APPM 2460
PLOTTING IN MATLAB

1. Introduction

Matlab is great at crunching numbers, and one of the fundamental ways that we understand the output of this number-crunching is through visualization, or plots. Today we'll learn about the basics of plotting in Matlab.

2. Plotting

2.1. **How does Matlab think about plotting?** It is very important to realize that Matlab does not think about plotting in the same way that you might think of it, or that you may have learned from Mathematica. Matlab can only plot lists of points, which it then connects with a line. Let me repeat that; **Matlab can only plot lists of points. Continuous functions are not used when plotting in Matlab.**

What does this mean? Let's take a look. First, build a vector of x values by typing (either in a script, or in the command window):

```
x = [0,1,2,3]
```

Now, make a list of corresponding y values by entering

```
y = [0,1,4,9]
```

We plot by calling the command

```
plot(x,y)
```

Note that we don't need to name our lists `x` and `y`, it's just clearer to do so right now. In general, they will have more descriptive names.

Take a look at the plot that is generated. See that Matlab essentially plots a piecewise linear function between the points (0,0), (1,1), (2,4) and (3,9). What Matlab does is match up the `x` and `y` vectors to make points. **If the `x` and `y` vectors are different lengths, you will get an error!**

To clarify the points that Matlab is plotting, enter the command

```
plot(x,y,'o-')
```

This third argument is called an "option." We'll play more with options in the next section. You should see the plot shown in Figure 1. The command you just entered tells Matlab to plot circles at the points given, with lines connecting them. **This is what Matlab does; it plots points, and draws lines between them.**

Suppose we wanted to plot a few points, say (0,1), (1,4), and (2,2). How would we plot these points? Well, first we would make a list of all the x-coordinates:

```
x = [0 1 2]
```

and the corresponding list of y-coordinates:

```
y = [1 4 2]
```

We could then call the plot command with the option `'+'` to put crosses at those points:

```
plot(x,y,'+').
```

We'll learn more about options below.

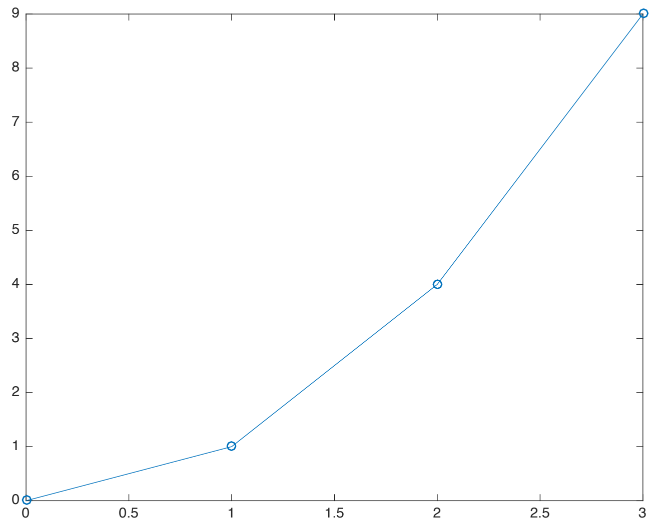

FIGURE 1. A plot of the vectors `[0,1,2,3]` and `[0,1,4,9]`. Matlab can only plot the points you give it! It draws a straight line between these points.

Looking at Figure 1, you probably recognize that this is the function $y = x^2$. It looks chunky on this plot because we move by too large of steps. We can use what we learned about vectors last class to make the first vector have a smaller step size. Note that we could create the same x-vector by typing:

```
x = 0:1:3
```

That is, `x` is the vector obtained by starting at 0, stepping by 1, and ending at 3. Since we determined the step size is too large, we can instead step by .01 using the following code:

```
x = 0:0.1:3
```

Now, we don't want to figure out the square of each element of this vector by hand, so we want a command that tells Matlab to square each element automatically. It's important to realize that `x` is a vector, and we don't have a way to square a vector. What we want to do instead is to square *each element* of `x`, to obtain the same element of `y`. For example, we want the first element of `y` to be `0^2`, which is 0. We call this *element-wise* arithmetic, and we tell Matlab to do this by using a period before the operation. For example, if we enter the command

```
[0,1,2,3,4,5,6].^2
```

the output would be the vector `[0,1,4,9,16,25,36]`. This also applies to multiplication and division. Now we can plot a better plot of $y = x^2$ on the interval $[0, 3]$. Since we have three commands to execute, let's use a script. In the script window, type:

```
x = 0:0.1:3; % Don't forget the semicolon!
y = x.^2; % The period before the ^ indicates ELEMENT-WISE operation
plot(x,y)
```

You should see plot shown in Figure 2.

Just to make sure we understand, let's look at the points that Matlab is plotting. Call the command

```
plot(x,y,'o-')
```

to see all the little dots. There are quite a few of them, and that is the reason that it looks like a smooth curve. If you look closely, however, you will see that this "curve" is actually just points connected by straight lines.

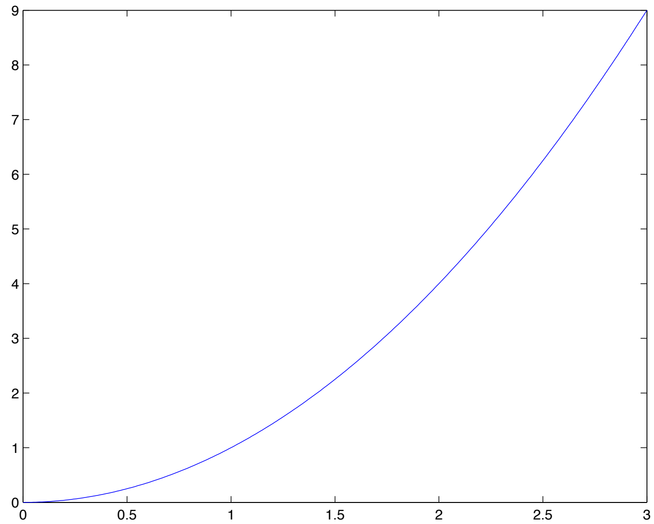

FIGURE 2. A plot of $y = x^2$ with resolution of 0.1.

2.2. **Multiple Plots on a Single Axes.** Suppose we want to plot multiple plots on a single set of axes. We might try and do it as follows:

```
x = 0:0.01:1;
y1 = x.^2;
y2 = x.^3;

plot(x,y1,'blue')
plot(x,y2,'red')
```

But only the red one shows up! What's the deal? Well, when you call `plot`, it will simply make the current figure to be the contents of the latest `plot` call, overwriting what came before. To avoid this, we use the `hold` command, as follows:

```
x = 0:0.01:1;
y1 = x.^2;
y2 = x.^3;

hold on
plot(x,y1,'blue')
plot(x,y2,'red')
hold off
```

This prevents Matlab from overwriting what is already on the axes. Try it yourself and see what happens!

3. MAKING PLOTS PRETTY

Now we're going to explore some options for adding labels and titles to our plots, and modifying the color, line thickness, and line labels. We already saw one such option: when we types `plot(x,y,'o-')` the `'o-'` was an option telling Matlab to mark points with circles and have a line connecting them. If we had used `'o--'` it would connect the circles with a dashed line; `'*-'` marks points with a star instead of a circle. Try it!

Now, by default, Matlab plots look rather boring. Go ahead, try this one.

```
x = linspace(-1,1,50); % 50 evenly spaced points between -1 and 1
y = x.^2;
plot(x,y)
xlabel('This is my x label')
```

```
ylabel('This is my y label')
title('Really boring plot :-\')
```

Yawn The biggest problem is...most everything is too small. And the default blue gets a bit old after a while.

3.1. **Colors and Line Thickness.** Sometimes you'll want a color that's not one of the defaults. It's very easy to use any RGB color code to create any color you want. you'll want to use the `'Color'` option in your plot statement. Right after color, include a vector with the RGB color values of whatever color you want. For example `plot(x,y,'Color',1/255*[148 0 211])` will produce a very nice shade of purple. You can easily find RGB color codes for various colors online. One thing to remember, Matlab accepts RGB values between 0 and 1, the codes you'll find online go from 0 to 255, so you'll want to scale everything you find by 1/255.

While purple is very nice, the line is still a bit thin. the `'LineWidth'` option will allow you to change the thickness of the plot. 1 is default, in this case 2 looks pretty good. Try the following code:

```
plot(x,y,'Color',1/255*[148 0 211],'LineWidth',2)
```

3.2. **Axis Font.** The defualt axis font size is too small. We're going to use `set` to do this. `set` is a built in function that sets properties for graphics objects. If we want to change the font size on the whole plot, the code we want is

```
set(gca,'FontSize',14).
```

You can put this code immediately after the plot statement, or if you used `figure`, right after that. 14 looks good here, but you can play with it to get things looking just right for whatever plot you're making. `gca` means get current axis, which `set` then uses to increase the font size. Notice how all of the font on the plot changed to 14 pt, not just the axes.

3.3. **Axis Labels and Title.** The axis labels should be a bit larger than the axis font, and the title should be a bit larger than the axis labels. For each of these, use the `'FontSize'` option for xlabel/ylabel and title. For example, replace your `xlabel` command with `xlabel('This is my x label','FontSize',16)`. Again, I thought 16 looked good here, but play around with it.

The title works the same way, only it should probably be a little larger. Something like: `title('Much Better!!  :-D','FontSize',18)`

After all that fiddling, our code isn't that much more complicated...

```
x = linspace(-1,1,50);
y = x.^2;
plot(x,y,'Color',1/255*[148 0 211],'LineWidth',2)
set(gca,'FontSize',14)
xlabel('This is my x label','FontSize',16)
ylabel('This is my y label','FontSize',16)
title('Much Better!! :-D','FontSize',18)
```

And the results are much more pleasing!

3.4. **Using LaTeX symbols.** Latex (pronounced "lay tech," not like the material) is a typesetting program commonly used in producing math and science documents. Latex symbols are very easy to use in matlab. Say you want to have something like $\sin(\theta)$ in your plot title. All you'd have to do is type in `title('sin(\theta)')` and you'd get a nice little θ symbol! This works in the `xlabel` and `ylabel` commands as well:

```
x = linspace(0,2*pi,100);
y = sin(x);
plot(x,y,'Color',1/255*[0 205 0],'LineWidth',2)
axis([0 2*pi -1 1])
```

```
title('sin(\theta)','FontSize',18)
xlabel('\theta','FontSize',16)
```

Latex greek letters are all the same...\ then whatever it is. So, π is `\pi`, ξ is `\xi`. If you wanted $\cos(\omega t+\delta)$, just type `cos(\omega t + \delta)`. If you want capital letter, just capitalize it! So, Π is `\Pi`. Easy as π!

3.5. **Legeneds.** If you want to display multiple plots on the same set of axes, sometimes it is nice to use a legend to distinguish your plots. Including a legend is as simple as typing

`legend('plotName1','plotName2',...)`.

0 For example, try the following script:

```
x = linspace(0,1,100);
y1 = x.^2;
y2 = x.^(1/2);
plot(x,y1,x,y2)
legend('This one is x^2','This one is x^3')
```

Note that when we type `x^2` in the legend it automatically puts the 2 as a superscript. To understand more about the power and limits of typesetting math like this, you'll need to delve into the wonderful worls of LaTeX. Feel free to come talk to me if you're interested in learning more about this.

3.6. **Saving Figures. When saving figures, NEVER use `.jpg`.** They look terrible—pixelated with lots of compression artifacts. png files are better: they don't have any of the compression artifacts, but they can still be pixelated. Scale your figures to be large enough so that when you access the saved png, you don't see any pixelation.

You can tell matlab to save figures for you within a script. The command is `saveas(gcf, 'filename.ext', 'format')`. So to save the figure we generated above, you might type: `saveas(gcf, 'sinPlot.png', 'png')`. This can be much more convenient than going in and trying to save figures by hand (which requires a lot of clicks, and can be a pain if you are saving multiple figures).

4. Homework

Write a script that creates plots of the following functions, labeling axes and titling appropriately. Submit a PDF to D2L. Perhaps the easiest way to do this is to copy and paste your figures into a Word document and print to PDF.

- $f_n(\theta) = \sin(n\theta\pi)$ over the interval $\theta \in [0, 1]$ for $n = 1, 2, 4$. Plot all on the same graph with thick lines of different colors. Use a legend.
- $\Omega(x) = (x + 2)(x - 1)(x - 4)$ over the interval $x \in [-4, 5]$. Plot a red line along the x-axis and plot crosses at the roots of Ω.