Lab #2: Basic Plotting of Signals

Author: Mayank Vanjani

ID: N18293791 (mv1506)

ECE-UY 3054

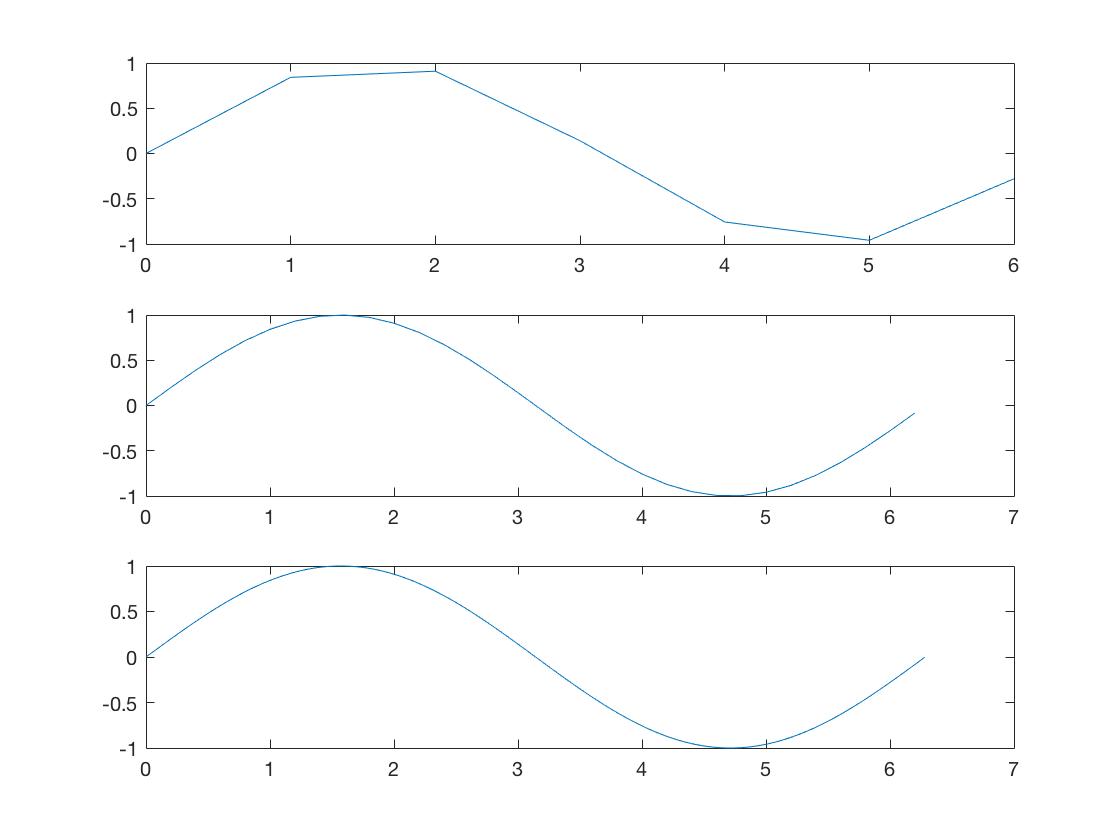
Date of Lab: 9/21/18 Due Date: 9/28/18

**Abstract:**

Experiment 2 was conducted as an introduction to signal plotting in MATLAB applying what was learned about rudimentary signal analysis. The four sub-components of this experiment were: Plotting Continuous Time Signals, Plotting Discrete Time Signals, The “*conv*” Command, and Plotting a Sampled Signal. Each section of the lab dealt with a different portion of signal analysis and build on what was already learned in previous sections. After conducting the experiment, we successfully graphed continuous time signals quite simply by using the *plot* command with the required axis parameters. The discrete signals were slightly harder since they needed anonymous declarations for the delta, step, and ramp functions. The convolution section and plotting a sample signals were straight-forward since they just used previous techniques.

**Results and Discussion:**

1: Plotting Continuous Time Signals

 For the first section, the lab manual was followed carefully along with any feedback given by the TA to make the plots easier to read. The resulting code for the first section and plot outputs are given below:

subplot(3,1,1);

t = 0:2\*pi; plot(t,sin(t));

subplot(3,1,2);

t = 0:0.2:2\*pi; plot(t,sin(t));

subplot(3,1,3);

t = 0:0.02:2\*pi; plot(t,sin(t));

We can see here from running these lines that the step value between the range indicates the spacing between points when the reading is taken. For the first plot, readings are taken every one x-axis tick resulting in a choppy graph. The subsequent graphs have intervals of 0.2 and 0.02 respectively resulting in cleaner graphs better representing sinusoidal waves.

t = 0:0.02:2\*pi;

figure(1);

clf; %clears figure

plot(t,sin(t),'r'); % third argument changes color

title("My Favorite Function");

xlabel("t (Seconds)");

ylabel("y(t)");

axis([0 2\*pi -1.2 1.2]);

figure(2);

clf;

subplot(3,1,1);

t = 0:0.2:2\*pi; plot(t,sin(t),t,sin(2\*t))

subplot(3,1,2);

t2 = 0:0.2:2\*pi; plot(t,sin(t),'.');

subplot(3,1,3);

t = 0:0.2:2\*pi; plot(t,sin(t),t,sin(t),'r.');

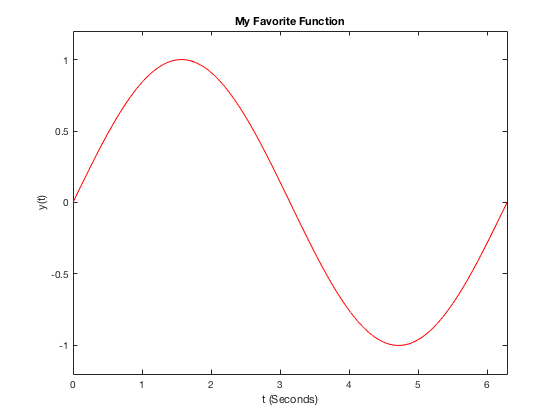
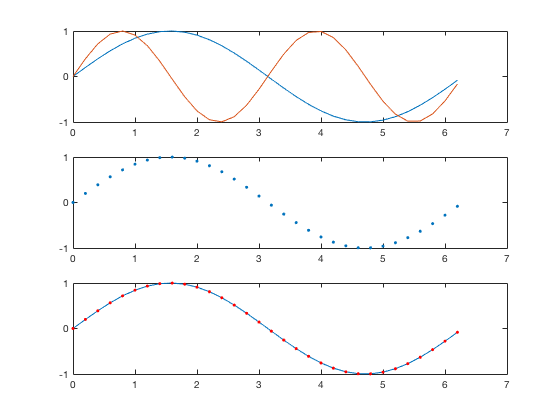
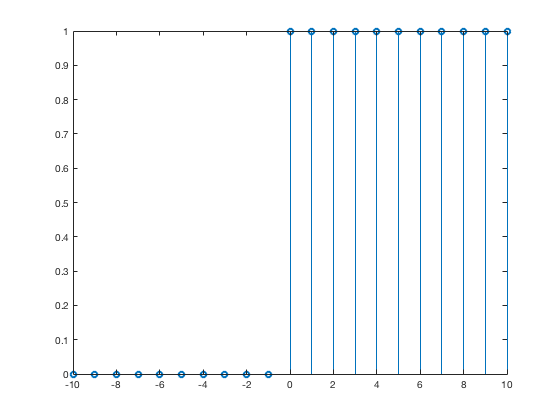
 Figure 1 Figure 2

Figure 1 in the code above prints out the simple third sinusoidal wave with points every 0.2 seconds while including a third parameter ‘r’ which changes the point color to red. Adding title, xlabel, ylabel, and axis can let you label the title of the graph, the x and y axes of the graph, and modify the axes to better represent your data. Figure 2 in the code implements a subplot of three graphs. The first plot shows 2 graphs on the same set of axes, a sin(x) and a sin(2\*x) waveform. The second subplot simply includes a ‘.’ quantifier in the plot field which simply outputs the dot values and doesn’t manually connect them. The last subplot has a ‘r.’ in the plot field representing red dots on the graph.

2: Plotting Discrete Time Signals

 The second section of the lab is similar, but we will be plotting discrete time signals instead of the previous continuous ones using the *stem* keyword. The code and outputs are as follows:

n = -10:10;

f = (n >= 0);

figure(1);

clf;

stem (n,f);

step\_f = @(n, t) n >= t;

delta\_f = @(n,t) n == t;

ramp\_f = @(n,t) (n-t).\*(n >= t);

As we can see here, we draw a stem discrete time signal plot using values of n from -10 to +10 and defining the function to check if the n value at a point is >= 0 (output is 1 if this is true and 0 otherwise). This results in our unit step function shown from the graph above. Additionally, we define the anonymous functions applying this principle to be used in the later part of the discrete signal plotting when we plot linear multiples of the delta, step, and ramp function.

figure(2);

clf;

f = step\_f(n,0) - step\_f(n,4);

stem(n, f);

axis([-10 10 0 2]);

figure(3);

clf;

g = ramp\_f(n,0) - 2\*ramp\_f(n,4) + ramp\_f(n,8);

stem(n,g);

axis([-10 10 0 5]);

figure(4);

clf;

x = delta\_f(n,0) - 2\*delta\_f(n,4);

stem(n,x);

axis([-5 5 -3 3]);

figure(5);

clf;

y = (0.9.^n) .\* ( step\_f(n,0) - step\_f(n,20) );

stem(n,y);

axis([-5 15 0 2]);

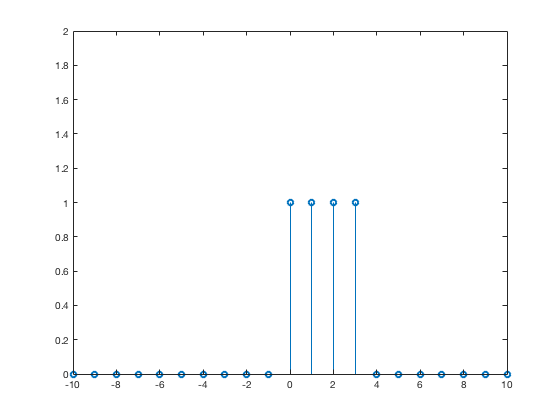
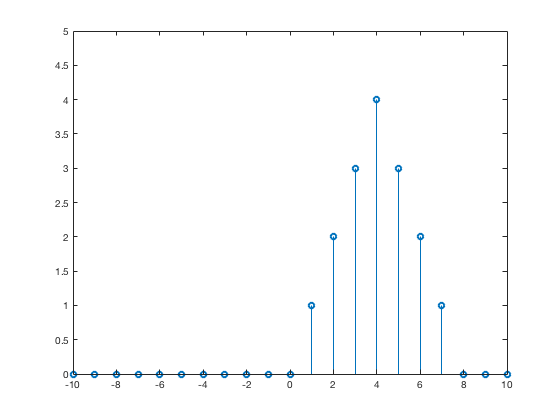
figure(6);

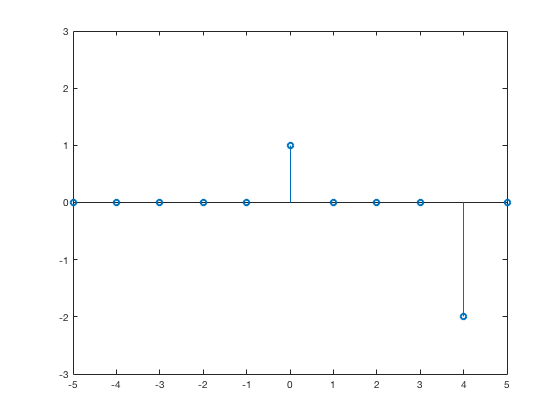
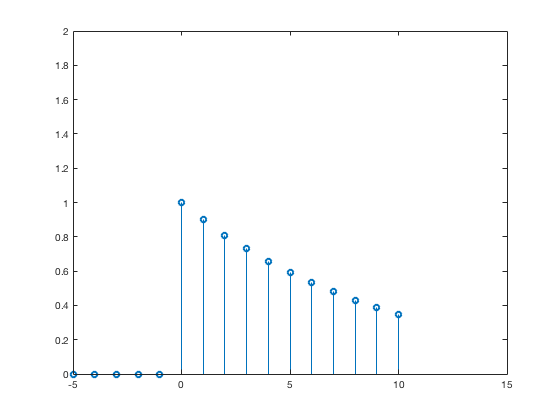
clf;

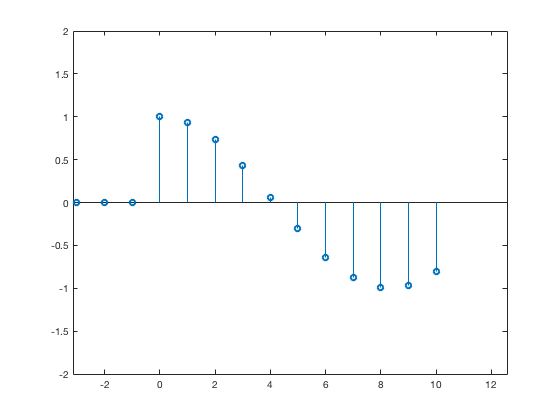
v = cos(0.12\*pi\*n) .\* step\_f(n,0);

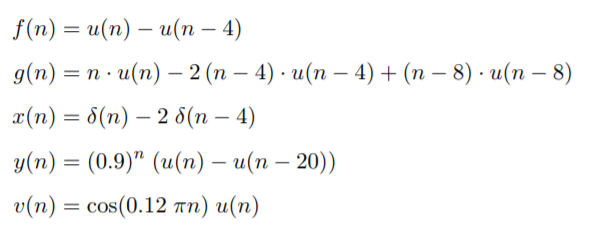
stem(n,v);

axis([-pi 4\*pi -2 2]);

Figure 2 (f) Figure 3 (g)

Figure 4 (x) Figure 5 (y)

Figure 6 (v)



From the code and equations shown, we can see how to implement the given equations in terms of the unit step, delta, and ramp functions using the anonymous functions declared earlier. Performing linear operations between these is trivial but it’s important to note that we need an element wise operator when performing an operation on the whole function (like when we use the “.^” or “.\*” in y(n) and v(n) ). The output of these functions is what we expect if we did these discrete signals quickly by hand.

3: The conv Command

The conv or convolution command helps us perform convolutions on input signals. Using help conv to learn the basics of the function, we use the f(n) and g(n) from the previous section to perform convolutions as shown:

n = -10:10;

step\_f = @(n, t) n >= t;

delta\_f = @(n,t) n == t;

ramp\_f = @(n,t) (n-t).\*(n >= t);

f = step\_f(n,0) - step\_f(n,4);

N\_f = length(f);

g = ramp\_f(n,0) - 2\*ramp\_f(n,4) + ramp\_f(n,8);

N\_g = length(g);

figure(1);

clf;

subplot(3,2,1);

ff = conv(f,f);

N\_ff = 2\*N\_f - 1;

%stem((-(N\_ff-1)/2:(N\_ff-1)/2,ff);

stem(-1\*((N\_ff-1)/2):((N\_ff-1)/2), ff);

%stem(n, f);

title("ff");

xlabel("n");

ylabel("ff(n)");

axis([-10 10 0 5]);

subplot(3,2,2);

stem(n,g);

title("g (compare with ff)");

xlabel("n");

ylabel("g(n)");

axis([-10 10 0 5]);

subplot(3,2,3);

fff = conv(ff,f);

N\_fff = N\_ff + N\_f - 1;

stem(-1\*((N\_fff-1)/2):((N\_fff-1)/2), fff);

title("fff");

xlabel("n");

ylabel("fff(n)");

axis([-5 15 0 15]);

subplot(3,2,4);

fg = conv(f,g);

N\_fg = N\_f + N\_g - 1;

stem(-1\*((N\_fg-1)/2):((N\_fg-1)/2), fg);

title("fg");

xlabel("n");

ylabel("fg(n)");

axis([-5 15 0 15]);

subplot(3,2,5);

gd = conv(g,delta\_f(n,0));

N\_gd = 2\*N\_g - 1;

stem(-1\*((N\_gd-1)/2):((N\_gd-1)/2), gd);

title("g\delta");

xlabel("n");

ylabel("g\delta(n)");

axis([-5 15 0 7]);

subplot(3,2,6);

gg = conv(g,g);

N\_gg = 2\*N\_g - 1;

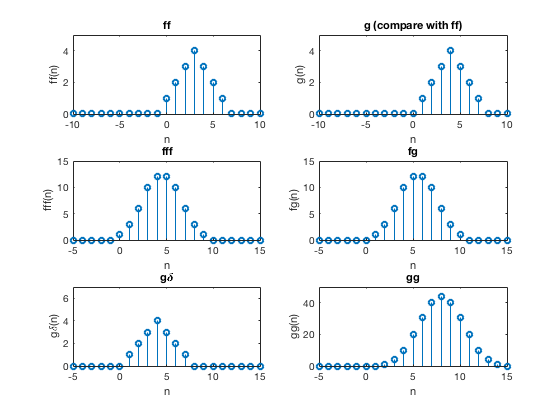
stem(-1\*((N\_gg-1)/2):((N\_gg-1)/2), gg);

title("gg");

xlabel("n");

ylabel("gg(n)");

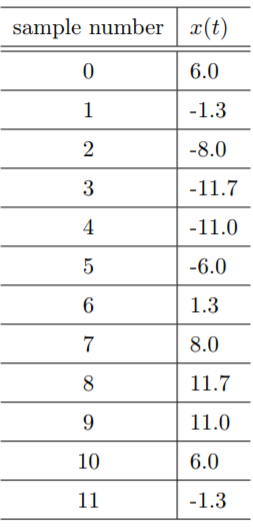
axis([-5 15 0 50]);

Figures Labelled with Corresponding Function Convolution

The convolutions in this case were relatively straightforward with the only difficult portion consisting of having the correct range for stem values which would otherwise result in incorrectly shifted data. The stem values must range from -N/2 to N/2 where N was the number of samples required for the signal convolution (product of length of the convolved signals – 1). From these plots we can observe some rather interesting and predictable characteristics. We see that f(n)\*f(n) results in the same graph as the g(n) previously shown because a convolution is just a binomial multiplication, the same form that the g(n) signal is in. We also see the pattern with the f(n), ff(n), and fff(n) graphs which results in a convolution that becomes increasingly pointed resulting in a Gaussian or Normal Distribution of Data. We can extrapolate this using the Central Limit Theorem which tells us that if we repeatedly convolve a signal with itself we will eventually get a large enough sample size with a given variance resulting in simply the mean of the total sample size.

4: Plotting a Sampled Signal

The last portion of the lab has us sample a continuous signal with a sampling period of 0.3 seconds (data is taken every 0.3 seconds) with a total of 12 samples taken who’s data is given.



x = [ 6.0, -1.3, -8.0, -11.7, -11.0, -6.0,

1.3, 8.0, 11.7, 11.0, 6.0, -1.3 ];

y = 0 : 0.3 : 0.3\*(length(x)-1);

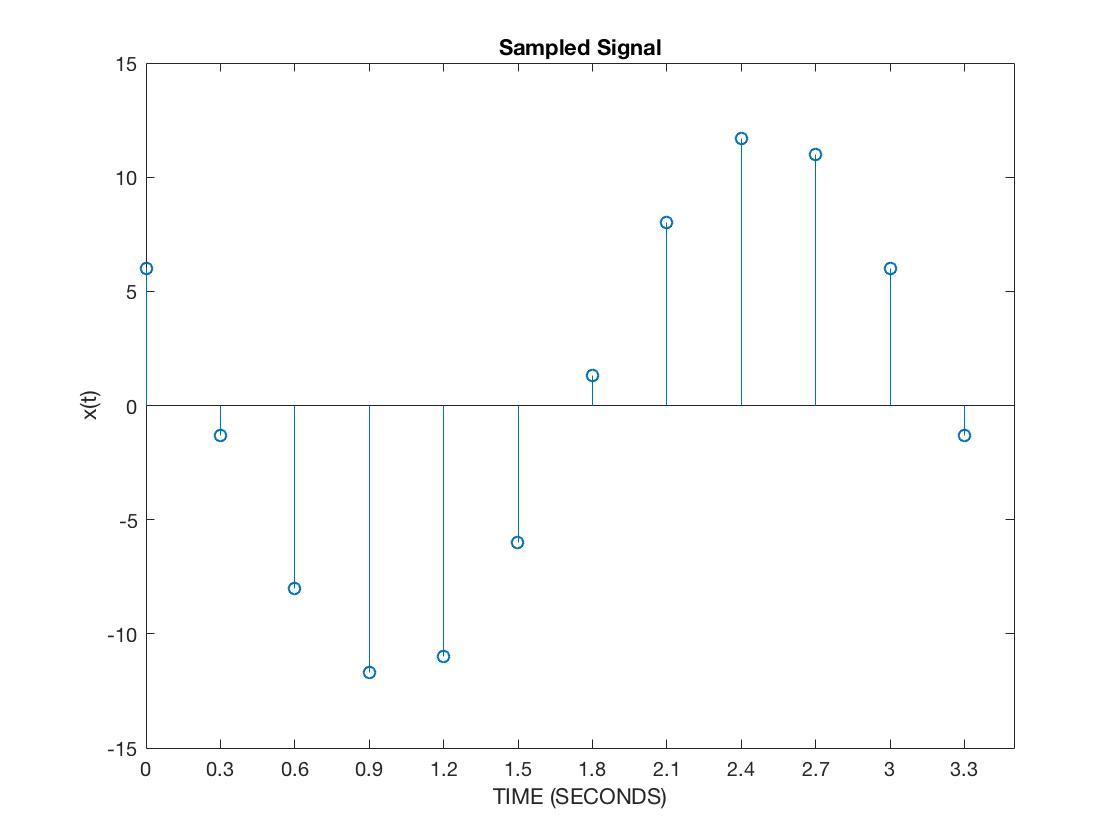
stem(y,x);

title("Sampled Signal");

xlabel("TIME (SECONDS)");

ylabel("x(t)");

set(gca,'XTick',0 : 0.3 : 0.3\*(length(x)-1));



We can see from the sampled signal table, code, and plot that the signal was a discrete sinusoidal wave since the sampling was taken at intervals. The code plots these values while changing the x-axis ticks to 0.3 seconds to make the signal easier to visualize.

**Conclusion:**

As seen from the results of the separate sections of the experiment, the required signals were successfully processed and analyzed. We worked on continuous and discrete time signals to see the difference between smaller steps and apply anonymous functions; we also learned about convolutions and signal sampling in the latter part of the lab and how to use MATLAB to represent these key attributes.