Circuits 2 - Project 2
Zachary Sasser

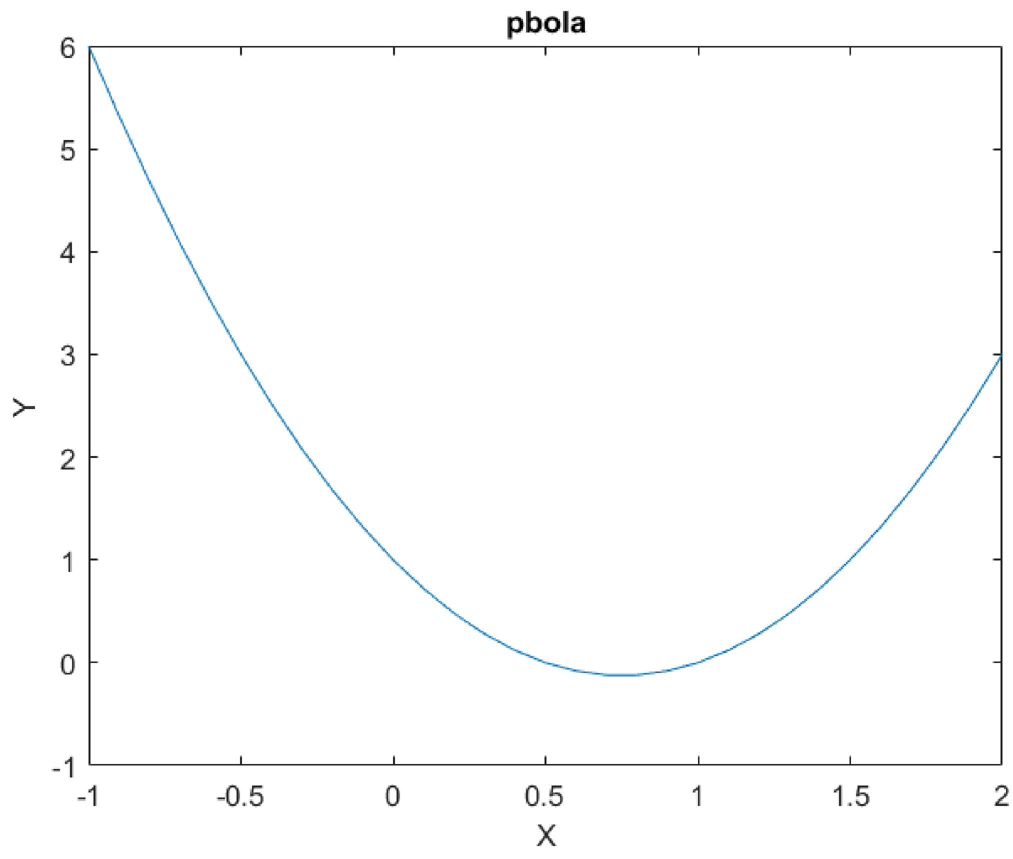
Introduction:

The purpose of this project is to exemplify basic usage of matlab functional programming. It is also helpful to take note that you can easily create functions that can extend into extra dimensionality without having to add any extra code to handle it as you would in a traditional language such as C++.

Part 1: Basics

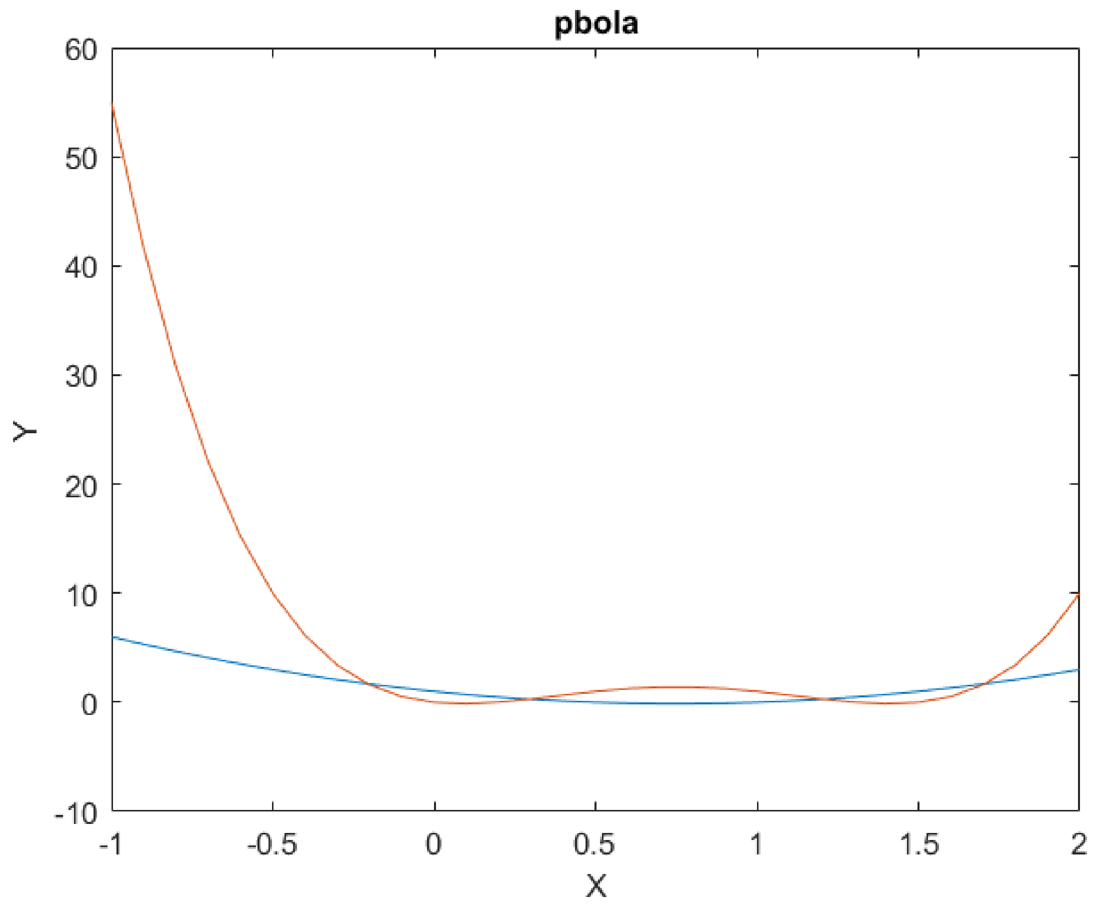
1. *Create the function file above. Then pass the number -1 (pbola(-1)) into the function. What is the output? Next create an x-vector from -1 to 2 and pass into the function. Plot the response (10 points)*

```
>> pbola(-1)

ans =


     6
>> x = -1:0.1:2;
>> y = pbola(x);
>> plot(x,y); title("pbola"); xlabel("X"); ylabel("Y");
```



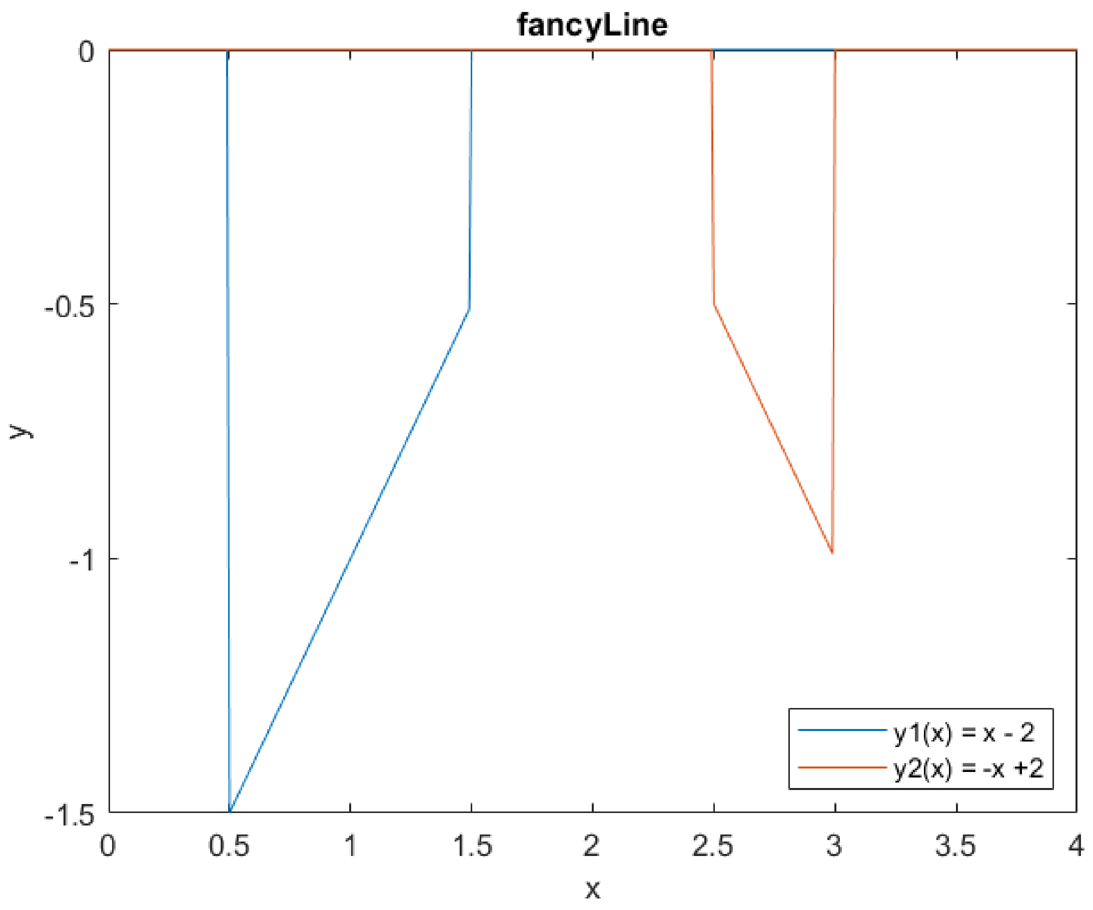
2. *Pass the output of pbola(x) into itself. Plot pbola(x) and pbola(pbola(x)) on the same graph using hold (10 points)*

```
>> hold on
>> y = pbola(pbola(x)); plot(x,y);
```



3. *The next task is to create a function for a line, y = hx + q. The input parameters should be the slope (h), the y-intercept point (q), and a vector x lims that contains the "turn on" and "turn off" values of x for the line. For example, if an x-vector is given that ranges from 0 to 5 and we want to plot a line that starts at x = 1 and turns off at x = 4, with a slope of 3 and a y-intercept point of 1.* *Show the plot in the report (10 points).*

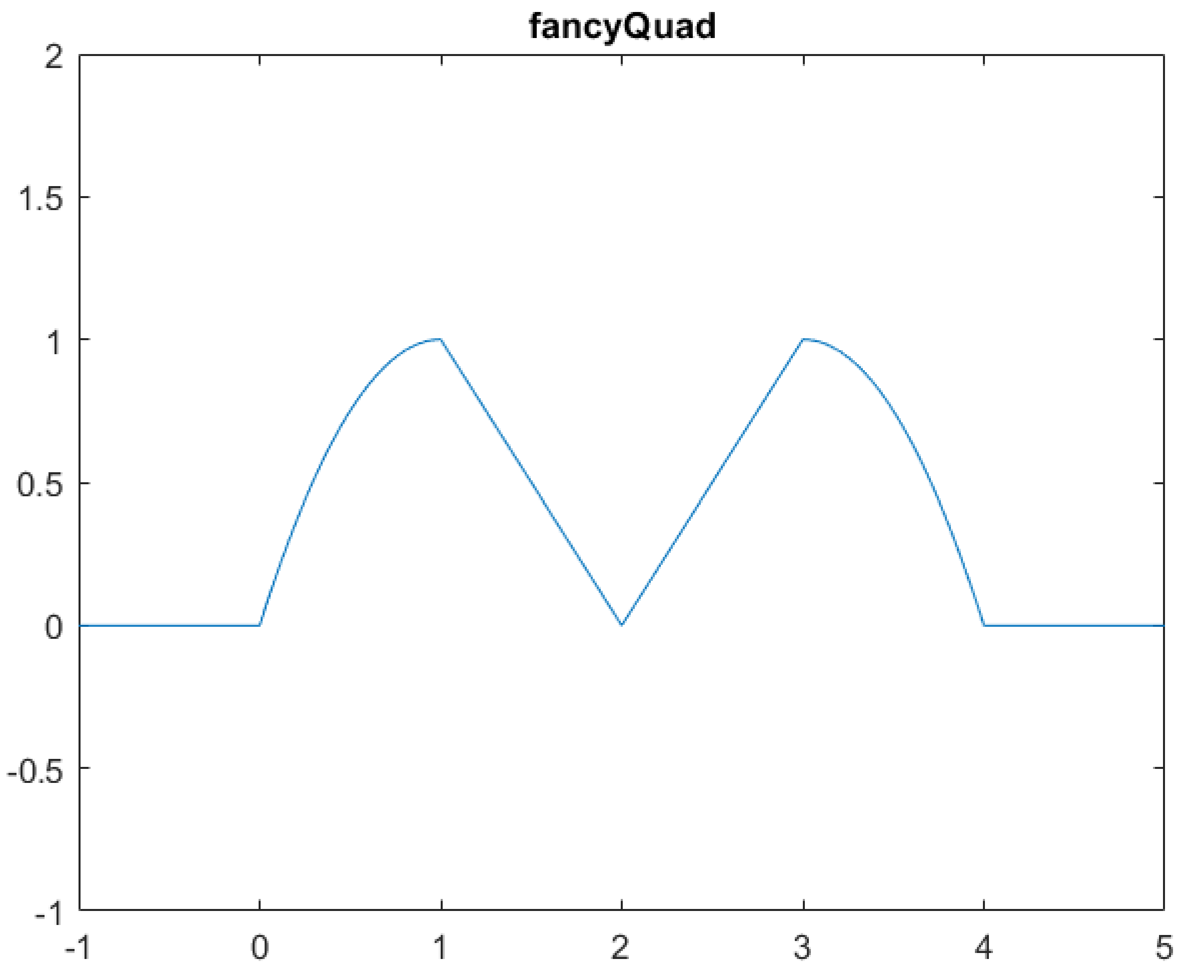
```
>> fancyLine([1;-1],[-2;2],0:0.01:4,[0.5,1.5; 2.5,3]);
>> xlabel("x"); ylabel("y"); title("fancyLine");
legend({"y1(x) = x - 2","y2(x) = -x +2"}, 'Location', 'southeast');
```



Note:
The document is not specific on what "format" means so I took that to mean they just needed to be differentiated somehow. It also doesn't specify what the off and on points for y1 and y2 should be, so I used that to differentiate my "format". I chose [0.5,1.5] for y1 and [2.5,3] for y2.

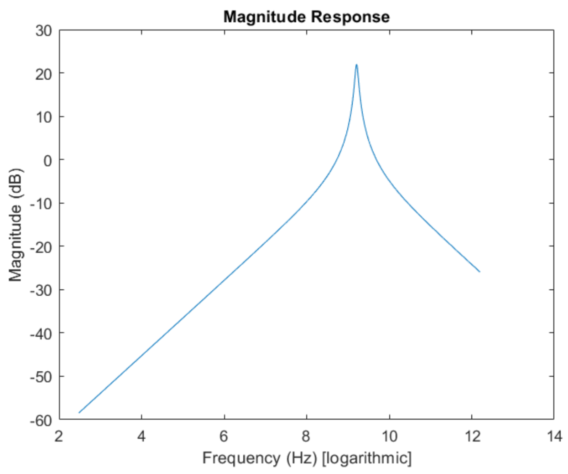
4. *Now create a function that can generate a parabola of the format $y(x) = a_2x_2 + a_1x + a_0$, where a_0, a_1, and a_2 are input parameters to the function as well as the vector x and the start and stop x values (x_1 and x_2). where x_1, x_2, and x_3 are the intercept points between the adjacent functions. (see document for y3 definition)*

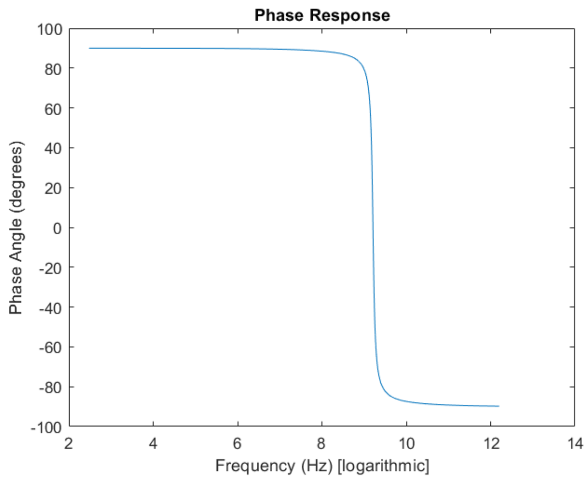
```
>> x = -1:0.01:5;
>> x_lims = [0,1; 1,2; 2,3; 3,4];
>> a2 = [-1;0;0;-1]; a1 = [2;-1;1;6]; a0 = [0;2;-2;-8];
>> y3 = sum(fancyQuad(a2,a1,a0,x,x_lims));
>> plot(x,y3); ylim([-1, 2*max(y3)]); title("fancyQuad");
```



Part 2: Filter Functions

```
>> bandpass(10000,5000,1);
```





Conclusion

Matlab is highly functional for mathematical operations at the expense of performance. It allows you to do what would normally be complex programming operations with simple mathematics notation. Doing operations with matrices and imaginary numbers is exceedingly simple compared to traditional methods, as shown in part 2. Additionally the seamless usage of matrices makes parallel computation a breeze as simply adding columns allows for parallel processing that would normally require loops of scalar operations in a language such as C++. The syntax of matlab puts those complex loop operations under the hood.

Appendix:

pbola.m

```
function y = pbola(x)
y = 2*x.^2-3*x+1;
```

fancyLine.m

```
function y = fancyLine(h,q,x,x_lims)
y = (h*x + q).*(ceil(heaviside(x-x_lims(1))) - ceil(heaviside(x-x_lims(2))));
plot(x,y);
```

fancyQuad.m

```
function fancyQuad(a2,a1,a0,x,x_lims)
y = (a2*(x.^2) + a1*x + a0).*(ceil(heaviside(x-x_lims(:,1))) - ceil(heaviside(x-x_lims(:,2))));
```

bandpass.m

```
function out = bandpass(f0,BW,pt)
%BW = 1/RC
%w0^2 = 1/LC
%This is inefficient memory use.
%I'm doing this for readability
w0 = 2*pi*f0;
fc1 = f0 - BW/2;
fc2 = f0 + BW/2;
lowBound = fc1/100;
highBound = fc2*100;
w = lowBound:1:highBound;

%Calculate complex H results
H = (w0.*(w*1i))./((w*1i).^2+BW.*(w*1i)+w0.^2);
phaseH = angle(H)*180/pi;
magH = mag2db(abs(H));

%Plot
if pt
   plot(log(w/2/pi),magH);
   xlabel("Frequency (Hz) [logarithmic]");
   ylabel("Magnitude (dB)");
   title("Magnitude Response");

   figure(2);
   plot(log(w/2/pi),phaseH);
   xlabel("Frequency (Hz) [logarithmic]");
   ylabel("Phase Angle (degrees)");
   title("Phase Response");
end

%return
out = [magH; phaseH];
```

Note: Calculating R and L was unnecessary, but would have just been solving the equations shown in the first two comments.