Plotting in Matlab - A quick tutorial

by Peter Norgaard - norgaard@princeton.edu

I. PLOTTING A LINE

The basic method for plotting a function $f(x)$ in matlab is to create an array of x values, calculate a second array $y = f(x)$, and then plot the set points (x_i, y_i). By default matlab draws a straight line connecting neighboring points, while the points themselves are invisible.

```
x1 = [0:0.25:4*pi];  % creates an array from 0 to 4*pi with a spacing of 0.25
x2 = linspace(0,4*pi,1000);  % creates an array from 0 to 4*pi with 1000 evenly spaced points
y1 = 5*sin(x1).*cos(x1).^2;  % calculate y = f(x)
y2 = 5*sin(x2).*cos(x2).^2;  % note the use of the element-by-element operations .* .^ ./

plot(x1,y1);
hold on;
plot(x2,y2,'color','g','LineStyle','--'),'LineWidth',2);
hold off;
```

In this example we two plots of $f(x)$ in the same figure, one with substantially better resolution than the other. For the second plot we also specify the color and linestyle by including pairs of arguements of the form 'propertyName', propertyValue. The property name is a string, so it has an apostrophe on each end. The property value is sometimes a string (as with the dashed line) and sometimes a number (as with the line width).

The procedure for plotting a parametric curve is nearly identical. Say we have two functions $x(t)$ and $y(t)$ - we simply need to create an array of points for x, an array for y, and then plot them.

```
t = linspace(-10,10,1000);
x = exp(-t.^2);
y = cos(t).^2;
plot(x,y);
```

Of course matlab doesn't know that one curve is a Gaussian and the other is cosine squared - the execution of each line simply stores big arrays (vectors) of numbers in the variables x and y, which we pass into the plot function. If we change the parameter t (say by increasing from 1000 to 10000 points), we must recalculate x and y, and then run the plot command again.

II. AXES LABELS

So we have a nice looking plot, but it's not really a figure until we add axis labels. Matlab has functions designated for this very task,

```
xlabel('This is the x-label','FontSize',14);
ylabel('This is the y-label','FontName','Times');
title('This is the title!','FontWeight,'b');
```

The default font is 10pt Helvetica, but as shown above we can change the size, make it bold, and so on. Often we want to include the value of some parameter in our plot. Let's look at an example and then parse it

```
a = 10;
b = 2;
c = 5;
x = linspace(-10,10,1000);
y = a*exp( -(x-b).^2 / c );
plot(x,y)
title(['A Gaussian with a = ' num2str(a) ', b = ' num2str(b) ', c = ' num2str(c)],...
'FontSize',20,'FontWeight','b');
```

In the argument of the title function, we have used the function num2str() to convert a numerical value (say the value a = 10) to a string. The square brackets *concatenate* the multiple strings into a single long string.

In some instances we might want to use a mathematical formula in a label. This can be done most elegantly if you are familiar with latex syntax. The label functions *interpret* the first argument (a string), applying the specified properties, and then add a label to the plot figure. To use latex commands we must change the *interpreter*, for example the line

```
title('The function $\int f(\theta) d\theta$','FontSize',20,'Interpreter','latex');
```

results in a title "The function $\int f(\theta)d\theta$" (conveniently this document was written in latex!)

When plotting multiple objects on the same graph it is useful (or necessary) to have a legend which identifies each plot. We need to store some information about each plot when it is generated, and then pass this information to the function legend().

```
p1 = plot([0:0.01:1],asin([0:0.01:1]),'--','color','b'); hold on;
p2 = plot([0:0.01:1],acos([0:0.01:1]),'-.','color','g'); hold off;
legend([p1,p2],'arcsine','arccosine',1);
```

The objects stored in p1 and p2 are called *handles*, and will be discussed in further detail in the next section. legend() takes an array of handles as its first argument, followed by a string associated with each handle, and finally an number to specify which corner the legend goes (for more information type "help legend").

III. HANDLES

Every single plot object in matlab (e.g. the figure, the axes, each plot line or surface, ...) has it's own unique identification number known as a handle. For example, to get the handle of the current figure simply type "gcf", which stands for "get current figure". It turns out that the handle for each figure is simply the figure number (1, 2, etc.). It turns out that handles are the secret to accessing every last detail of an object. Try the following commands:

```
x = linspace(0,10,100);
p1 = plot(x,sin(x)./x,'-s','color','r');
f1 = gcf; % stores the handle for this figure in the variable f1
set(f1,'position',[50 50 800 400]);  % sets the "position" property of f1 to [50 50 800 400]
get(f1); % returns all of the properties associated with f1
```

Most everything there is to know about handles has to do with the functions get() and set(). The get() command returns all of the properties associated with your figure in the command window. For example we find that the 'Color' is [0.8 0.8 0.8], which is gray, and that the 'Renderer' is 'painters' (it can be changed to 'OpenGL' for hardware accelerated 3D rendering...). The set() function is used to change the value of a property associated with the specified handle.

Of course the figure handle is just one of several. In the sequence of commands above we also stored the handle of the plot line in the variable p1. Type 'get(p1)' to see all of the properties associated with the plot line. Here's a quick set of sample commands that can be useful - note that more than one property can be changed within a single set() command.

```
set(p1,'Color',[0.5 0 1],'LineWidth',2); % changes the color to purple, and the thickness to 2
set(p1,'LineStyle','--','Marker','none'); % makes the line dashed, without square markers
set(p1,'YData',cos(x)); % changes the y-data and replots
```

Perhaps the most useful handle is associated with the axes. The current axes handle is returned by the command "gca", so the set of axes properties can be obtained by typing "get(gca)". As before, these various properties can then be modified using the set() command. Here are some examples:

```
set(gca,'FontSize',12,'FontName','Times') % changes the size and font of the axis numbers
set(gca,'XLim',[0 5],'YLim',[0 1]); % changes the x and y plot range
set(gca,'XTick',[0:2:10],'XGrid','on'); % sets new ticks on the x-axis and adds grid lines
```

Note that when we changed the x and y limits, this just changed the plot box - the plot still only contains the points specified by the original data arrays (try typing set(gca,'XLim',[0 20]);)

As you work with higher-dimensional plots, or figures with lots of plots on top of each other, the set of handles can seem a little unwieldy. Fortunately, the set of handles for each figure are nicely organized in a *parent-child hierarchy*. The figure handle is at the top. It's *child* is the axes. To see this, type the command get(gcf,'Children') and compare the result to just typing gca. We could also discover that the figure is the parent by typing get(gca,'Parent') and comparing the result to just typing gcf. The axes are in turn the parent of each plot, and also of the xlabel, ylabel, and title. At this point one is tempted to wonder why all this matters... the important result here is that we can find handles that we did not store in variables when the object was created. Let's conclude with a nice, long example:

```
clear all % clears the variable space
close all % closes all figures
x1 = linspace(0,2*pi,1000);
y1 = sin(x1);
y2 = sin(x1)./x1;
x2 = linspace(0,2*pi,9);
y3 = sin(x2);
y4 = sin(x2)./x2;

figure(2) % opens a new figure with ID 2, or goes to figure 2 if it is open
p1 = plot(x1,y1,'color','b');
hold on;
p2 = plot(x1,y2,'color','r');
p3 = plot(x2,y3,'-o','color','c');
p4 = plot(x2,y4,'-s','color','m');
hold off;

set(gca,'XLim',[0 2*pi],'XTick',[0:pi/2:2*pi],'XGrid','on');
set(gca,'XTickLabel',{'0','p/2','p','3p/2','2p'},'FontName','Symbol','FontSize',12)

xlabel('$\theta$ [rad]','FontSize',14,'Interpreter','latex')
ylabel('$f(\theta)$','FontSize',14,'Interpreter','latex')
title('Plots of $f_1(\theta) = sin(\theta)$ and $f_2(\theta) = \frac{sin(\theta)}{\theta}$',...
'FontSize',16,'FontWeight','b','Interpreter','latex');
l1 = legend([p1,p2,p3,p4],'f_1','f_2','f_1 undersampled','f_2 undersampled',1);
set(l1,'FontName','Helvetica')
```