⊗ notation, I could use:

```
In[18]:= a_ ⊗ b_ := KroneckerProduct[a, b]

In[19]:= ψ1 = {1, 0}; ψ2 = {0, 1};
         ψ1 ⊗ ψ2 // MatrixForm
Out[20]//MatrixForm=
         ( 0  1 )
         ( 0  0 )
```

Rule replacements

Mathematica is a functionally-styled programming language, and has nice rule replacements. You can replace any symbol in an expression with another expression by using a rule replacement: “/.”

```
In[21]:= x /. x → 4

Out[21]= 4

In[22]:= Clear[ξ];
         myExpression /. ξ → 1

Out[23]= 1 + ⅇ
```

Prefixes and suffixes

These aren’t terribly important, but can save you time typing and make your input look more nicely formatted. You can add a prefix or suffix to any function by using the @ and // operators, respectively. These work with any function that takes only a single argument. For example:

```
In[24]:= N@π

Out[24]= 3.14159

In[25]:= π // N

Out[25]= 3.14159
```

In[26]:= $x^2 + 2x + 1$ `// FullSimplify`

Out[26]= $(1 + x)^2$

Lists

Lists in Mathematica are simply collections of items (that don’t have to be of the same type) and are handled similarly to Python, with a few exceptions. They are entered using curly braces: {item1, item2, item3}.

```
In[27]:= myList = {1, 2, π, 3, 4, 5 "hello"};
```

Lists are 1-indexed (important!) and can be sliced with the [[;;]] operators:

```
In[28]:= myList[[1]]

Out[28]= 1
```

```
In[29]:=  myList[[1 ;; 3]]
Out[29]=  {1, 2, π}

In[30]:=  myList[[1 ;; 5 ;; 2]]
Out[30]=  {1, π, 4}
```

You can also generate lists with a Table[] function:

```
In[31]:=  Table[x^2, {x, 5}]
Out[31]=  {1, 4, 9, 16, 25}

In[32]:=  Table[Sin[x], {x, 0, π, π/8}]
```

Out[32]= $\left\{0, \mathrm{Sin}\left[\frac{\pi}{8}\right], \frac{1}{\sqrt{2}}, \mathrm{Cos}\left[\frac{\pi}{8}\right], 1, \mathrm{Cos}\left[\frac{\pi}{8}\right], \frac{1}{\sqrt{2}}, \mathrm{Sin}\left[\frac{\pi}{8}\right], 0\right\}$

There are a lot of other list manipulation functions available in the documentation.

3. Commonly-used things in Mathematica

Mathematica can be an indispensable tool for use in your problem sets. Although the language has basically everything you could ever think of built in as dedicated functions, you’ll find yourself using a handful of functions fairly often. I’ll cover these below.

Copy results to LATEX

If you use TEX to type up your problem sets, you can copy results from Mathematica output by selecting the desired output and selecting “Copy As -> LATEX".

Simplifying expressions with FullSimplify[]

There’s a lot of built-in simplification functions and expression manipulation functions (Expand[], Factor[], Together[], etc.), but the go-to function is always FullSimplify, which will get you the result you want about 80% of the time, anecdotally.

```
In[33]:=  FullSimplify[x^2 + 2 x + 1]
Out[33]=  (1 + x)^2

In[34]:=  FullSimplify[Cosh[x] - Sinh[x]]
Out[34]=  e^-x
```

You can also use FullSimplify to verify solutions satisfy an equation

```
In[35]:=  soln = Solve[x + 2 Exp[x] == 1, x] // First // Quiet
Out[35]=  {x → 1 - ProductLog[2 e]}
```

```
In[36]:= 1 == x + 2 Exp[x] /. soln // FullSimplify
Out[36]= True
```

Solving equations with Solve[]

Solve[] lets you solve an algebraic equation or system of equations for possible solutions. Note that when using Solve[] (or any other similar function, like NSolve, DSolve, etc), you need to use the test equality operator == rather than the assignment operator =, just like you would in Python. Solve[] gives a list of rule replacements, not a list of solutions, so if you want to assign something to a value from Solve, you need to destructure the list (First[] and Last[] are frequently helpful here) and apply the corresponding rule replacement to isolate the variable.

In[37]:= `Solve[`$x^2 + a x + 1 == 0$`, x]`

Out[37]= $\left\{\left\{x \to \frac{1}{2}\left(-a-\sqrt{-4+a^2}\right)\right\}, \left\{x \to \frac{1}{2}\left(-a+\sqrt{-4+a^2}\right)\right\}\right\}$

```
In[38]:= Solve[{x + y == 0, x - y == 2}, {x, y}]
Out[38]= {{x → 1, y → -1}}
```

In[39]:= `solution = Solve[`$5 x == \frac{1}{2}$`, x]`

Out[39]= $\left\{\left\{x \to \frac{1}{10}\right\}\right\}$

```
In[40]:= x /. solution // First
```

Out[40]= $\frac{1}{10}$

In[41]:= `variableToAssign = blah /. Solve[`$\text{blah}^2 == 4$`, blah] // First`

Out[41]= -2

Solving differential equations with DSolve[]

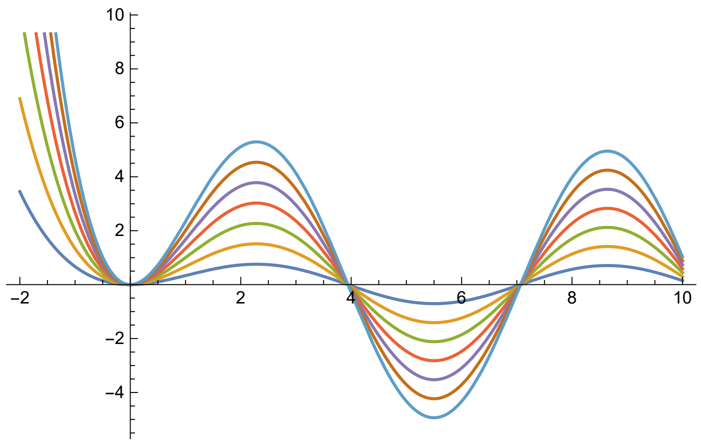
DSolve[] works very similarly to Solve[], except that it requires initial conditions in some cases. (As of Mathematica 10, there’s also a handy function to retrieve the solutions, rather than rule replacements, from DSolve, which is the wrapper function DSolveValue[].) When typing the equations, use ‘ to denote derivatives with respect to the independent variable.

```
In[42]:= DSolve[y'[x] + y[x] == a Sin[x], y[x], x]
```

Out[42]= $\left\{\left\{y[x] \to e^{-x} C[1] + \frac{1}{2} a \left(-\text{Cos}[x] + \text{Sin}[x]\right)\right\}\right\}$

```
In[43]:= sol = DSolveValue[{y'[x] + y[x] == a Sin[x], y[0] == 0}, y, x];
         Plot[Evaluate@Table[sol[x] /. {a → i}, {i, 7}], {x, -2, 10}]
```

Out[44]=



```
In[45]:= DSolve[{
           x''[t] == - k x[t],
           x[0] == 1,
           x'[0] == 0
          }, x[t], t] (*solve for a spring released from rest*)
```

Out[45]= $\{\{x[t] \to \text{Cos}[\sqrt{k}\ t]\}\}$

```
In[46]:= eqns = {
            y'[x] == x^2 y[x],
            z'[x] == 5 z[x]
           };
         initialConditions = {
            y[0] == 1,
            z[0] == 5
           };
         DSolve[Join[eqns, initialConditions], {y[x], z[x]}, x]
```

Out[48]= $\{\{y[x] \to e^{\frac{x^3}{3}},\ z[x] \to 5\ e^{5\,x}\}\}$

Free-form and Wolfram|Alpha input

As of Mathematica 8, you can use “free-form input” to express what you want to do in natural language. Mathematica is generally pretty good at figuring out what you’re trying to do. It also allows an extension of this: free-form Wolfram|Alpha input.

To use free-form input, press “=” in any blank code cell to switch it to free-form mode. Pressing “=” again will switch it to Wolfram|Alpha input, and pressing “=” a third time cycles it back to normal input. The use of both input styles is shown below.

```
In[49]:=  integrate 5x^2 from x=2 to x=5
          Integrate[5*x^2, {x, 2, 5}]

Out[49]= 195
```

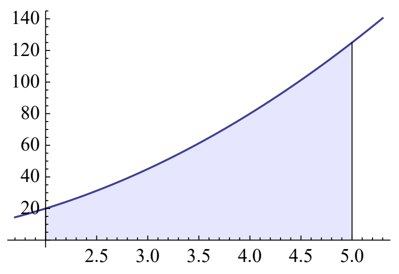
In[50]:= **integrate 5x^2 from x=2 to x=5**

Definite integral:

Step–by–step solution

$$\int_2^5 5\,x^2\,dx = 195$$

Visual representation of the integral:

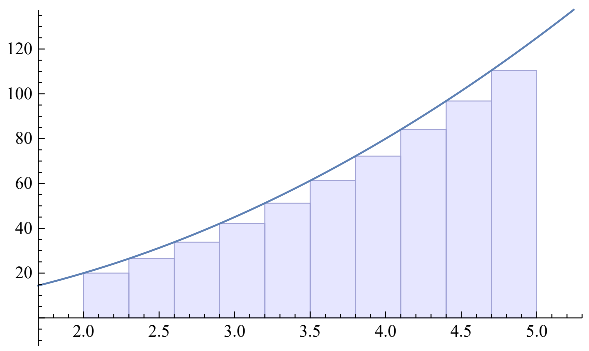


Riemann sums:

More cases

left sum	$\frac{15\,(26\,n^2-21\,n+3)}{2\,n^2} = 195 - \frac{315}{2\,n} + O\left(\left(\frac{1}{n}\right)^2\right)$

(assuming subintervals of equal length)



integral: 195
Riemann sum: 179.475
error: 15.525

number of subintervals

summation method [✓ left endpoint] midpoint right endpoint

Indefinite integral:

Step–by–step solution

$$\int 5\,x^2\,dx = \frac{5\,x^3}{3} + \text{constant}$$

WolframAlpha

Should I use Mathematica as a Wolfram|Alpha client?

Probably not. Mathematica can be a challenging platform to learn, and it's alright to use Wolfram|Alpha if you don't know how to do something, but, in my experience, Wolfram|Alpha stops being useful for physics problem sets around sophomore year. There are a tremendous number of things Mathematica can do that Wolfram|Alpha simply can't, and you will eventually see some of those things appearing in your problem sets.

Using free-form input in standard input cells

If you press Ctrl+"=" (on both Windows and Mac), you can enter a small portion of free-form input in a portion of a standard input cell. This is extraordinarily useful when working with units! (See later in this section for working with units.) For example,

In[51]:= $\texttt{UnitConvert}\left[\frac{1}{4\pi\,\varepsilon_0} * \frac{(1\,\text{C})^2}{1\,\text{m}}, \texttt{"SI"}\right] \texttt{// N}$

```
Out[51]= 8.98755 GJ
```

Working with units

Mathematica has an excellent and robust units system. The formal command for using units is Quantity[]:

```
In[52]:= Quantity[1, "meters per second"]

Out[52]= 1 m/s
```

However, the easiest way to input units, and the way I almost always use, is with free-form input using Ctrl+"=".

```
In[53]:= 1 m/s

Out[53]= 1 m/s
```

Using UnitConvert[], you can convert some arbitrary expression into any equivalent set of units. Or, more frequently, you'll just want to convert it to SI:

```
In[54]:= UnitConvert[1 m/s , "mph"] // N

Out[54]= 2.23694 mi/h
```

In[55]:= UnitConvert[$h\ c$, "SI"]

Out[55]= $1.9864458 \times 10^{-25}$ m J

You can also look up a huge amount of quantitative data using free form input! For example, using "gravitational constant", "earth mass", and "earth radius" in free-form input boxes, I can calculate the surface acceleration of the Earth:

```
In[56]:= surfaceAcceleration = UnitConvert[ G * Earth (planet) [ mass ] / ( Earth (planet) [ average radius ] )^2 , "m/s^2"]

Out[56]= 9.820 m/s^2
```

Solving numerically with NSolve[] and NDSolve[]

As you probably know, many differential equations are not analytically solvable (and some algebraic equations are gross), so you can use Mathematica's built-in numerical solvers to work with these types of equations. NSolve[] works almost exactly the same as Solve[]:

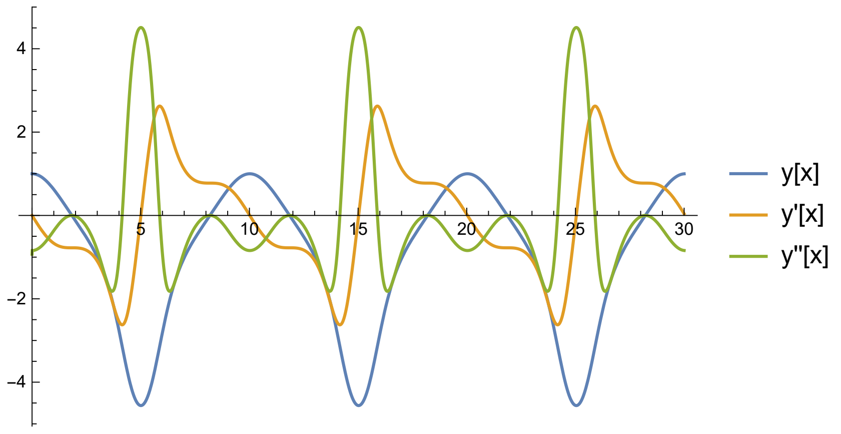
```
In[57]:= NSolve[{x^2 + y^3 == 1, 2 x + 3 y == 4}, {x, y}, Reals]

Out[57]= {{x → 7.93641, y → -3.95761}}
```

When using NDSolve[], you'll get a numerical "InterpolatingFunction" as an output:

```
In[58]:= s = NDSolve[{y''[x] + Sin[y[x]] y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 30}]

Out[58]= {{y → InterpolatingFunction[ Domain: {{0., 30.}} Output: scalar ]}}
```

You can use this InterpolatingFunction exactly as you would a regular function:

```
In[59]:= Plot[Evaluate[{y[x], y'[x], y''[x]} /. s],
          {x, 0, 30}, PlotLegends → {"y[x]", "y'[x]", "y''[x]"}]

Out[59]=
```

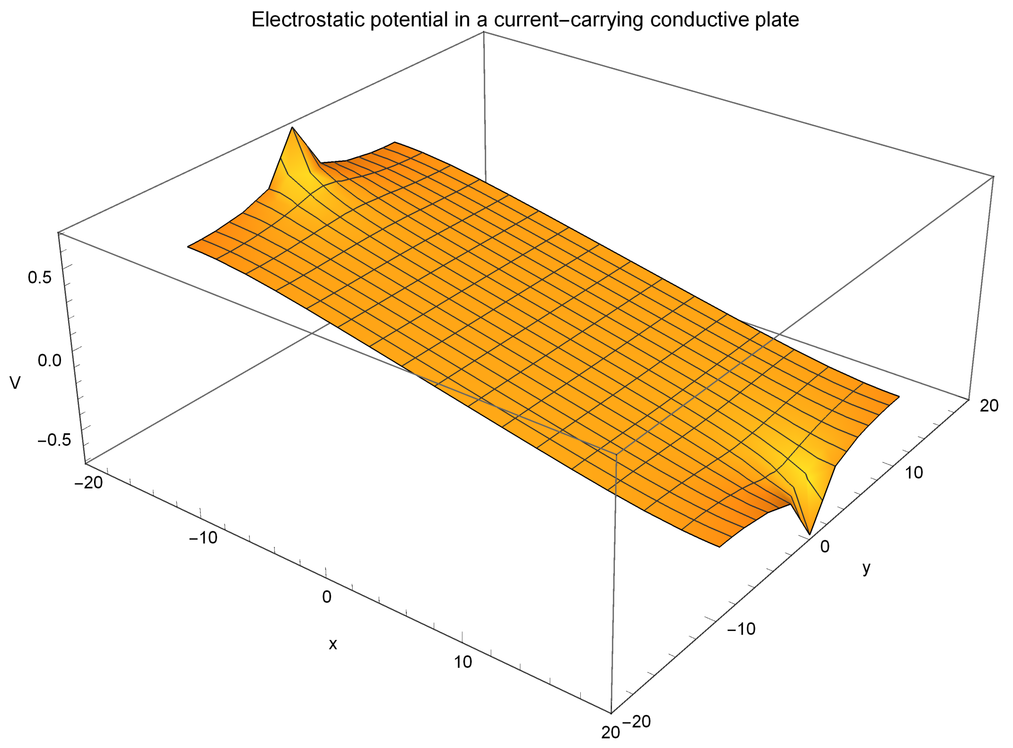


Application: using boundary conditions

As of Mathematica 10, it allows you to input Neumann and Dirichlet boundary conditions when numerically solving differential equations! For example, this is the solution to the potential throughout a metal plate with current passing through it via wires attached at the midpoints of the short end:

In[60]:=
```
ymin = -10; ymax = 10; xmin = -20; xmax = 20; dy = 1; dx = 1;
potentialSoln = NDSolveValue[{
    Laplacian[u[x, y], {x, y}] == NeumannValue[-1/(2 dy), x - xmin < dx && Abs[y] < dy] +
      NeumannValue[1/(2 dy), xmax - x < dx && Abs[y] < dy],
    u[0, y] == 0
   }, u, {x, y} ∈ Rectangle[{xmin, ymin}, {xmax, ymax}],
   Method → {"FiniteElement", "MeshOptions" → {MaxCellMeasure → 0.5}}];
Plot3D[potentialSoln[x, y], {x, y} ∈ Rectangle[{xmin, ymin}, {xmax, ymax}],
 PlotRange → {{xmin, xmax}, {xmin, xmax}}, AxesLabel → {"x", "y", "V"},
 PlotLabel → "Electrostatic potential in a current-carrying conductive plate"]
```

Out[62]=



Working with vectors and matrices

Matrices in Mathematica work the same way they do in Python, except with the changes you've already seen for how vectors work. You can make matrices as lists of lists:

In[63]:=
```
mat = {{1, 1}, {0, 1}};
vec = {1, 0};
```

There are lots of matrix operations you can do. For example, you can transpose a matrix and display it like it would be written on paper:

```
In[65]:= Transpose[mat] // MatrixForm
Out[65]//MatrixForm=
```

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

You can find the eigenvectors and eigenvalues of a matrix:

```
In[66]:= Eigensystem[mat] (*this gives both eigenvectors and associated eigenvalues*)
Out[66]= {{1, 1}, {{1, 0}, {0, 0}}}
```

You can multiply matrices and vectors together with the "." operator:

```
In[67]:= mat.vec
Out[67]= {1, 0}
```

And you can retrieve the magnitude of a vector with Norm[]:

```
In[68]:= Norm[vec]
Out[68]= 1
```

Lots of vector calculus operations will require you to work with vectors and matrices. For example, you can take the curl of a vector field in Mathematica:

```
In[69]:= Curl[{z, -y, -x}, {x, y, z}]
Out[69]= {0, 2, 0}
```

You can also do this using esc+delx+ esc to make a nice pretty template:

In[70]:= $\nabla_{\{x,y,z\}} \times \{y, -x, z\}$

```
Out[70]= {0, 0, -2}
```

Mathematica also allows for very easy coordinate changes. For example, you can take the curl of something in spherical coordinates:

```
In[71]:= Curl[{1, 1, 1}, {r, θ, ϕ}, "Spherical"]
```

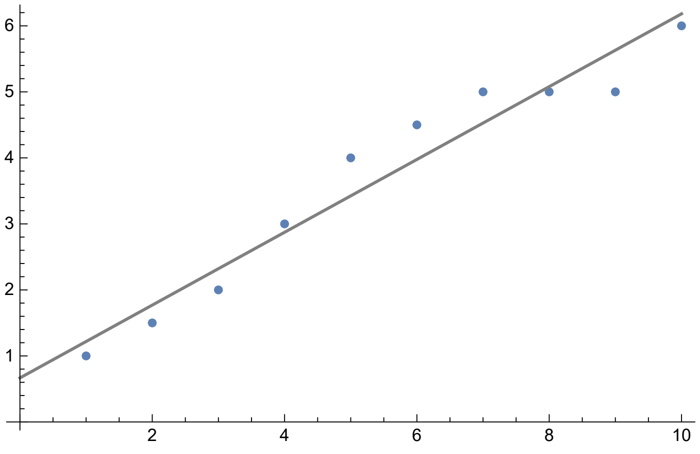
Out[71]= $\left\{\frac{\text{Cot}[\theta]}{r}, -\frac{1}{r}, \frac{1}{r}\right\}$

Fitting functions to data

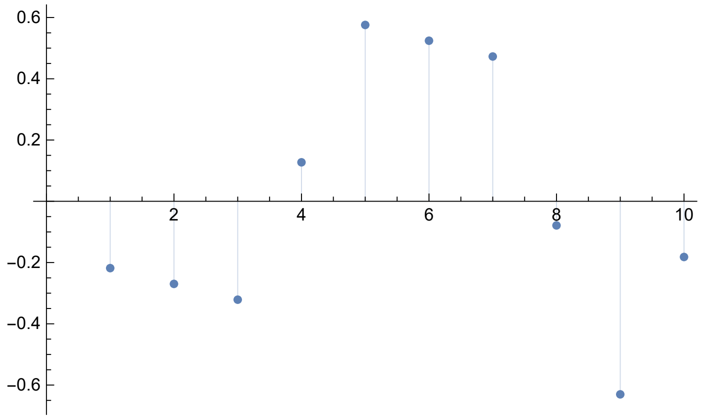
Although it isn't the program's primary purpose, Mathematica includes a lot of statistical manipulation functions. The most commonly used of these, in my experience, is the fitting functions LinearModelFit[] and NonLinearModelFit[]. Let's try these out:

LinearModelFit[]

```
In[72]:= myX = Range[10];
         myY = {1, 1.5, 2, 3, 4, 4.5, 5, 5, 5, 6};
         myData = Transpose[{myX, myY}]

Out[74]= {{1, 1}, {2, 1.5}, {3, 2}, {4, 3}, {5, 4}, {6, 4.5}, {7, 5}, {8, 5}, {9, 5}, {10, 6}}

In[75]:= lm = LinearModelFit[myData, x, x]

Out[75]= FittedModel[ 0.666667 + 0.551515 x ]

In[76]:= (*plot the data and the fitted model*)
         Show[ListPlot[myData], Plot[lm[x], {x, 0, 10}, PlotStyle → Gray]]

Out[76]=
```



```
In[77]:= (*plot the fit residuals*)
         ListPlot[lm["FitResiduals"], Filling → Axis]

Out[77]=
```



We can do more complex stuff with LienarModelFit[] too, like fitting data to multiple variables:

```
In[78]:= myData2 = Flatten[Table[{x, y, Sin[x + y]}, {x, 5}, {y, 5}], 1];
         LinearModelFit[myData2, {x, y}, {x, y}]

Out[79]= FittedModel[ −0.782616 + 0.127533 x + 0.127533 y ]
```

Or fitting data to functions of variables:

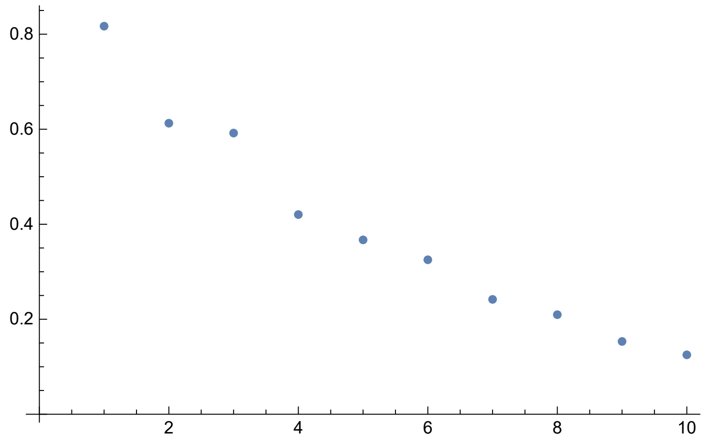
```
In[80]:= LinearModelFit[myData2, {Sin[x], Cos[y]}, {x, y}]

Out[80]= FittedModel[ -0.00760098 + 0.00481437 Cos[y] - 0.244815 Sin[x] ]
```

NonlinearModelFit[]

Sometimes you get data that depends on some functional forms of variables. For example, if we considered “noisy” radioactive decay:

```
In[81]:= decayData = Table[Exp[-t/5] (1 + RandomReal[{-1/10, 1/10}]), {t, 10}];
         ListPlot[decayData]
```

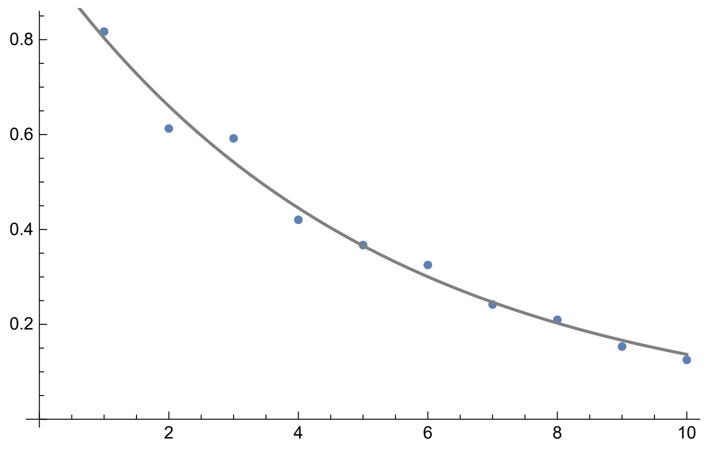
Out[82]=



You can fit this type of data to a model using NonlinearModelFit[] and specifying the desired form of the data along with the constants to solve for. For example, in this case, we know decay rate follows $A * e^{-t/\tau}$, so we can do:

```
In[83]:= nlm = NonlinearModelFit[decayData, A * Exp[-t/τ], {A, τ}, t]

Out[83]= FittedModel[ 0.978302 e^(-0.196818 t) ]
```

In[84]:=
```
Show[ListPlot[decayData], Plot[nlm[t], {t, 0, 10}, PlotStyle → Gray]]
```

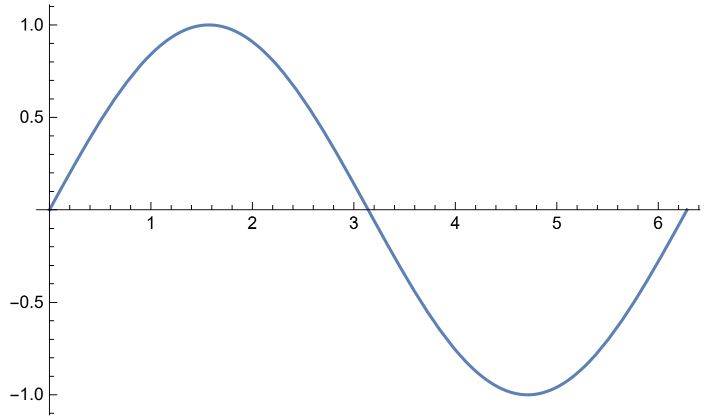
Out[84]=


4. Plotting relationships and data in Mathematica

Plotting symbolic relationships with Plot[]

Mathematica’s plotting functionality is the easiest to use and most robust of any library I’ve come across. Plot[] is your bread-and-butter function here, and can be used to plot arbitrary symbolic relationships:

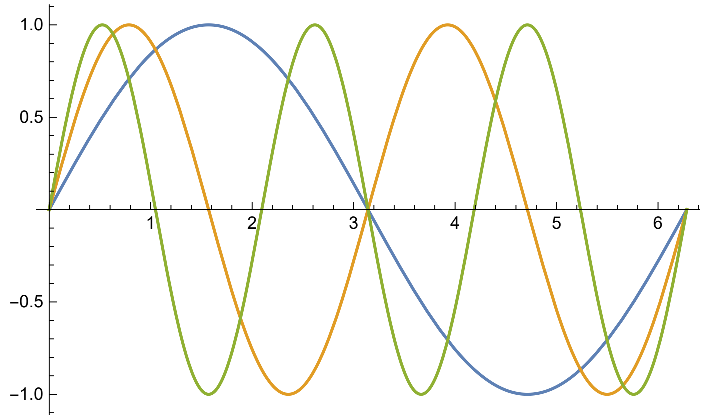
In[85]:=
```
Plot[Sin[x], {x, 0, 2 π}]
```

Out[85]=


You can also use Plot[] with lazily-defined (:=) functions:

```
In[86]:= f1[x_] := Sin[x];
         f2[x_] := Sin[2 x];
         f3[x_] := Sin[3 x];
         Plot[{f1[x], f2[x], f3[x]}, {x, 0, 2 π}]
```

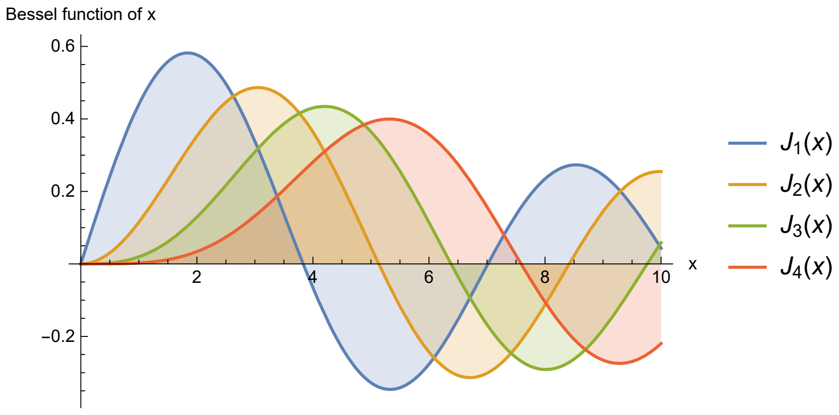
Out[89]=



You can add legends with the option PlotLegends, axes labels with AxesLabel, and many other possibilities available in the documentation.

```
In[90]:= Plot[Evaluate[Table[BesselJ[n, x], {n, 4}]], {x, 0, 10}, Filling → Axis,
          AxesLabel → {"x", "Bessel function of x"}, PlotLegends → "Expressions"]
```

Out[90]=

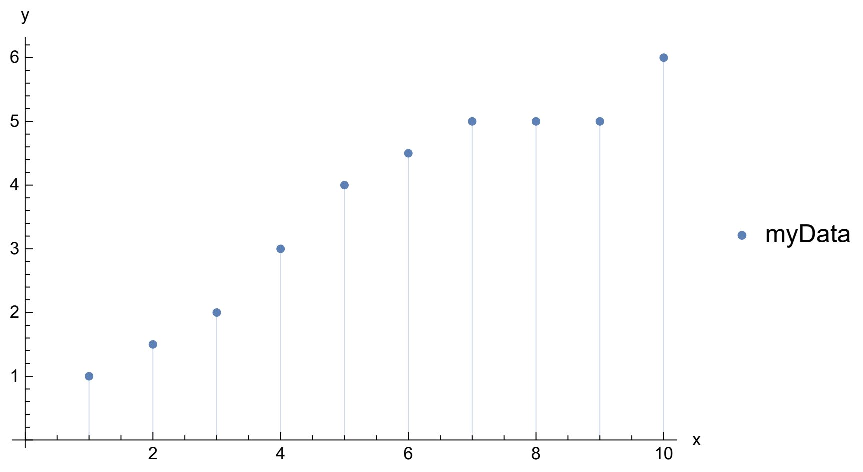


Plotting lists with ListPlot[]

ListPlot[] works similarly to plot, but on lists of data rather than functional relationships. For example:

```
In[91]:= ListPlot[myData, Filling → Axis, AxesLabel → {"x", "y"}, PlotLegends → {"myData"}]
```

Out[91]=

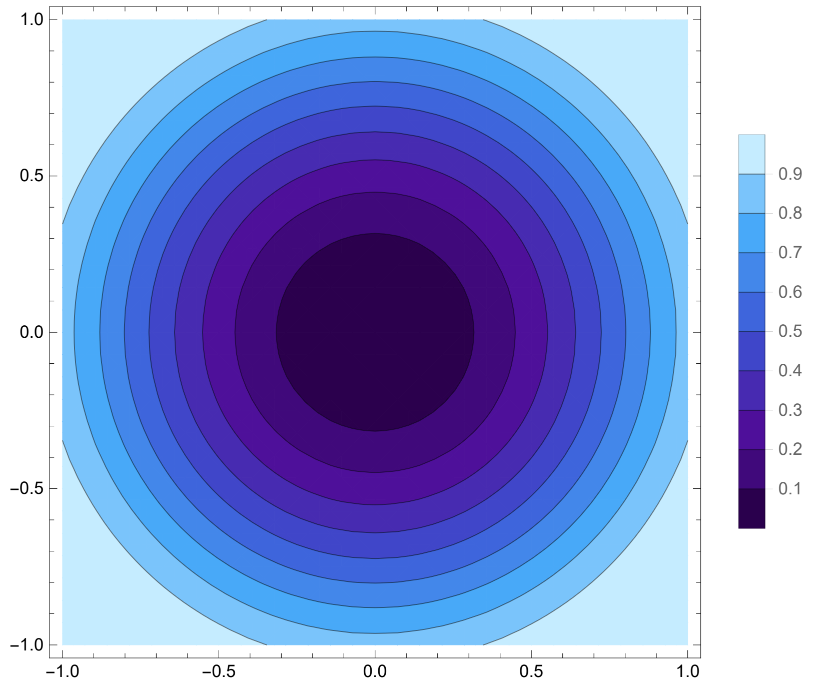


Other common types of plots

Other useful plot functions include ContourPlot[], Plot3D[], ListContourPlot[], and ListPlot3D[]. Extensive documentation is available for these functions, so I'll just demonstrate one of each:

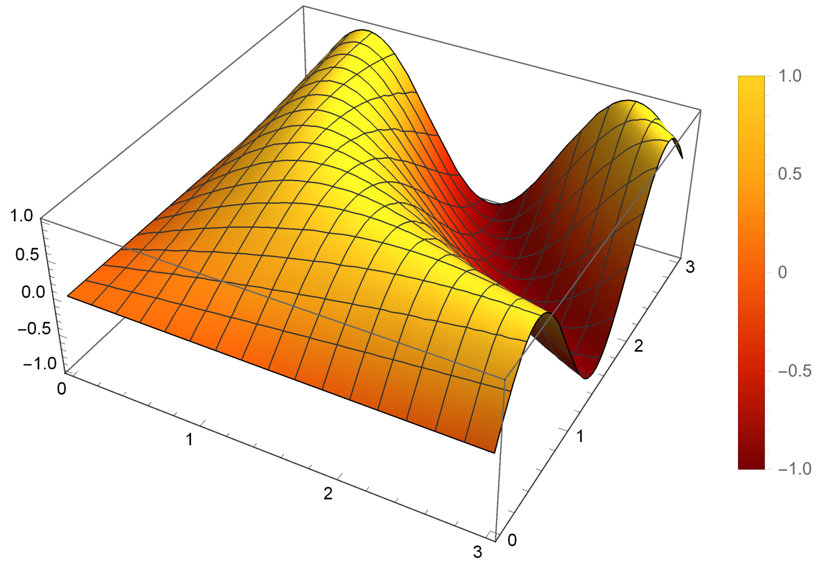
```
In[92]:= ContourPlot[Sin[x^2 + y^2], {x, -1, 1}, {y, -1, 1},
          ColorFunction → "DeepSeaColors", PlotLegends → Automatic]
```

Out[92]=



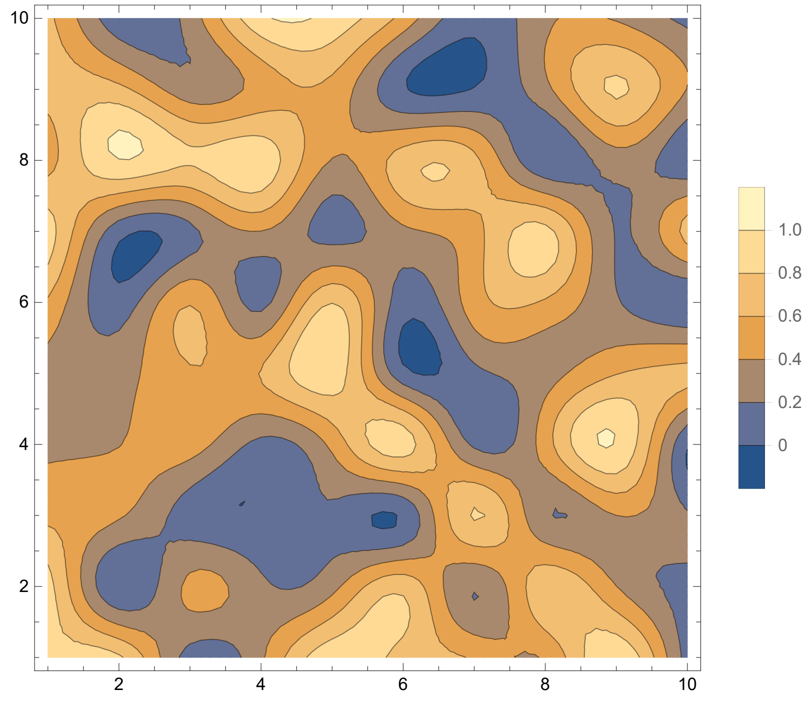
```
In[93]:= Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, ColorFunction → "SolarColors", PlotLegends → Automatic]
```

Out[93]=

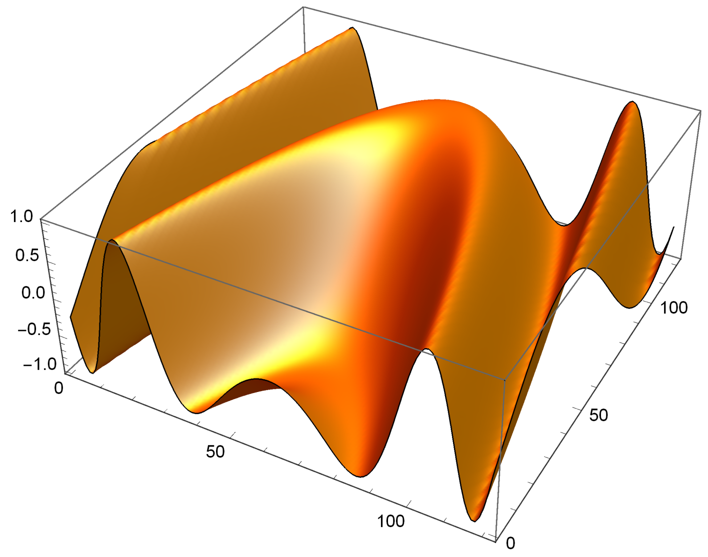


```
In[94]:= ListContourPlot[RandomReal[1, {10, 10}], InterpolationOrder → 3, PlotLegends → Automatic]
```

Out[94]=



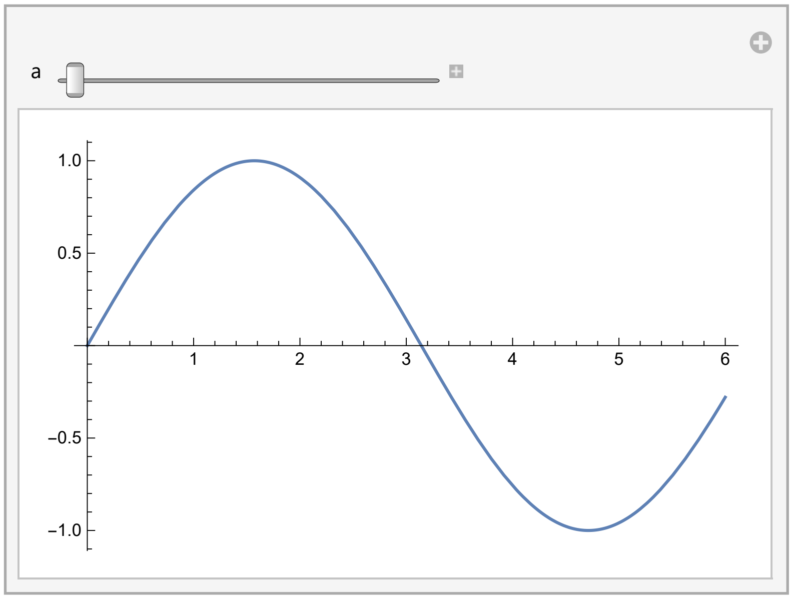
In[95]:=
```
ListPlot3D[Table[Sin[j^2 + i], {i, -3, 3, 0.05}, {j, -3, 3, 0.05}],
 PlotStyle → Directive[Orange, Specularity[White, 20]], Mesh → None]
```

Out[95]=


Using Manipulate[]

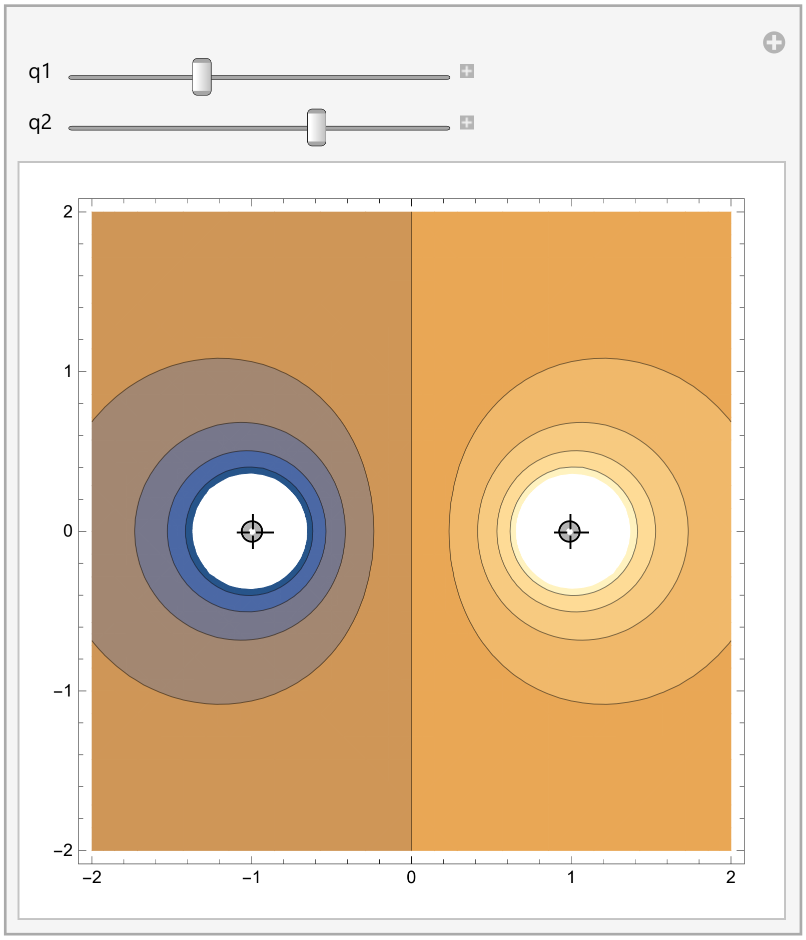
This is one of my favorite features of Mathematica: using Manipulate[], you can create interactive plots that allow you to explore the behavior of something as a function of some input variable(s). For example:

In[96]:=
```
Manipulate[Plot[Sin[x (1 + a x)], {x, 0, 6}], {a, 0, 2}]
```

Out[96]=


You can do a lot of really cool things with this. For example, here's a plot of the equipotential lines from two charge sources: (you can adjust the sliders to adjust the magnitude of the charges and drag the positions of the charges around in the plot)

In[97]:=
```
Manipulate[
 ContourPlot[q1/Norm[{x, y} - p[[1]]] + q2/Norm[{x, y} - p[[2]]], {x, -2, 2}, {y, -2, 2}],
 {{q1, -1}, -3, 3}, {{q2, 1}, -3, 3}, {{p, {{-1, 0}, {1, 0}}}, {-1, -1}, {1, 1}, Locator}]
```

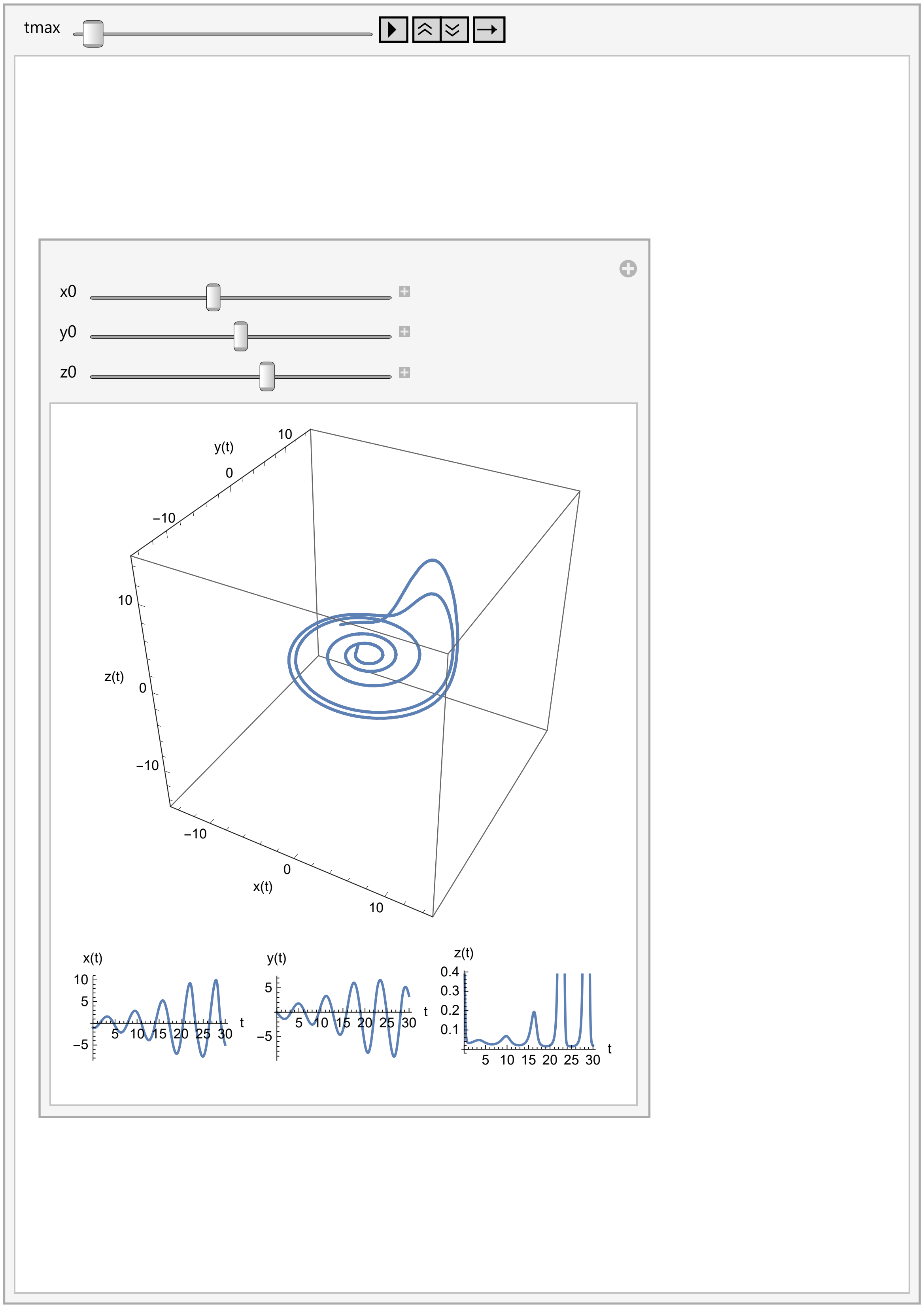
Out[97]=


And here's a manipulable solution to the Rossler attractor (a set of chaotic differential equations) displaying the behavior of all component coordinates:

In[98]:=
```
a = 0.2; b = 0.2; c = 5.7;
RosslerEquations = {
   x'[t] == -z[t] - y[t],
   y'[t] == x[t] + a y[t],
   z'[t] == b + z[t] (x[t] - c)
  };
```

```
In[100]:= Tmax = 40;
          rad = 15;
          animation = Animate[
             Manipulate[
              Module[{sol = NDSolve[Join[RosslerEquations, {x[0] == x0, y[0] == y0, z[0] == z0}],
                  {x[t], y[t], z[t]}, {t, 0, Tmax + 1}]},
               pplot = ParametricPlot3D[{x[t], y[t], z[t]} /. sol, {t, 0, tmax},
                 AxesLabel → {"x(t)", "y(t)", "z(t)"},
                 PlotRange → {{-rad, rad}, {-rad, rad}, {-rad, rad}}];
               xplot = Plot[{x[t]} /. sol, {t, 0, tmax}, AxesLabel → {"t", "x(t)"}];
               yplot = Plot[{y[t]} /. sol, {t, 0, tmax}, AxesLabel → {"t", "y(t)"}];
               zplot = Plot[{z[t]} /. sol, {t, 0, tmax}, AxesLabel → {"t", "z(t)"}];
               GraphicsGrid[{
                 {pplot, SpanFromLeft, SpanFromLeft},
                 {SpanFromAbove, SpanFromBoth, SpanFromBoth},
                 {SpanFromAbove, SpanFromBoth, SpanFromBoth},
                 {xplot, yplot, zplot}
                }]
              ], {{x0, -1}, -5, 5}, {{y0, 0}, -5, 5}, {{z0, 1}, -5, 5}],
             {tmax, 30, Tmax}, AnimationRunning → False] // Quiet
```

Out[102]=


tmax
x0
y0
z0
y(t)
10
0
−10
z(t)
x(t)
x(t)
y(t)
z(t)
t
5
10
15
20
25
30
0.4
0.3
0.2
0.1
−5