Plotting functions of more than one variable with Mathematica

Physics 3510, Weber State University

This tutorial assumes that you are already somewhat familiar with Mathematica. So, for example, you should know how to express basic arithmetic operations, and understand the difference between parentheses `()`, square brackets `[]`, and curly braces `{}`, and remember that the names of variables and functions are case-sensitive, but the names of all built-in functions start with capital letters.

You should also already know how to plot a simple function of one variable, for example,

```
Plot[x^3-2x, {x, -2, 2}, Frame->True, PlotStyle->{Red, Thick}]
```

where `{x, -2, 2}` is a common list construction that means (in this context) "for values of x ranging from -2 to 2," while `Frame` and `PlotStyle` are two of the many options that can modify the appearance of the plot. Please type this instruction into Mathematica now, to warm up your fingers and make sure that your Mathematica system is working. Remember that you need to use shift-enter to send the instruction to the Mathematica kernel.

Scalar functions of two variables

Our main goal in this tutorial is to explore ways to plot functions of *two* variables. An example of such a function, which we'll work with for a while, is

```
f = (x+y)^3 - 8(x+y) - 2(x-y)^2
```

where I've given the function a name, `f`, because we'll be using it repeatedly. Again, please enter this line into Mathematica.

The most basic way of plotting a function of two variables is `DensityPlot`:

```
DensityPlot[f, {x, -2, 2}, {y, -2, 2}]
```

After you enter this instruction, you should see a square-shaped plot with the first variable (here `x`) along the horizontal axis and the second variable (here `y`) along the vertical axis. The *values* of the function are represented by colors. Which colors correspond to the lowest function values, and which correspond to the highest?

Just as with `Plot`, there are many options you can use to modify the output of `DensityPlot`. First try the option `ColorFunction->"SunsetColors"` and see how you like the new color scheme. To see a list of all the built-in color schemes, use the Help menu to open the Documentation Center and search for "color schemes". Try a couple of other color functions in your `DensityPlot` and see how you like them. Can you find the name of the default color function? You can even create your own color functions by saying something like this:

```
ColorFunction -> Function[Blend[{Black, Blue, White}, #]]
```

Try it! (The symbol `#` signifies the argument of the function; feel free to look up "pure functions" and `Blend` in the Documentation Center if you're curious about how exactly this syntax works.) If you're not satisfied with blends of named colors like `Blue`, you can define your own color using `RGBColor[r, g, b]`, where each of the three arguments is a number between 0 and 1, for example, `RGBColor[.5, 0, 1]`.

Another common way to visualize a function of two variables is with contour lines, which connect points that have the same function value as on a topographic map. Try changing `DensityPlot` to `ContourPlot` in the previous example, and notice that in addition to the contour lines you still get the colors, but they now go in steps at the contour lines instead of varying smoothly. Also notice that if you hover the mouse cursor over a contour line, you get a little popup note that tells you the function value on that contour. Try the option `Contours->30` to increase the number of contours.

A third way of plotting a function of two variables is with a surface plot, where the function value is plotted on a third axis in three dimensions:

```
Plot3D[f, {x, -2, 2}, {y, -2, 2}]
```

To view the surface from different directions, just drag it with the mouse. You can again use the `ColorFunction` option to change the colors, but to apply a "homemade" `Blend` to the function values you need to change `#` to `#3` in the syntax given above. Notice that the colors are modified by simulated lighting that illuminates the surface from certain directions (which you can also change if you wish). Another useful option for surface plots is `BoxRatios->{1,1,1}` (which you should now try).

Vector functions in two dimensions

So much for *scalar*-valued functions of two variables; now what about a function (often called a "field") whose value at any point is a *vector*?

Although we could start over and define each component of a vector field from scratch, we can also make a vector field by taking the gradient of our scalar function:

```
vx = D[f, x]
vy = D[f, y]
```

This is Mathematica's notation for partial derivatives. (Mathematica actually comes with an add-on package that defines `Grad`, `Div`, and `Curl` functions, but it's not worth the trouble when you're working in Cartesian coordinates.)

To plot a two-dimensional vector field, use `VectorPlot`:

```
VectorPlot[{vx, vy}, {x, -2, 2}, {y, -2, 2}]
```

This evaluates the vector field at points on a rectangular grid, and draws an appropriate arrow centered on each grid point. To change the grid resolution, use the `VectorPoints` option:

```
VectorPoints -> {20, 20}
```

Of course there's also an option to change the colors of the arrows:

```
VectorColorFunction -> Function[Green]
```

(This pure function simply assigns the color `Green` to all vectors; if you want to get fancy, you can assign colors based on the vectors' components or lengths or locations.)

`VectorPlot` scales the lengths of the arrows automatically. Often you'll want to omit a few of the biggest vectors from the plot, in order to enlarge the rest of them. You can do this with the `VectorScale` option:

```
VectorScale -> {Automatic, Automatic, Function[If[#5>40, 0, #5]]}
```

Here `40` is the cutoff applied to the vector magnitudes. Try some other values of this cutoff, and also try replacing the first `Automatic` with a number like `0.1`; this is the maximum size at which the vectors are drawn, in units of the full size of the plot. If you want to fully understand `VectorScale`, look it up in the Documentation Center.

Sometimes you might want to overlay a vector plot on a density or contour plot of a related scalar function. While Mathematica does provide a combined `VectorDensityPlot` function, you'll have more control if you create the two plots separately and then combine them using `Show`. For example:

```
cplot = ContourPlot[f, {x, -2, 2}, {y, -2, 2}];
vplot = VectorPlot[{vx, vy}, {x, -2, 2}, {y, -2, 2}];
Show[cplot, vplot]
```

Notice that I've given names to the two individual plots so I can refer to them in the third line. Please try this example and then dress it up using several of the options described above.

Functions of three variables

Visualizing a function of three variables is considerably harder, because there's no perfect way to prevent the "stuff in front" from obscuring your view of the "stuff in back". Mathematica doesn't even have a three-dimensional version of `DensityPlot` (although it's possible to use brightness and transparency to make 3-D density plots, as in some of the applets at falstad.com).

But Mathematica does provide a `ContourPlot3D` function for three-dimensional contour plots, as well as a `VectorPlot3D` function for 3-D vector plots. Both of these plot types work best on functions (fields) that are large within some central region, dying out with distance. Fortunately, many examples in electromagnetism have that property.

Rather than trying to describe all the options and pitfalls of these 3-D plots in detail, I'll just give you the following example to try out:

```
f3 = 1/Sqrt[x^2 + y^2 + (z-1)^2] + 1/Sqrt[x^2 + y^2 + (z+1)^2]

ContourPlot3D[{f3==1.5, f3==2, f3==2.5, f3==3},
    {x, -2, 2}, {y, -2, 2}, {z, -2, 2},
    Mesh->None, ContourStyle->Directive[Opacity[0.5]]]

v3x = D[f3, x]; v3y = D[f3, y]; v3z = D[f3, z];

VectorPlot3D[{v3x, v3y, v3z}, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]
```

Try modifying this example to see the effects of the various choices I've made, and use the Documentation Center to learn more.

Because 3-D visualization is inherently difficult, a better approach is often to view just one two-dimensional slice at a time, using the various 2-D visualization techniques described above. So I'll leave you with this final, quick example to try:

```
slice = f3 /. y -> 0

ContourPlot[slice, {x, -2, 2}, {z, -2, 2}]
```

And with that, I hope you now feel ready to plot and visualize the functions and fields that we will encounter in electromagnetic theory.