High-Quality Figures in MATLAB[1]

Contents

1 Exporting the Figure

Before delving into methods of making figures colorful or full of well-placed text, we'll talk about how to even get those beautiful figures out of MATLAB and into your write-ups.

Say you have the xy data in Figure 1a, generated and plotted with this code:[2]

```
X = linspace(0,20,100)';
Y = X/20 + 0.2*rand(100,1) + 0.2*rand(100,1).*exp(X/10);
plot(X,Y,'*');
```

I obtained the image in Figure 1a by clicking into the figure and then saving it as a PNG file. This method is intuitive but unreliable, and produces poor-quality images. Saving the figure as a PDF file results in Figure 1b, which is a vectorized image and looks alright. But

[1] Mike Hansen; University of Utah
January, 2013

[2] For those of you who notice that this data is random: I generated it once, saved it to a .mat file, and then loaded that for each new plot

the PDF prints "Student Version of MATLAB" a few inches below the plot, and there is some extra whitespace, which we never told MATLAB to do. Exporting directly to an JPG file (Figure 1c) is worse than the PNG.

Finally, when saving a figure, replication of the same plot is difficult because the size and proportions of the Figure window in your MATLAB desktop will determine the size and proportions of the saved image. If you're going to place multiple images right next to one another this can make things trickier than they should be.

I'm sure there are many other ways to fix these problems and obtain high-quality images, but the following way in particular is easy to implement, works very well, and can be used to automate some annoying tasks. Although this method suggests the use of PDF files, EPS images have benefits as well, such as post-MATLAB modification with Adobe Illustrator. Using EPS format instead of PDF may be done with many of the following commands by replacing `-dpdf` with lines such as `-depsc2 -tiff`. There is seemingly unending information about file formats at the MathWorks website.

1.1 The `PaperSize`

We want to use PDF's so we can get vectorized graphics, but comparing Figure 1b with Figures 1a and 1c indicates that some extra white space is produced with the PDF. We can control this, as well as the Figure size, by changing the `PaperSize`. This can be done from within an m-file or in the command window. The commands below set the paper upon which the PDF is printed to an 8-inch by 8-inch square.

```
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [8 8]);
```

The variable `gcf` is the handle of the current figure. This command is discussed later in section 3. Once we've set the `PaperSize`, we can also set the position and size of the figure with the `PaperPosition` property. The line below is read from left to right as:

1. Place the figure 0.5 inches from the left edge
2. Place the figure 0.5 inches from the bottom edge
3. The figure should be 7 inches wide
4. The figure should be 7 inches tall

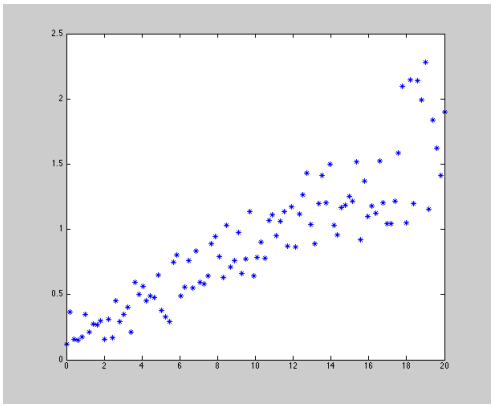
```
set(gcf,'PaperPosition',[0.5 0.5 7 7]);
```

I like these settings because it gives me an 8-inch by 8-inch square image, with half-inch margins on all sides, which makes scaling multiple figures on an 8.5"x11" page very straightforward. In general, the command can be written as

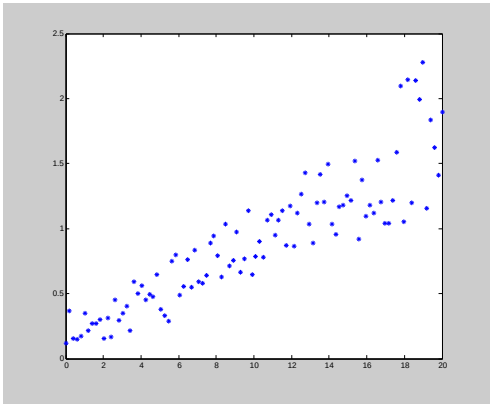
```
set(gcf,'PaperPosition',[left bottom width height]);
```

Finally, in order to make MATLAB accept our manual setting of `PaperSize` and `PaperPosition`, we have to use the following command:

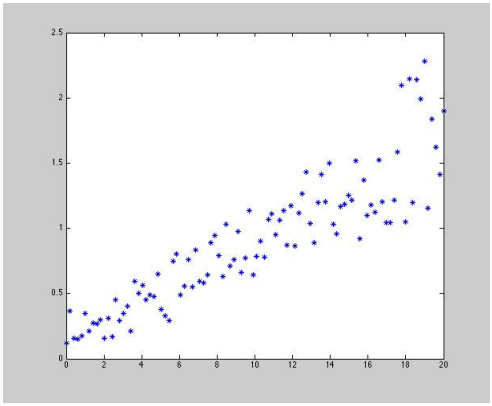
```
set(gcf,'PaperPositionMode','Manual');
```



(a) Example Data PNG



(b) Example Data PDF



(c) Example Data JPG

Figure 1: Example Data; Different Output Formats

1.2 The `print` Command

Now that we've set the paper for printing and placed the image right where we want it, we use the `print` command to put the figure on the paper. We want vectorized PDFs with nice resolution, so we use the following line:

```
print(gcf, '-dpdf', '-r150', 'filename.pdf');
```

This can be read as "print the current figure to a pdf, with resolution of 150 dots per inch, and save the resultant pdf as filename.pdf." This line can also be written as: `print -painters -dpdf -r150 filename.pdf`.

The difference between the PDF output in Figure 1b and Figure 2 is clearly evident: the undesired and uncontrolled white space isn't in Figure 2, and the figure looks surprisingly nicer as a result. And because we controlled image sizing & placement, we could easily repeat the process.

Because these `print` commands can be placed in an m-file, you could create twenty plots in a single run, and save all twenty without having to open them all and click File->Save. If you have a lot of similar plots to create, you could put the print command into a loop and change the filename at each iteration of the loop. The commands below created the five plots (in separate files) in Figure 3 with the push of a single button.

```
x = rand(5,100);
y = rand(5,100);
for i = 1:5
    plot(x(i,:), y(i,:));
    filename_string = ['Plot_Number_', num2str(i) ,'.pdf'];
    print(gcf, '-dpdf', '-r600', filename_string);
end
```

Now that we know how to produce high-quality vectorized PDFs (and even how to automate the process), let's move on to making the plots themselves look better.

2 Modifying the Lines/Markers

2.1 Inline Line/Marker Settings

As done in the example plots above, we can quickly customize the plot by using statements like `plot(X,Y,'*')` or `plot(X,Y,'r--')`. Figure 4 shows four common settings. Type `help plot` in your command window or go to the MathWorks website to see a comprehensive list of options.

To reiterate the power of the print command, here is the code that generated all four of the plots in Figure 4:

```
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [8 8]);
set(gcf,'PaperPosition',[0.5 0.5 7 7]);
set(gcf,'PaperPositionMode','Manual');
```

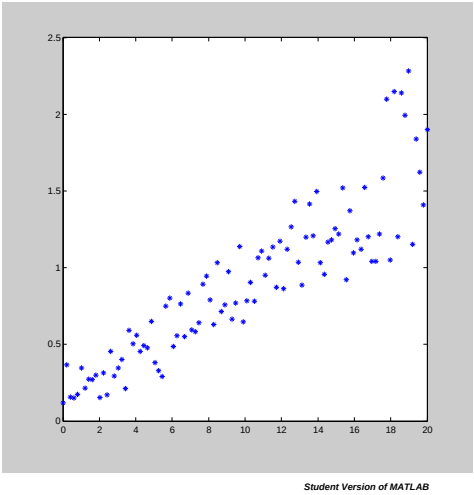


Figure 2: Example Data; PDF created with the `print` command and a manual paper size

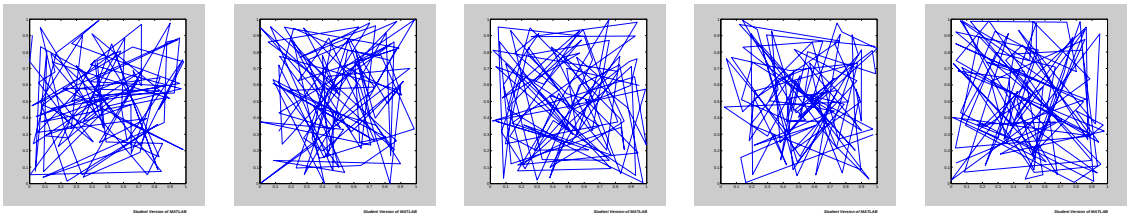
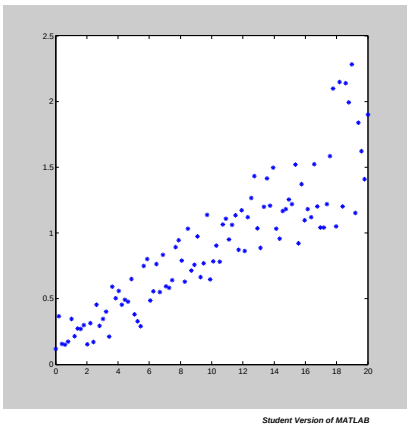
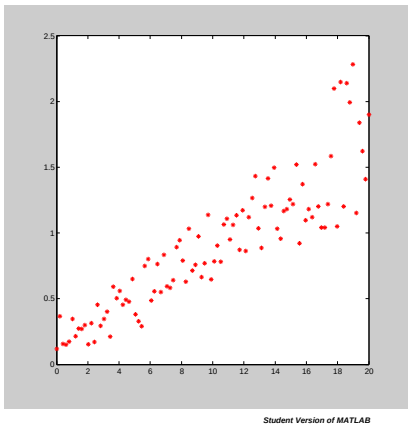


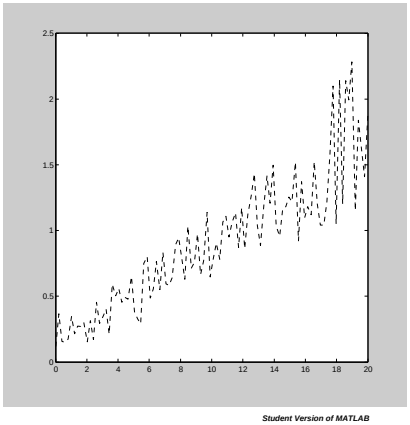
Figure 3: Example of Automated PDF Printing



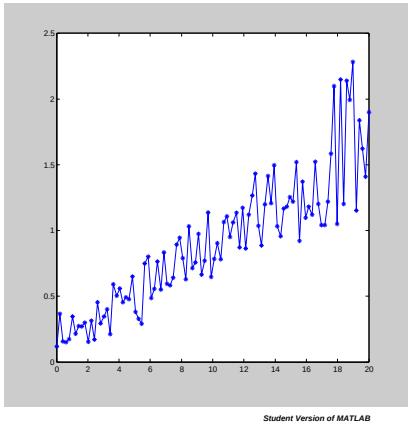
(a) `plot(X,Y)`



(b) `plot(X,Y,'r*')`



(c) `plot(X,Y,'k--')`



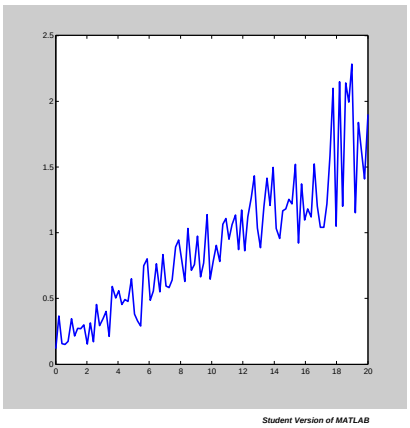
(d) `plot(X,Y,'b*-')`

Figure 4: Common inline plot customizations

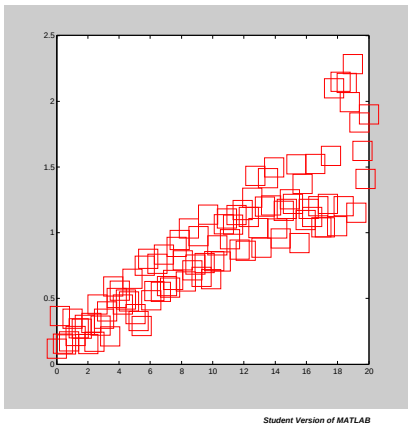
```
plot(X,Y);
print -painters -dpdf -r150 Data_Example_DefCust.pdf
plot(X,Y,'r*');
print -painters -dpdf -r150 Data_Example_RedAst.pdf
plot(X,Y,'k--');
print -painters -dpdf -r150 Data_Example_BlackDash.pdf
plot(X,Y,'b*-');
print -painters -dpdf -r150 Data_Example_BlueDotDash.pdf
```

There are more options for inline plot customizations. Any property of the line/markers can be modified, as shown in the code below (see Figure 5 for the results). The syntax is pretty straightforward, and if you want to customize a specific property that isn't done somewhere in this document, check the MathWorks website.

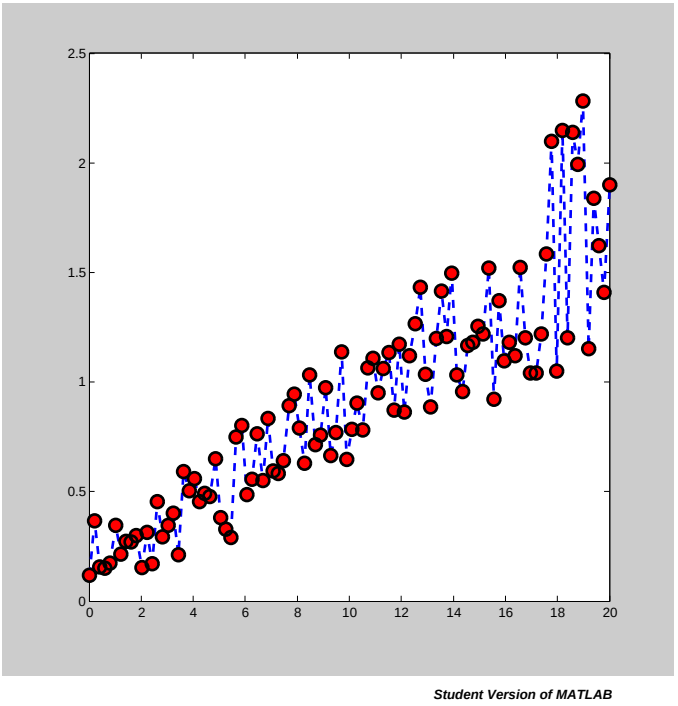
```
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [8 8]);
set(gcf,'PaperPosition',[0.5 0.5 7 7]);
set(gcf,'PaperPositionMode','Manual');
plot(X,Y,'LineWidth',2);
print -painters -dpdf -r150 Demo_LineWidth.pdf
plot(X,Y,'rs','MarkerSize',30);
print -painters -dpdf -r150 Demo_MarkerSize.pdf
plot(X,Y,'bo--',...
          'LineWidth',2,...
          'MarkerEdgeColor','k',...
          'MarkerFaceColor','r',...
          'MarkerSize',10);
print -painters -dpdf -r150 Demo_CrazyMarker.pdf
```



(a) `plot(X,Y,'LineWidth',2)`



(b) `plot(X,Y,'rs','MarkerSize',30)`



(c) `plot(X,Y,'bo--', 'LineWidth',2, 'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',10)`

Figure 5: Common inline plot customizations

2.2 Using `set` After `plot`

Any of the above changes may be made after having written the `plot(x,y,...)` line. To `set` properties of a plot, however, you need a handle to it. To create a handle to a plot, simply assign the handle variable to the plot when you create it:

```
handle = plot(X,Y);
```

Then you can set properties with code like this:

```
set(handle,'LineWidth',2);
set(handle,'color','r');
set(handle,'Marker','o');
set(handle,'MarkerSize',10);
```

2.3 Creating New Colors

The standard blue, red, green, magenta, etc. in MATLAB's color properties aren't always the best choices. A dark green would be great, but unfortunately there is no immediate option. However, we can create colors with RBG (red-green-blue) vectors. An RGB vector is simply a 1x3 vector with values between 0 and 1, which indicate the "amount" of red, green, or blue in the color. To find the RGB values for your favorite color, Google it or experiment for a while. Most often the RBG values will be given from 0 to 255; just divide the given values by 255 to get the MATLAB equivalent. An example of creating and setting a line to forest green is below:

```
forest_green_RGB = [34 139 34]/255;
set(handle,'color',forest_green_RGB);
```

3 Modifying the Axes and Figure

3.1 Axis Labels & Title; Font Modification

To add labels to the axes, simply use `xlabel('string for x axis')` and `ylabel('string for y axis')`. To add a title use `title('string for title')`. Because the text you get from these commands is usually way too small, use the following commands to modify the font size and weight. These commands use the variable `gca`, which returns a handle to the current axes.

```
set(get(gca,'xlabel'),'FontSize', 18, 'FontWeight', 'Bold');
set(get(gca,'ylabel'),'FontSize', 18, 'FontWeight', 'Bold');
set(get(gca,'title'),'FontSize', 18, 'FontWeight', 'Bold');
```

To modify the font name, the keyword is `'FontName'`, and you can type `listfonts` at your command window to see the available fonts.

The commands for changing the font of the numbers on the axes are similar. The linewidth of the axes should be increased as well.

```
set(gca,'FontSize',16);
set(gca,'FontWeight','Bold');
set(gca,'LineWidth',2);
```

3.2 Axis Scaling

To remove the box around the axes, use the `box off` command. For some data the box is helpful but other times is terrible.

The axes can be changed or removed, and four useful ways are shown in Figure 6.

To enforce specific bounds on the axis, use the commands

```
axis([xmin xmax ymin ymax]);
```

And to change the tick spacing, use the commands

```
set(gca,'XTick',xmin:xspacing:xmax);
set(gca,'YTick',ymin:yspacing:ymax);
```

Another useful command is `gcf`, which gets a handle to the current figure. I've only ever used this command to remove the gray background, which is done with

```
set(gcf,'color','w');
```

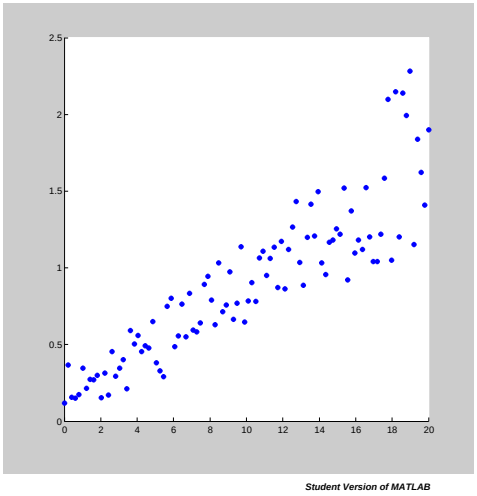
Throwing together a bunch of these commands (see below) results in Figure 7.

```
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [8 8]);
set(gcf,'PaperPosition',[0.5 0.5 7 7]);
set(gcf,'PaperPositionMode','Manual');
plot(X,Y,'*','LineWidth',2);
xlabel('XData','FontWeight','Bold','FontSize',18);
ylabel('YData','FontWeight','Bold','FontSize',18);
title('Some Example Data','FontWeight','Bold','FontSize',18);
axis([0 20 0 2.5]);
set(gca,'YTick',0:0.25:2.5);
set(gca,'XTick',0:4:20);
box off; axis square;
set(gca,'LineWidth',2);
set(gca,'FontSize',16);
set(gca,'FontWeight','Bold');
set(gcf,'color','w');
print -painters -dpdf -r150 Demo_AxesFontEtc.pdf
```

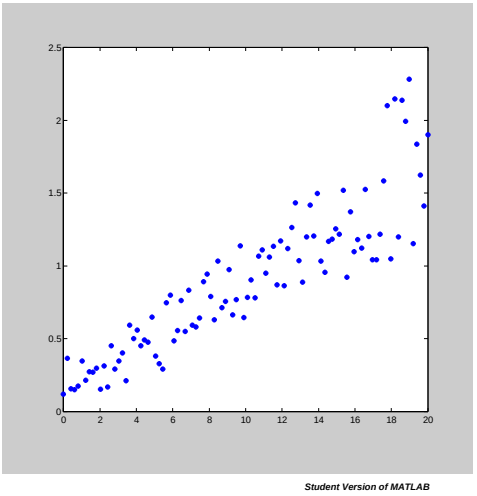
3.3 Adding a Legend

Adding a legend is straightforward, but we'll want more than one data set. We've fit a function $ax + b\cos(cx)$ to our example data, and the results with a legend are below. The code to create the legend is below. The first two arguments to the `legend` command are the names we want to appear in our legend. The final two arguments tell MATLAB where to put the legend. The keywords for `Location` are things like `'NorthEast'` or `'WestOutside.'` Choosing `...'Location','Best')` lets MATLAB pick.

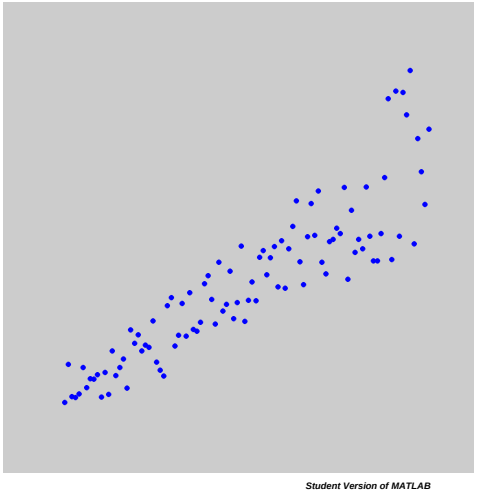
```
legend('Data','Fit','Location','NorthWest');
```



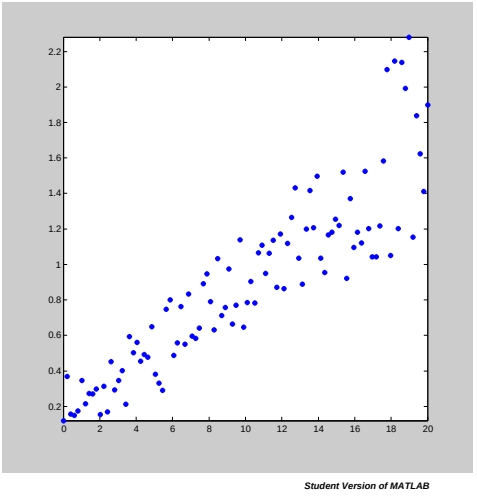
(a) `axis normal, box off`



(b) `axis square`



(c) `axis off`



(d) `axis tight`

Figure 6: Common Axes Modifications

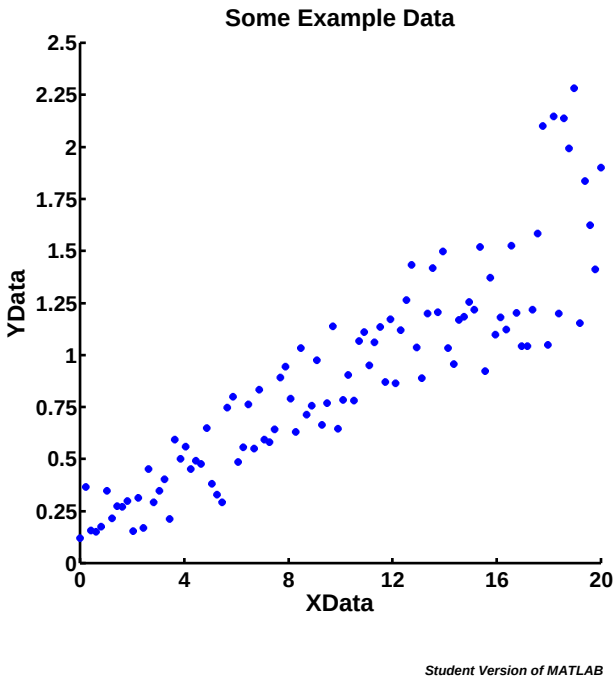


Figure 7: Example Data after Selected Axes, Font, Line Modifications

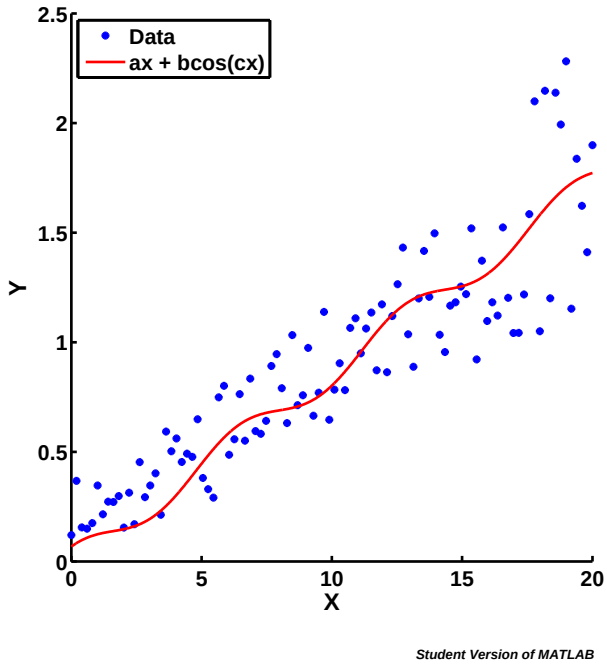


Figure 8: Example Data with a Fitted Curve and Legend

Nonlinear Fitting Moment Before moving on, I'll describe how we can quickly get that curve to fit to our data. MATLAB has an awesome command called `nlinfit`, that will open up a lot of really neat things for you.

To fit a function f to the data $\{x, y\}$, you first have to create a function which takes in a vector of fitting parameters and the independent variable, and returns the function's output. For the function above, we have

$$y(x) = ax + b\cos(cx)$$

```
fun = @(p,x) p(1)*X + p(2)*cos( p(3)*X );
```

So p(1) is a, p(2) is b, and p(3) is c. Next we simply plug this function and our data into `nlinfit` with the line below. The vector [1 0.2 1] is just our initial guess at the fitting parameters.

```
par = nlinfit(X,Y,fun,[1 0.2 1]);
```

`par` is the vector of fitting parameters that give the best fit to the data. If you're interested in finding 95% confidence intervals on fitting parameters or model predictions, type `help nlinfit`, `help nlparci`, and `help nlpredci`.

4 Automated Modifications

The beautiful figures we can obtain obviously come with the cost of a lot of lines of code. There's a very simple way to replace all of this code with one line: put it in a function. One implementation, which replaces some axes and figure modifications with "`goodplot;`" is below. To use this function, you would simply plot your data and then type `goodplot;` on the next line. It is a good idea to save goodplot.m in your root MATLAB directory, so it is always in the path (so every m-file you write can use it).

```
function goodplot()
% function which produces a nice-looking plot
% and sets up the page for nice printing
set(get(gca,'xlabel'),'FontSize', 18, 'FontWeight', 'Bold');
set(get(gca,'ylabel'),'FontSize', 18, 'FontWeight', 'Bold');
set(get(gca,'title'),'FontSize', 18, 'FontWeight', 'Bold');
box off; axis square;
set(gca,'LineWidth',2);
set(gca,'FontSize',16);
set(gca,'FontWeight','Bold');
set(gcf,'color','w');
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [8 8]);
set(gcf,'PaperPosition',[0.5 0.5 7 7]);
set(gcf,'PaperPositionMode','Manual');
```

This is a fairly simple implementation. Adding arguments to the function is a great idea; you can obtain different results without changing the actual function each time. Using `nargin` to set default arguments is particulary handy here (example below). If you wanted to you could write a function which gives the option of printing the plot to a pdf file, and which accepts the name of the pdf as an argument (you could even include resolution as an argument).

```
function goodplot(papersize, margin, fontsize)
% function which produces a nice-looking plot
% and sets up the page for nice printing
if nargin == 0
    papersize = 8;
    margin = 0.5;
    fontsize = 18;
elseif nargin == 1
    margin = 0.5;
    fontsize = 18;
elseif nargin == 2
    fontsize = 18;
end
set(get(gca,'xlabel'),'FontSize', fontsize, 'FontWeight', 'Bold');
set(get(gca,'ylabel'),'FontSize', fontsize, 'FontWeight', 'Bold');
set(get(gca,'title'),'FontSize', fontsize, 'FontWeight', 'Bold');
box off; axis square;
set(gca,'LineWidth',2);
set(gca,'FontSize',16);
set(gca,'FontWeight','Bold');
set(gcf,'color','w');
set(gcf,'PaperUnits','inches');
set(gcf,'PaperSize', [papersize papersize]);
set(gcf,'PaperPosition',[margin margin papersize-2*margin papersize-2*margin]);

set(gcf,'PaperPositionMode','Manual');
```

5 Automated Text

Sometimes you'll want some text on a figure, and you can add it with the Insert->TextBox button. But on occasion you'll want enough text to warrant some form of automation. In order to display a string (such as 'some text") at point (x, y), use the command below. This will place the left side of the string at the point (x,y).

```
text(x,y,'some text');
```

To automate the addition of multiple text boxes, you'll want a loop of some sort. See the example below. Note that the code also uses the `num2str` function. When you want to use a number in a string, you need to use this function. If you type `num2str(number)` you'll get the number as it is. If you type `num2str(number, precision)`, you'll get `precision` number of digits beyond the decimal.

Also note that we can treat a text object like a plot, by setting a handle equal to it and using `set` to change properties.

It's pretty clear from Figure 9 that automated text placement can create awkward overlap if not done properly. It takes some tuning but when you get it just right you'll be glad you aren't typing all of those text labels every time you change the plot.

```
z = linspace(0,1,100)';
r = z.^2;
p = plot(z,r,'LineWidth',2);
for i = 20:10:100
    z_str = num2str(z(i),2);
    r_str = num2str(r(i),2);
    str = ['(z,r) = (',z_str,', ',r_str,')'];
    h = text(z(i)-0.25, r(i), str);
    set(h,'FontSize',18);
    set(h,'FontWeight','Bold');
end
```

6 Scatter Plots

6.1 Basics

While the `plot` command is useful, the `scatter` command allows more customization of individual data markers. To make a simple scatter plot, create vectors `x` and `y`, and then just type `scatter(x,y);`. For the following examples, I'll use this data:

```
N = 50;
x = linspace(0,100,N)';
y = exp(-x/10).*x.^2;
```

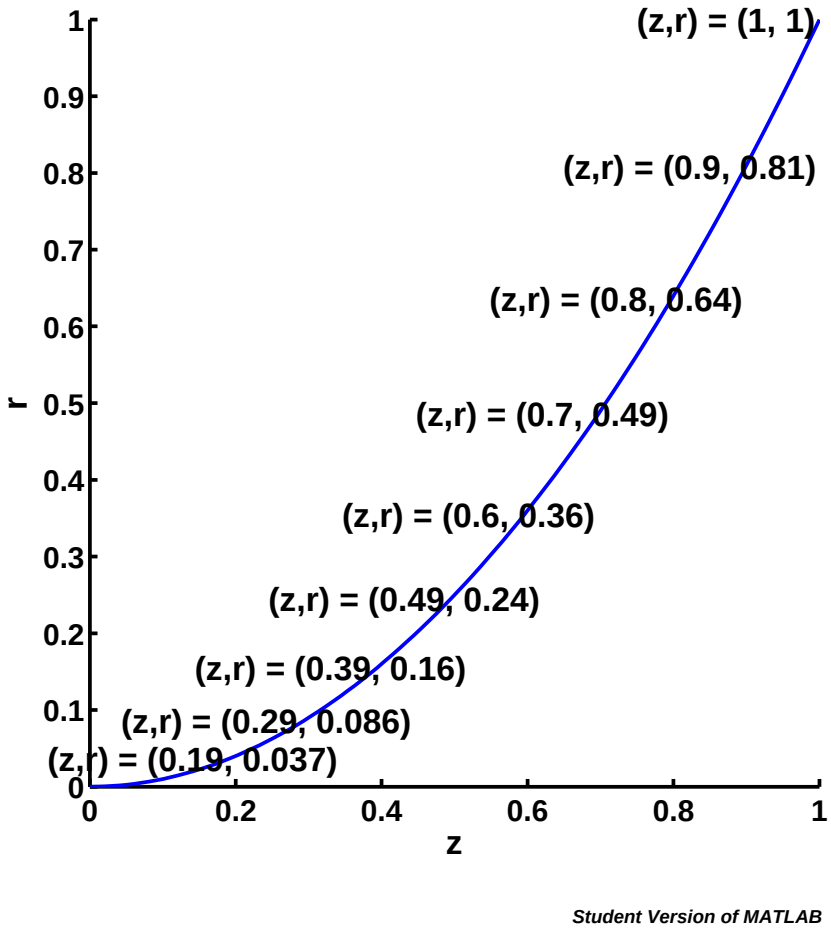


Figure 9: Example of Automated Text

Using `scatter(x,y);` with some of the axes modifications described in previous sections gives the plot in Figure 10a. We can modify the size of the markers by adding an argument to the command, i.e. `scatter(x,y,200);`. This scales the markers to be 200 points-squared, see Figure 10b for the result. We modify the color of the markers after the size. Red markers are obtained with `scatter(x,y,200,'r');`, which is shown in Figure 10c. Filling in the markers is done with `scatter(x,y,200,'r','filled');`, and the result is Figure 10d.

6.2 More Advanced Marker Sizes and Colors

In this section we modify the sizes and colors with vectors, allowing for scatter plots with markers of different colors and different sizes. Two examples:

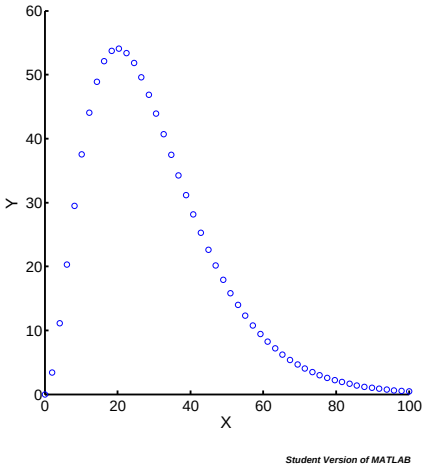
1. You want a plot where the height of the marker determines its size.
2. You want to see markers go from red to blue as you move away from the y-axis.

The first example is done with the following code. The result is shown in Figure 11a. Figure shows what happens if we size by x instead of y.

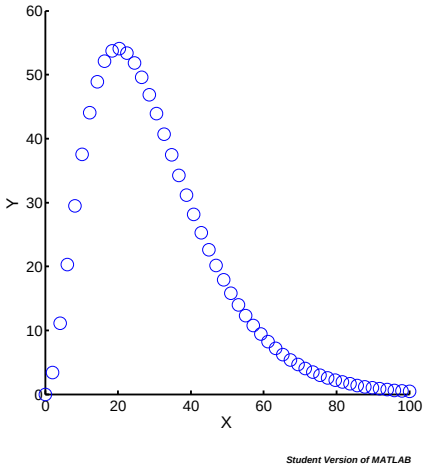
```
N = 50;
x = linspace(0,100,N)';
y = exp(-x/10).*x.^2;
sizes = 100*y + 10;
scatter(x,y,sizes);
```

Example two is solved below. The code is very similar to that for vector sizes. We are able to get a nice gradient from red to blue by using `linspace` to determine the rgb content of each marker. Although we don't have any green in this picture, it's included in the code anyways (note that the linspace for green goes from zero to zero - if you change it from zero to one and run the code you'll go from red to cyan).

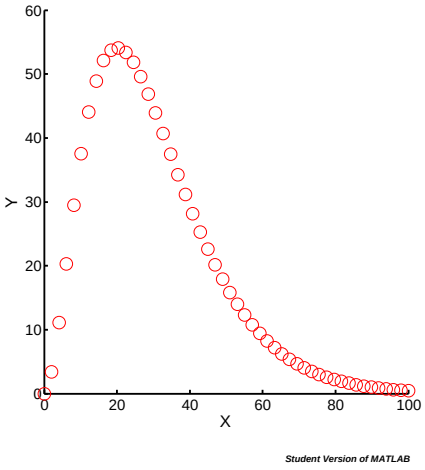
```
N = 50;
x = linspace(0,100,N)';
y = exp(-x/10).*x.^2;
colors_redblue_spectrum = zeros(N,3);
red = linspace(1,0,N)';
green = linspace(0,0,N)'; % just here for later modification
blue = linspace(0,1,N)';
for i = 1:N
    colors_redblue_spec(i,:)  = [red(i) green(i) blue(i)];
end
scatter(x,y,150,colors_redblue_spec,'filled');
```



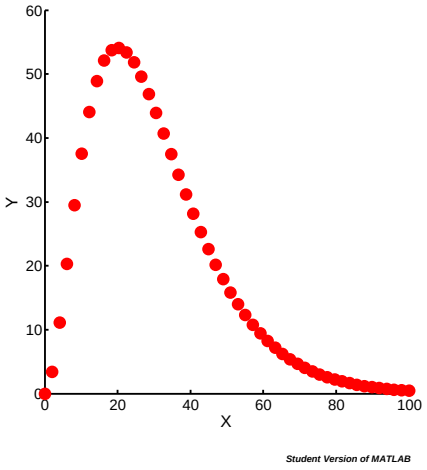
(a) `scatter(x,y)`



(b) `scatter(x,y,200);`



(c) `scatter(x,y,200,'r');`



(d) `scatter(x,y,200,'r','filled');`

Figure 10: Basic scatter plots

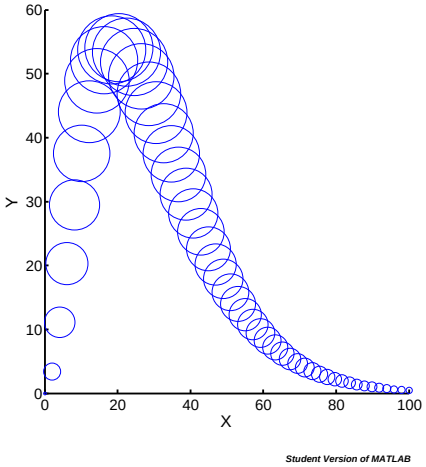
Although this produces a nice scatter plot, we have to change the code slightly to use our `print -painters` lines required to get high-quality PDFs. The code below defines the `colors_redblue_spec` matrix as our `colormap`, and then uses a vector of integers from 1 to N to color the markers. This allows the `-painters` renderer to produce the vectorized PDF. See the MathWorks website for more info.

```
% replace the end of the previous code with this!
colormap(colors_redblue_spec);
scatter(x,y,150,1:N,'filled');
```

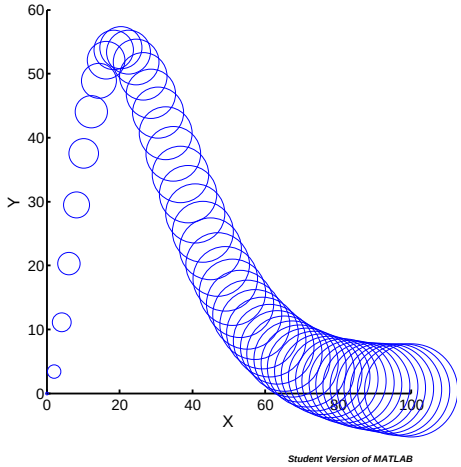
Figure 12a shows the result of this code. Figure 12b shows the results of using `green = linspace(0,1,N)';`.

Figure 13 shows how crazy/powerful this can get. This is generated by the code below. It colors the marker by its placement in the xy-plane, such that the bottom-left is pure red, top-right is pure green, bottom-right is pure black, top-left is pure red-green mix, and the middle is a gradient of everything.

```
N = 500;
x = rand(N,1);
y = rand(N,1);
red = (max(x)-x)/max(x);
green = y/max(y);
blue = zeros(N,1);
colors_redblue_spec = zeros(N,3);
for i = 1:N
    colors_redblue_spec(i,:)  = [red(i) green(i) blue(i)];
end
colormap(colors_redblue_spec);
h = scatter(x,y,500*rand(N,1),1:N,'filled');
```

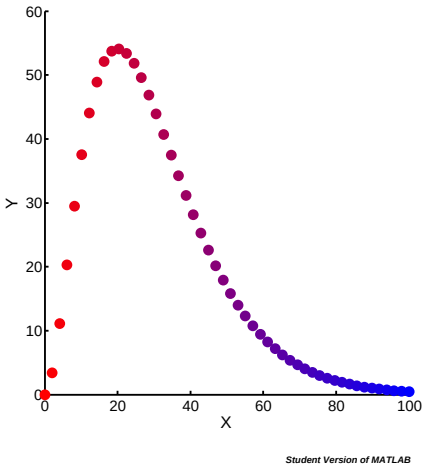


(a) Height-based marker size

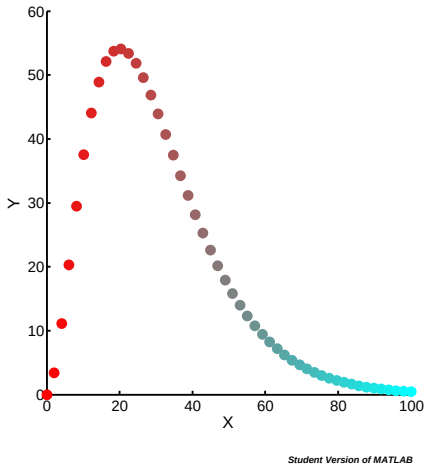


(b) Length-based marker size

Figure 11: Marker size modifications



(a) Marker color as a red-blue gradient



(b) Marker color as a red-cyan gradient

Figure 12: Marker size modifications

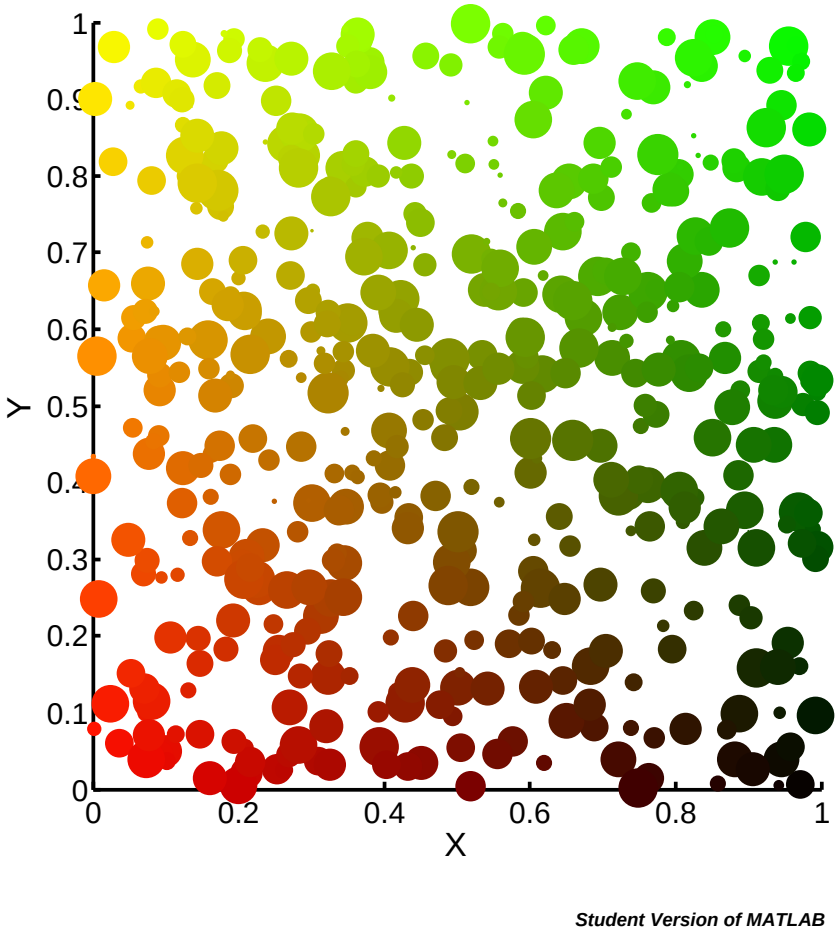


Figure 13: Marker color that depends on both x and y