Plotting wavefunctions with Mathematica

Daniel V. Schroeder, Physics Department, 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 differences 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[Exp[-x^2], {x, -3, 3}, PlotRange->{0, 1.5}, PlotStyle->{Red, Thick}]
```

where `{x, -3, 3}` is a common list construction that means (in this context) "for values of x ranging from -3 to 3," while `PlotRange` 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.

Complex functions

A quantum-mechanical wavefunction is complex, with so-called real and imaginary components. The easiest way to plot such a function is to simply plot the two components separately on the same set of axes. To do this you provide Mathematica with a list of the two component functions, in curly braces, as in this example:

```
Plot[{Cos[x], Sin[x]}, {x, 0, 10}, PlotStyle->{{Blue, Thick}, {Red, Thick}}]
```

Notice that you also need an extra set of curly braces in the `PlotStyle` specification, to say which styles apply to which function. Try this instruction as written, but also check to see what happens if you use simply say `PlotStyle->{Red, Thick}` as before.

Alternatively, you can just define the full complex-valued function and tell Mathematica to break it into real and imaginary parts. You do this with the `Re` and `Im` functions:

```
psi = Exp[I*x];
Plot[{Re[psi], Im[psi]}, {x, 0, 10}]
```

Notice that the Mathematica symbol for $\sqrt{-1}$ is capital `I`.

A prettier way to plot a complex-valued function is to plot the magnitude (or sometimes the squared magnitude) along the vertical axis and then use color hues to fill the area below according to the phase. To extract the magnitude and phase you use the Mathematica functions `Abs` and `Arg`, respectively. Here's a complete example using a Gaussian wavepacket:

```
psi = Exp[-x^2]*Exp[10*I*x];
Plot[Abs[psi], {x, -3, 3}, PlotPoints->300, Filling->Axis,
  ColorFunction->Function[x, Hue[Arg[psi]/(2Pi)]],
  ColorFunctionScaling->False]
```

The important part of this instruction is the `ColorFunction`, which we assign to a function that takes `x` to a color hue determined by the phase of `psi`. We need to divide the phase by 2π because the `Hue` function cycles through all the color hues when its argument changes by 1. The `PlotPoints` option merely tells Mathematica to be sure to plot enough points to show all the intermediate colors. In any case, please try this example now, and try making some changes to the definition of `psi`. To adjust the appearance of the plot, try inserting the options `AxesOrigin->{-3, 0}` and `AspectRatio->0.3`.

Functions of two variables

What if a wavefunction depends on *two* variables? One way to visualize such a function is to plot the two independent variables along the horizontal and vertical axes, and then use color or brightness to indicate the function value. In Mathematica this is called a `DensityPlot`, although other software systems sometimes call it a "heat map" or simply an image. Here is an example of plotting the square of a two-dimensional sinusoidal wavefunction:

```
psi = Sin[2*Pi*x]*Sin[3*Pi*y];
DensityPlot[psi^2, {x, 0, 1}, {y, 0, 1}]
```

When you try this instruction you may notice some irregularities in the plot that result from sampling the function at too few points. To fix this, insert the option `PlotPoints->100`.

You'll also notice that Mathematica draws the density plot using a default color scheme, which you may or may not like. To change the color scheme to a basic white-on-black, insert the following option:

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

Here `z` is merely a dummy variable whose name is arbitrary. (You may notice that the color function works a little differently here than in the `Plot` example above. To see what variables get passed to a `ColorFunction` in different contexts, look it up in Mathematica's Documentation Center.) The `Blend` function can take any number of arguments, so you can get a prettier plot (and often bring out more detail) by using intermediate colors. Try inserting `Blue`, `Red`, and `Yellow` in between `Black` and `White`, all separated by commas. Mathematica defines about two dozen named colors and also lets you create any color you like using either `Hue[h,s,b]` or `RGBColor[r,g,b]` (both with arguments that range from 0 to 1).

If for some reason you're not happy with a simple density plot, Mathematica offers two other options. One is to add contour lines, which you can do by simply changing `DensityPlot` to `ContourPlot` (try it now). A nice feature of `ContourPlot` is that if you hover the cursor over a contour line, you'll see a little popup that tells you the function value along that line. The other option is to make a three-dimensional surface plot; to do this, change `DensityPlot` to `Plot3D`, and remove the `ColorFunction` specification (which you *can* use with `Plot3D`, but it works differently so you would need to look it up to see how to get good results). You can use your mouse (or trackpad) to rotate the 3D plot, to see it from different directions.

Plotting the square of a wavefunction is fairly straightforward because the minimum value is always zero. But when you plot a wavefunction itself, even if it is real, you'll probably want to use different colors to distinguish positive and negative values. Here is an example that works for the `psi` expression defined above:

```
DensityPlot[psi, {x, 0, 1}, {y, 0, 1}, PlotPoints->100,
  ColorFunction->Function[z, Blend[{Cyan, Black, Red}, z]]]
```

I've chosen red to represent positive values and cyan to represent negative values. Black represents zero, and the brightness of either hue increases as the values get farther from zero. Try changing the hues to suit your taste, and try changing `Black` to `White`, to get an image that's more printer-friendly.

The previous example won't work correctly, though, if the range of values being plotted isn't centered on zero. In that case you need to set `ColorFunctionScaling->False`, to tell Mathematica not to automatically map your function values to the full range of colors. Then function values in the range 0 to 1 will be mapped to your color range, with 0.5 mapped to the middle of the range, so you need to manually scale and shift your function accordingly. Here is an example:

```
psi = Sin[Sqrt[x^2 + y^2]]/Sqrt[x^2 + y^2];
DensityPlot[psi/2 + 0.5, {x, -10, 10}, {y, -10, 10},
  PlotRange->All, PlotPoints->100, ColorFunctionScaling->False,
  ColorFunction->Function[z, Blend[{Cyan, Black, Red}, z]]]
```

The most important detail here is that I shifted the function upward by 0.5, to map 0 to my middle color, black. Because my function has a maximum value of 1, I also divided it by 2 to keep that maximum value at 1. If I hadn't divided by 2 then the brightest part of the function would be "overexposed" (which isn't always a bad thing—try it!). Notice also that I had to say `PlotRange->All` to prevent Mathematica from completely chopping off the largest portion of my function (check what happens if you omit this option).

Finally let's plot a *complex*-valued function of two variables, using the full range of color hues to represent the phases. The trick is to use Mathematica's `RegionPlot` function, but use `True` to fill the entire plotted area and let the `ColorFunction` do all the work. This example plots a two-dimensional Gaussian wavepacket:

```
psi = Exp[-(x^2+y^2)]*Exp[10*I*(x-y)];
RegionPlot[True, {x, -3, 3}, {y, -3, 3},
  PlotPoints->100, ColorFunctionScaling->False,
  ColorFunction->Function[{x, y}, Hue[Arg[psi]/(2 Pi), 1, Abs[psi]]]]
```

Notice that I've used the optional second and third arguments of the `Hue` function to set the saturation equal to 1 and the brightness equal to the magnitude of my function. To get a printer-friendly white background, try interchanging these two arguments so the brightness is 1 and the saturation is the function magnitude. Also try multiplying the function by 2 or 1/2, to see what happens when its maximum magnitude isn't equal to 1.