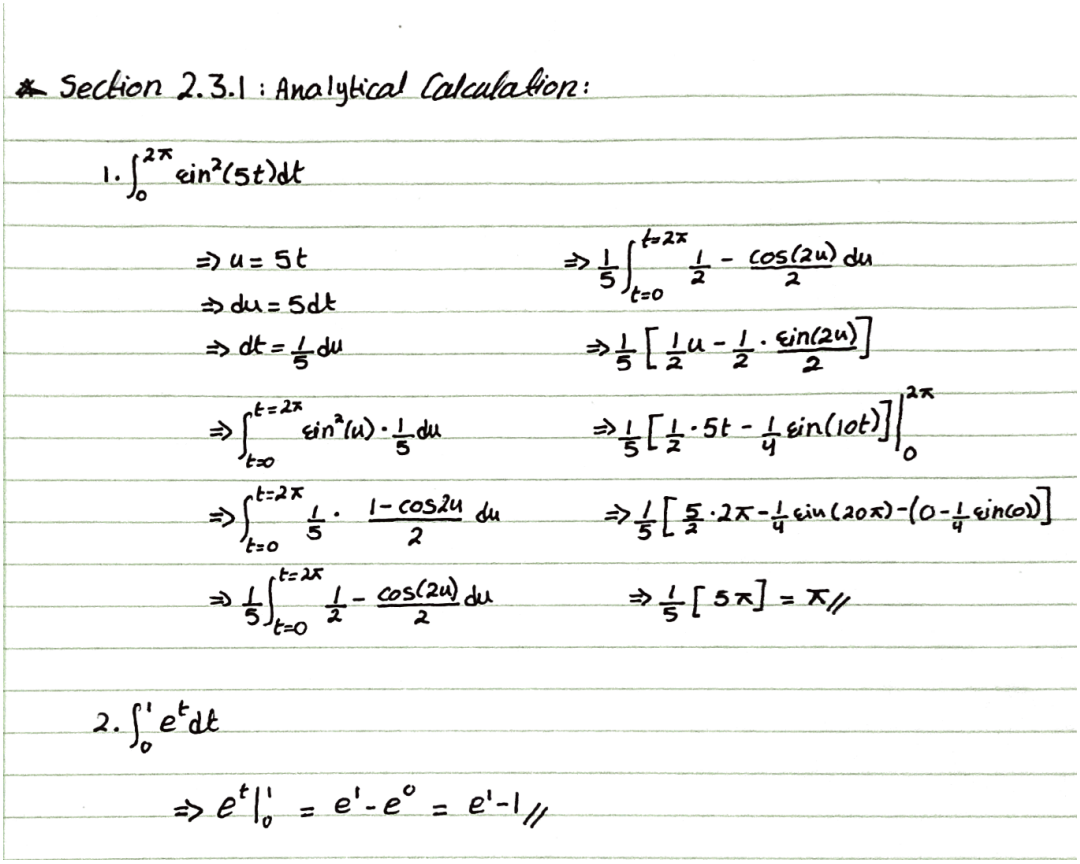
ECSE 313 Lab #1
Discrete and Continuous-Time Signals
1/28/22

Ayham Ratrout (arr90)
Kelsey Bare (khb35)

Section 2.3: Continuous-Time Vs. Discrete-Time

- 2.3.1: Analytical Calculation

For the purpose of this section of the lab, we were instructed to compute two integrals manually, showing all of our work. Furthermore, we were then told to attach our calculations of these two integrals to the lab report. With that said, attached below are our calculations of these two integrals.



* Section 2.3.1 : Analytical Calculation:

1. $\int_0^{2\pi} \sin^2(5t)\,dt$

$\Rightarrow u = 5t$

$\Rightarrow du = 5\,dt$

$\Rightarrow dt = \frac{1}{5}\,du$

$\Rightarrow \int_{t=0}^{t=2\pi} \sin^2(u)\cdot\frac{1}{5}\,du$

$\Rightarrow \int_{t=0}^{t=2\pi} \frac{1}{5}\cdot\frac{1-\cos 2u}{2}\,du$

$\Rightarrow \frac{1}{5}\int_{t=0}^{t=2\pi} \frac{1}{2} - \frac{\cos(2u)}{2}\,du$

$\Rightarrow \frac{1}{5}\int_{t=0}^{t=2\pi} \frac{1}{2} - \frac{\cos(2u)}{2}\,du$

$\Rightarrow \frac{1}{5}\left[\frac{1}{2}u - \frac{1}{2}\cdot\frac{\sin(2u)}{2}\right]$

$\Rightarrow \frac{1}{5}\left[\frac{1}{2}\cdot 5t - \frac{1}{4}\sin(10t)\right]\Big|_0^{2\pi}$

$\Rightarrow \frac{1}{5}\left[\frac{5}{2}\cdot 2\pi - \frac{1}{4}\sin(20\pi) - (0 - \frac{1}{4}\sin(0))\right]$

$\Rightarrow \frac{1}{5}\left[5\pi\right] = \pi$ //

2. $\int_0^1 e^t\,dt$

$\Rightarrow e^t\Big|_0^1 = e^1 - e^0 = e - 1$ //

- 2.3.2: Displaying Continuous-Time and Discrete-Time Signals in MATLAB

For the purpose of this section of the lab, we were instructed to start MATLAB, input a sequence of commands that is given to us by the lab manual, then observe the resulting discrete-time and continuous-time plots. Now, the first sequence of commands we were asked to type into our MATLAB workstation is as follows:

```
% Plot of a discrete-time signal formed by computing the values of the function sin(t/6) at
points which are uniformly spaced at intervals of size 2.
n = 0:2:60;
y = sin(n/6);
subplot(3,1,1);
stem(n,y);
title('Plot of a discrete-time signal formed by computing the values of the function
sin(n/6) at points which are uniformly spaced at intervals of size 2', 'Ayham Ratrout
(arr90) and Kelsey Bare (khb35)');
xlabel('n');
ylabel('sin(n/6)');
```

We can observe that this sequence of commands results in a plot displaying a discrete-time signal which is formed by computing the values of the function sin(t/6) at points which are uniformly spaced at intervals of size 2. With that said, the second sequence of commands we were asked to type into our MATLAB workstation is as follows:

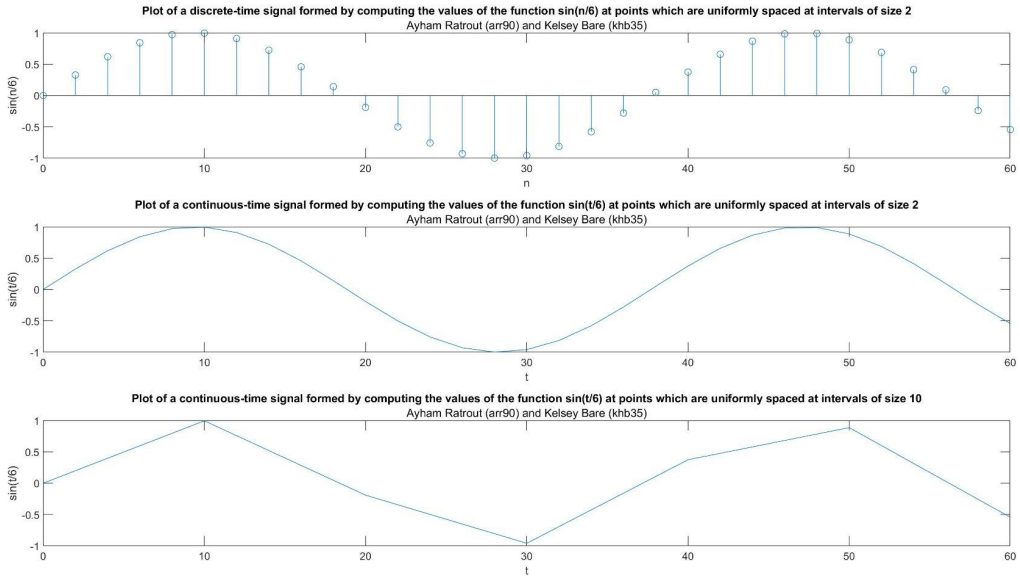
```
% Plot of a continuous-time signal formed by computing the values of the function
sin(t/6) at points which are uniformly spaced at intervals of size 2.
n1 = 0:2:60;
z = sin(n1/6);
subplot(3,1,2);
plot(n1,z);
title('Plot of a continuous-time signal formed by computing the values of the function
sin(t/6) at points which are uniformly spaced at intervals of size 2', 'Ayham Ratrout
(arr90) and Kelsey Bare (khb35)');
xlabel('t');
ylabel('sin(t/6)');
```

Contrary to the first sequence of commands, the second sequence of commands results in a plot displaying a continuous-time signal which is formed by computing the values of the function sin(t/6) at points which are uniformly spaced at intervals of size 2. Finally, the third sequence of commands we were asked to type into our MATLAB workstation is as follows:

```
% Plot of a continuous-time signal formed by computing the values of the function
sin(t/6) at points which are uniformly spaced at intervals of size 10.
n2 = 0:10:60;
w = sin(n2/6);
```

```
subplot(3,1,3)
plot(n2,w);
title('Plot of a continuous-time signal formed by computing the values of the function
sin(t/6) at points which are uniformly spaced at intervals of size 10', 'Ayham Ratrout
(arr90) and Kelsey Bare (khb35)');
xlabel('t');
ylabel('sin(t/6)');
```

Similar to the second sequence of commands, the third sequence of commands results in a plot displaying a continuous-time signal which is formed by computing the values of the function sin(t/6) at points which are uniformly spaced at intervals of size 10. With that in mind, attached below is the figure containing the, appropriately labeled, plots of the discrete-time signal as well as the two continuous-time signals in the order in which they were talked about above.



Finally, regarding the accuracy of each of the continuous-time plots, we can see that the first continuous-time plot is significantly more accurate than the second. This variation in accuracy can be attributed to the fact that the first continuous-time plot was constructed (sampled) by computing the values of the function sin(t/6) at points which are uniformly spaced at intervals of size 2 as opposed to the second plot which was constructed (sampled) by computing the values of the function sin(t/6) at points which are uniformly spaced at intervals of size 10. Therefore we can conclude that, by taking more samples, we are able to construct a more accurate continuous-time representation of a given signal.

- 2.3.3: Numerical Computation of Continuous-Time Signals

For the purpose of this section of the lab, we were instructed to write two MATLAB functions, *integ1(N)* and *integ2(N)*. Each one of the functions, numerically, computes the value of the integral of a given function defined over a specified time interval using a left Riemann Sum. As per the lab manual, the motivation behind this is to help us better understand the effects of using a different number of points to represent a continuous-time signal. To accomplish this, we started out by writing the first MATLAB function, *I=integ1(N)*, which numerically computes the integral of the of the function $sin^2(5t)$ over the interval [0, 2π] using a left Riemann Sum. The function we wrote is as follows:

```
function I = integ1(N)
   dx  = (2*pi - 0) / N;
   nArray = 0:dx:2*pi;

   % arrayfun to calculate height of each rectangle
   fullarray = arrayfun(@(x)(sin(5.*x))^2, nArray);

   %sum the elements of the array and multiply by dx for the area
   I = dx*sum(fullarray, "all");
end
```

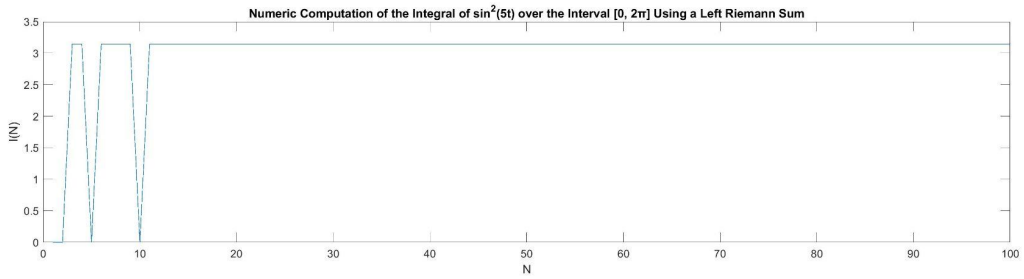
Similarly, we wrote a second MATLAB function, *J=integ2(N)*, which numerically computes the integral of the of the function $exp(t)$ over the interval [0, 1] using a left Riemann Sum. The function we wrote is as follows:

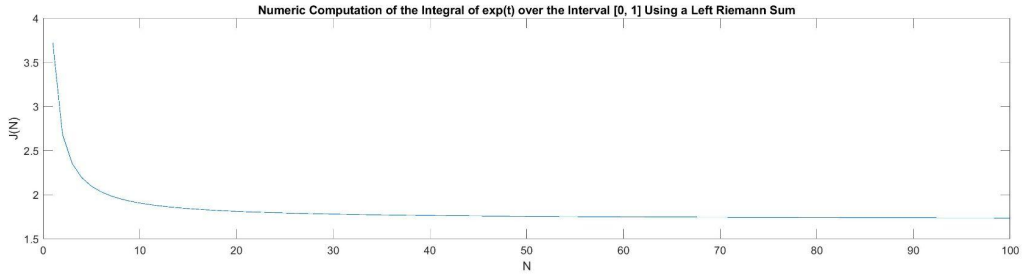
```
function J = integ2(N)
   dx  = (1 - 0) / N;
   nArray = 0:dx:1;

   % arrayfun to calculate height of each rectangle
   fullarray = arrayfun(@(x)exp(x), nArray);

   %sum the elements of the array and multiply by dx for the area
   J = dx*sum(fullarray, "all");
end
```

After writing the functions above, we were instructed to write an m-file script that evaluates both *I(N)* and *J(N)* for $1 \leq N \leq 100$, stores the results of each function call in its respective vector, then plots the resulting vectors as functions of N, each on a separate plot. The m-file script we wrote is as follows:

```
% Define vectors I and J where the results of integ1 and integ2 functions will be stored.
I = zeros(100, 0);
J = zeros(100, 0);

% For every value of i between 1 and 100, approximate the integral of sin^2(5t) using a
Riemann sum with i rectangles
for i = 1:100
   I(i) = integ1(i);
end

% Plot the vector I as a function of N
subplot(2, 1, 1);
plot(I);
title('Numeric Computation of the Integral of sin^2(5t) over the Interval [0, 2π] Using a
Left Riemann Sum');
xlabel("N");
ylabel("I(N)");

% For every value of j between 1 and 100, approximate the integral of exp(t) using a
Riemann sum with j rectangles
for j = 1:100
   J(j) = integ2(j);
end

% Plot the vector J as a function of N
subplot(2, 1, 2);
plot(J);
title('Numeric Computation of the Integral of exp(t) over the Interval [0, 1] Using a Left
Riemann Sum');
xlabel("N");
ylabel("J(N)");
```

After running the above m-file script, we obtained the following plots for I(N) and J(N), respectively:





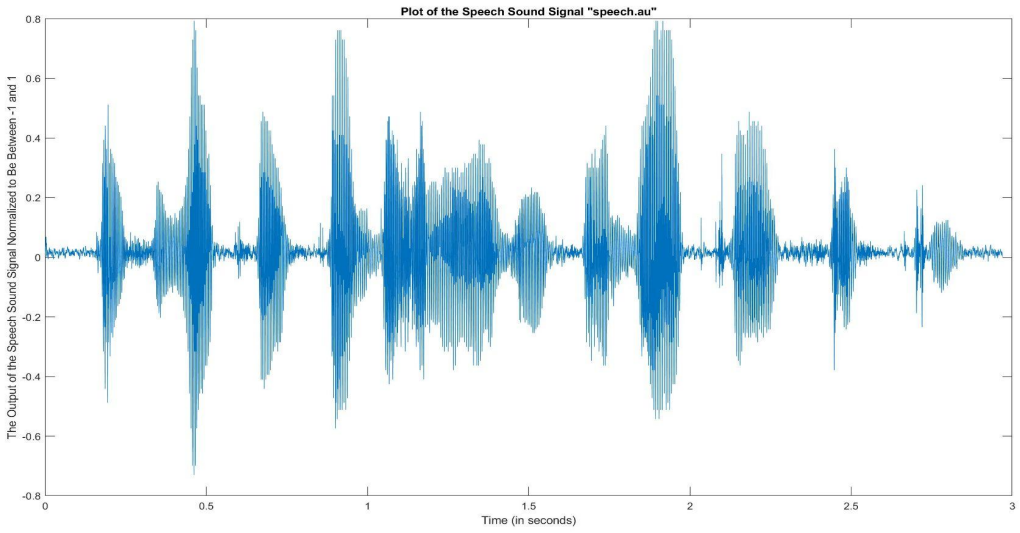
As the plots above show, the numerically computed value for the integral of the function $sin^2(5t)$ over the interval [0, 2π] falls somewhere between 3.1 and 3.2 which is approximately π. Similarly, the numerically computed value for the integral of the function $exp(t)$ over the interval [0, 1] is approximately 1.7 or e^1 - 1. Therefore, it is safe to conclude that the numerically computed values for the two integrals match closely with the analytical results obtained in *section 2.3.1 (Analytical Calculation)* hence proving the correctness of our approach. Finally, as the plot above shows, it is important to point out that *I(5) = I(10)* = 0. These peculiar values can be attributed to the fact that when dividing the interval of integration, which is [0, 2π], by either 5 or 10 to obtain dx, we will always end up with a dx that is a fraction whose denominator is 5. This, in turn, implies that when substituting dx for the variable t inside the function $sin^2(5t)$, we will always end up evaluating the the squared sine of a some integer multiple of π (i.e. +/- mπ where m is an integer) which will yield a value of zero. Therefore, when numerically computing the integral of $sin^2(5t)$ using a Riemann Sum where N = 5 or N = 10, we will end up obtaining and summing zeros for every partition across the interval which will result in a value of zero for the overall numerical value of the integral.

Section 2.4: Processing of Speech Signals

For the purpose of this section of the lab, we were instructed to process a speech audio (i.e sound) signal using MATLAB's built-in digital-to-analog converter. In order to accomplish this, we started out by downloading the speech audio file linked in the lab manual. Then, we used

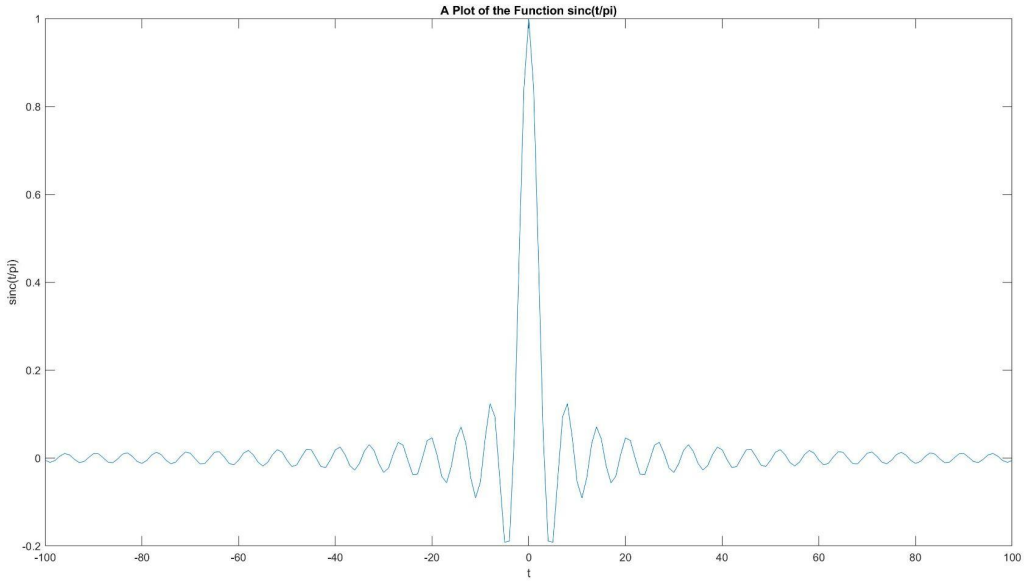
MATLAB's *audioread* command to load the speech audio file into MATLAB. After that, we plotted the speech audio signal, in the form of a continuous-time signal, using MATLAB's *plot* command. Finally, we utilized MATLAB's *sound* command for the purpose of playing the speech audio signal via MATLAB's built-in digital-to-analog converter. With that said, as per the instructions specified in the lab manual, attached below is the MATLAB code used to complete this section followed by the plot MATLAB had generated for the speech signal.

```
% Load the speech.au sound file into Matlab.
[audio, Fs] = audioread('speech.au');

% Plot the sound signal.
N = length(audio);
t = linspace(0, N/Fs, N);
plot(t, audio);
title('Plot of the Speech Sound Signal "speech.au"');
xlabel('Time (in seconds)');
ylabel('The Output of the Speech Sound Signal Normalized to Be Between -1 and 1');

% Play the sound signal via Matlab's digital-to-analog converter.
sound(audio, Fs);
```



Section 2.5: Attributes of Continuous-Time Signals

For the purpose of this section of the lab, we were instructed to write three MATLAB functions in order to calculate the basic attributes (i.e. the minimum, maximum, and energy) of the continuous-time signal, y = sinc(t/π). First, we were told to plot the function (i.e. the signal) using MATLAB's *plot* command. With that said, we were told to experiment with different values for the sampling period as well as the starting and ending times before choosing the values that yield the most accurate representation of the signal. After experimenting with a number of different values for the previously mentioned parameters, we decided that the interval that yielded the best representation of the signal y = sinc(t/π) was [-100, 100] hence why we decided to plot the signal over that interval. The plot we obtained is shown below:



After plotting the signal, we were asked to write three individual MATLAB functions in order to compute the minimum, maximum, and approximate energy of the signal y = sinc(t/π). The functions we wrote are as follows:

```
% function to find minimum of a signal
function minimum = signal1_min(t)
    minimum = min(sinc(t/pi));
end

% function to find maximum of a signal
function maximum = signal1_max(t)
    maximum = max(sinc(t/pi));
end

% function to find energy of a signal
```

```
function energy = signal1_energy
   fun = @(t) (abs(sinc(t/pi))).^2;
   energy = integral(fun, -inf, inf);
end
```

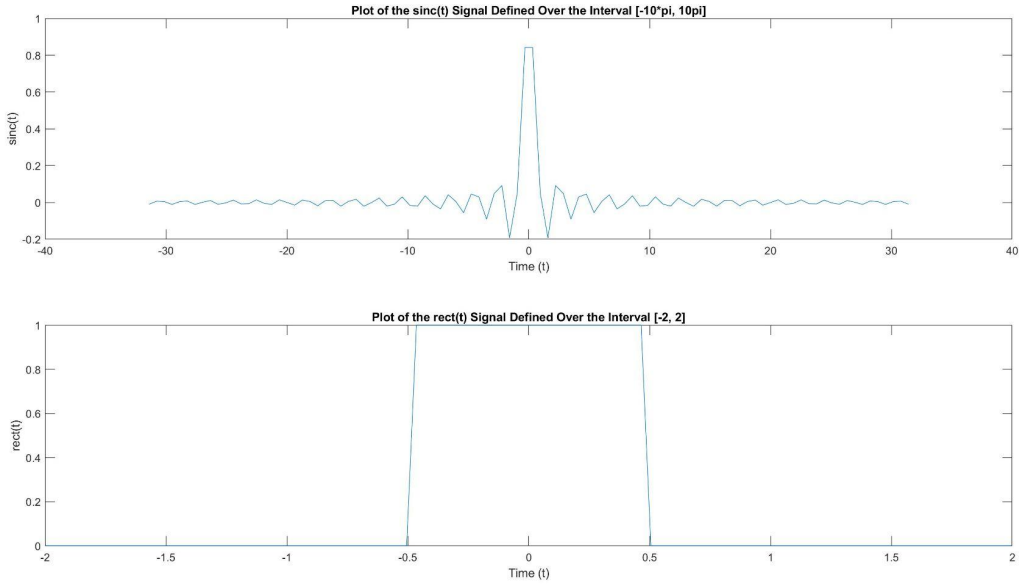
After running the above functions, we obtained that the computed values for the minimum, maximum, and energy of the signal are -0.1918, 1, and 3.1416, respectively. Finally, regarding our choice of the sampling period as well as the starting and ending times, we made sure that our starting and ending times were large enough (from -100 to 100) to ensure that we capture as much of the signal as possible as well as obtain the correct minimum and maximum values. Similarly, we choose a sampling period that is large enough to give a clear representation of the signal but not too large as to degrade performance and/or cause jitter.

Section 2.6: Special Functions

For the purpose of this section of the lab, we were instructed to generate and then plot/stem a number of different continuous-time and discrete-time functions over their specified time intervals. With that said, we were instructed to start out by plotting the *sinc(t)* and *rect(t)* continuous-time functions over the given time intervals. To accomplish this, we wrote a script file defining each function as well as its associated time interval. After that, we used the *subplot* and *plot* commands to generate one figure containing both plots. Finally, we used the *xlabel* and *ylabel* commands to ensure that the axis of each of the two graphs are labeled correctly and appropriately. With that said, attached below is the printout of the MATLAB .m-file used to generate the plots followed by the figure we obtained for the two functions indicated above.

```
% Plot the sinc(t) function/signal over the interval [-10*pi, 10pi]
t1 = linspace(-10*pi, 10*pi);
y1 = sinc(t1);
subplot(2, 1, 1);
plot(t1, y1);
title('Plot of the sinc(t) Signal Defined Over the Interval [-10*pi, 10pi]');
xlabel('Time (t)');
ylabel('sinc(t)');


% Plot the rect(t) function/signal over the interval [-2, 2]
t2 = linspace(-2, 2);
y2 = (abs(t2) <= 0.5);
subplot(2, 1, 2);
```

plot(t2, y2);
title('Plot of the rect(t) Signal Defined Over the Interval [-2, 2]');
xlabel('Time (t)');
ylabel('rect(t)');

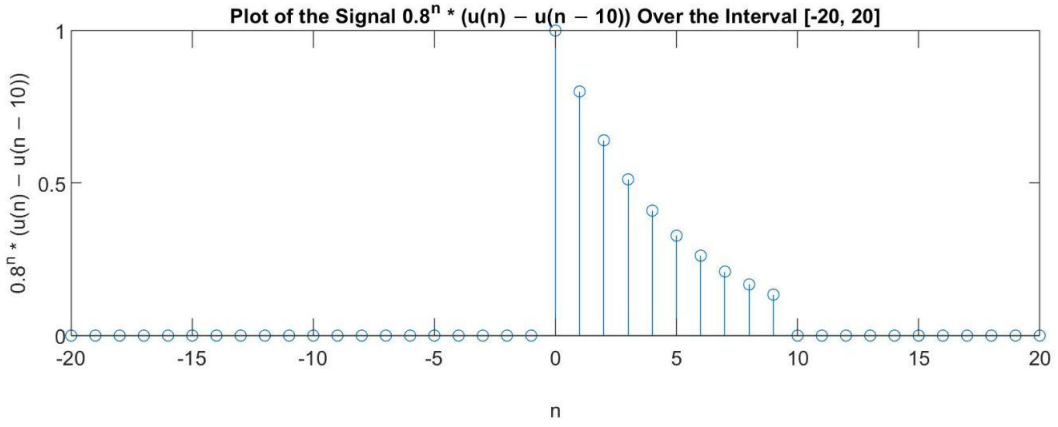


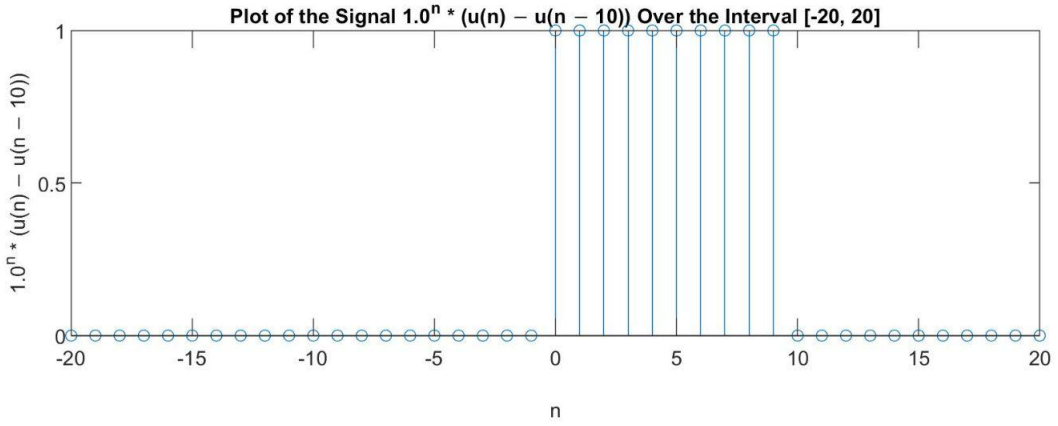
Second, we were instructed to write a MATLAB .m-file in order to stem the discrete-time function, $a^n\,(u(n) - u(n - 10))$, over the specified time interval using different values for a. To accomplish this, we wrote a script file in which we defined the discrete-time function indicated above as well as the time interval over which the function is defined and the three different values for a. After that, we used the *subplot* and *stem* commands to generate one figure containing all three plots of the function. Finally, we used the *xlabel* and *ylabel* commands to ensure that the axis of each of the three plots are labeled correctly and appropriately. With that said, attached below is the printout of the MATLAB .m-file used to generate the plots followed by the figure we obtained containing the three plots of the three variations of the function defined above.

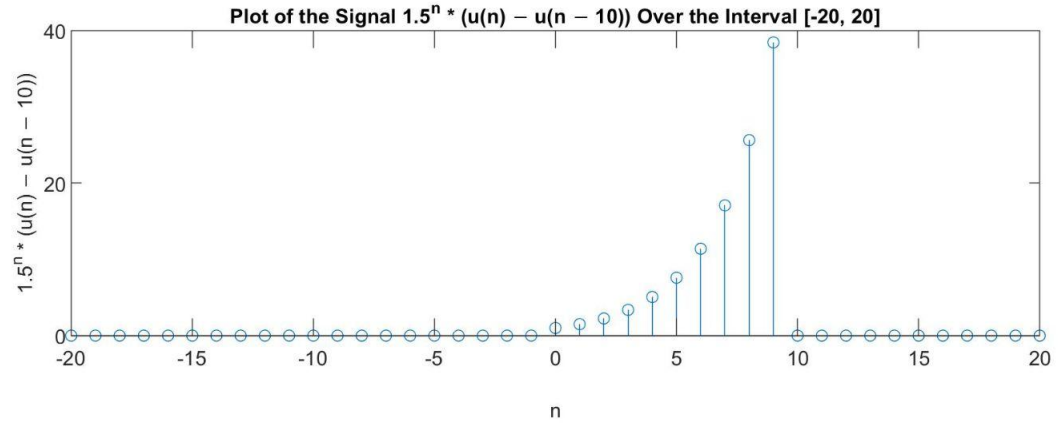
% Define the interval over which the function is defined (the discrete n values that will be substituted into the function).
n = (-20:20);

% Define the unit step function and a shifted unit step function which are helpful in the function definition for the signal we want to generate.
unit_step = n>= 0;

```
shifted_unit_step = n >= 10;

% Define the three values of a which are used in our function definition to generate the three function variations.
a1 = 0.8;
a2 = 1.0;
a3 = 1.5;

% Define and plot the first variation of the function/signal.
y1 = ((a1).^n) .* (unit_step - shifted_unit_step);
subplot(3, 1, 1);
orient('tall');
stem(n, y1);
title('Plot of the Signal 0.8^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]');
xlabel('n');
ylabel('0.8^n * (u(n) − u(n − 10))');

% Define and plot the second variation of the function/signal.
y2 = ((a2).^n) .* (unit_step - shifted_unit_step);
subplot(3, 1, 2);
orient('tall');
stem(n, y2);
title('Plot of the Signal 1.0^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]');
xlabel('n');
ylabel('1.0^n * (u(n) − u(n − 10))');

% Define and plot the third variation of the function/signal.
y3 = ((a3).^n) .* (unit_step - shifted_unit_step);
subplot(3, 1, 3);
orient('tall');
stem(n, y3);
title('Plot of the Signal 1.5^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]');
xlabel('n');
ylabel('1.5^n * (u(n) − u(n − 10))');
```


Plot of the Signal 0.8^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]
0.8^n * (u(n) − u(n − 10))
n
1
0.5
0
-20
-15
-10
-5
0
5
10
15
20

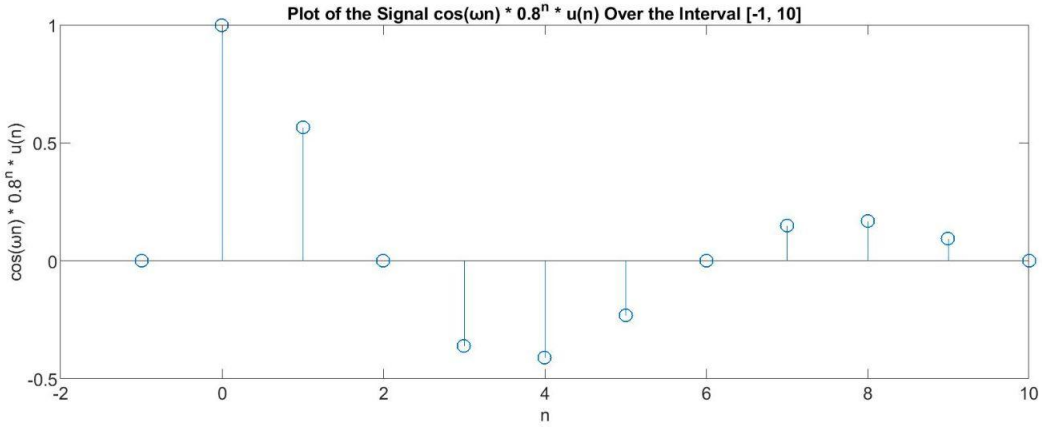

Plot of the Signal 1.0^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]
1.0^n * (u(n) − u(n − 10))
n
1
0.5
0
-20
-15
-10
-5
0
5
10
15
20

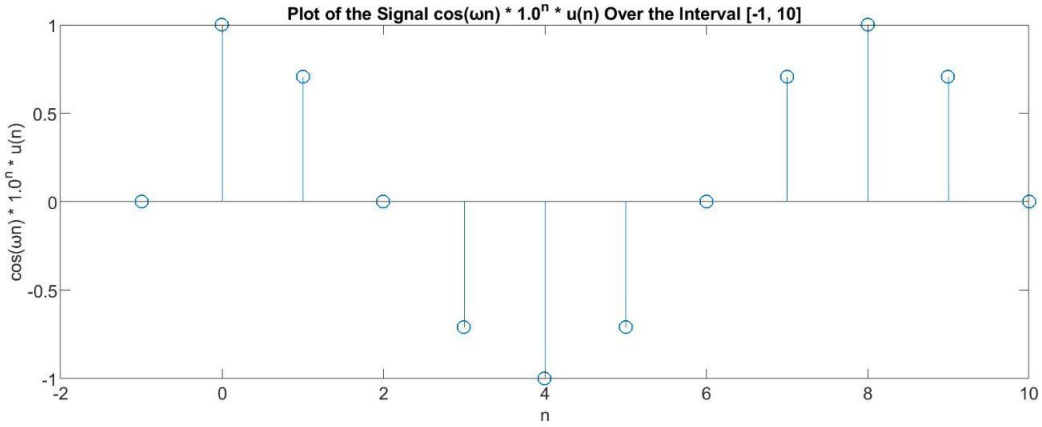

Plot of the Signal 1.5^n * (u(n) − u(n − 10)) Over the Interval [-20, 20]
1.5^n * (u(n) − u(n − 10))
n
40
20
0
-20
-15
-10
-5
0
5
10
15
20

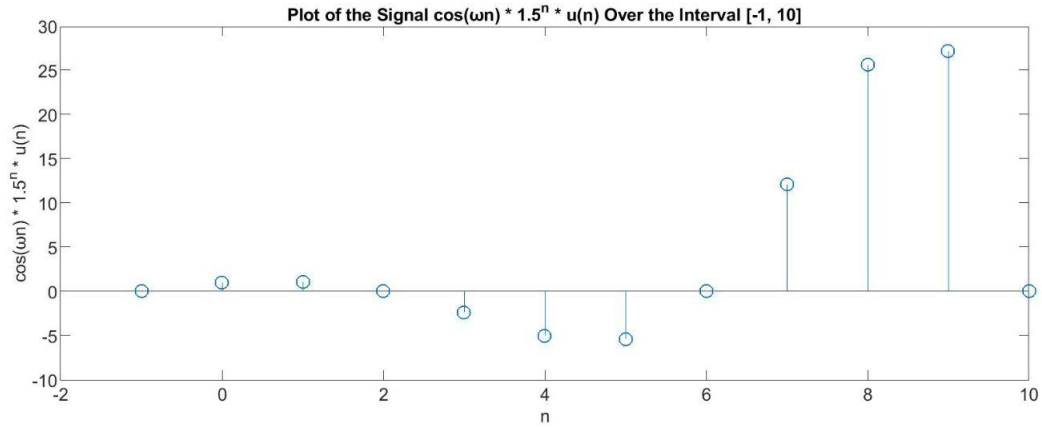
Finally, we were asked to repeat the same process as above to stem the discrete-time function, $cos(\omega n)\ a^n\ u(n)$, over the specified time interval using different values for a. With that said, attached below is the printout of the MATLAB .m-file used to generate the plots followed by the figure we obtained containing the three plots of the three variations of the function defined above.

```
% Define the interval over which the function is defined (the discrete n values that will be substituted into the function).
n = (-1:10);

% Define a value for omega to make the code more readable and organized.
omega = pi / 4;

% Define the unit step function which is helpful in the function definition for the signal we want to generate.
unit_step = n>= 0;

% Define the three values of a which are used in our function definition to generate the three function variations.
a1 = 0.8;
a2 = 1.0;
a3 = 1.5;

% Define and plot the first variation of the function/signal.
y1 = cos(omega .* n) .* ((a1).^n) .* unit_step;
subplot(3, 1, 1);
orient('tall');
stem(n, y1);
title('Plot of the Signal cos(ωn) * 0.8^n * u(n) Over the Interval [-1, 10]');
xlabel('n');
ylabel('cos(ωn) * 0.8^n * u(n)');

% Define and plot the first variation of the function/signal.
y2 = cos(omega .* n) .* ((a2).^n) .* unit_step;
subplot(3, 1, 2);
orient('tall');
stem(n, y2);
title('Plot of the Signal cos(ωn) * 1.0^n * u(n) Over the Interval [-1, 10]');
xlabel('n');
```

```
ylabel('cos(ωn) * 1.0^n * u(n)');

% Define and plot the first variation of the function/signal.
y3 = cos(omega .* n) .* ((a3).^n) .* unit_step;
subplot(3, 1, 3);
orient('tall');
stem(n, y3);
title('Plot of the Signal cos(ωn) * 1.5^n * u(n) Over the Interval [-1, 10]');
xlabel('n');
ylabel('cos(ωn) * 1.5^n * u(n)');
```

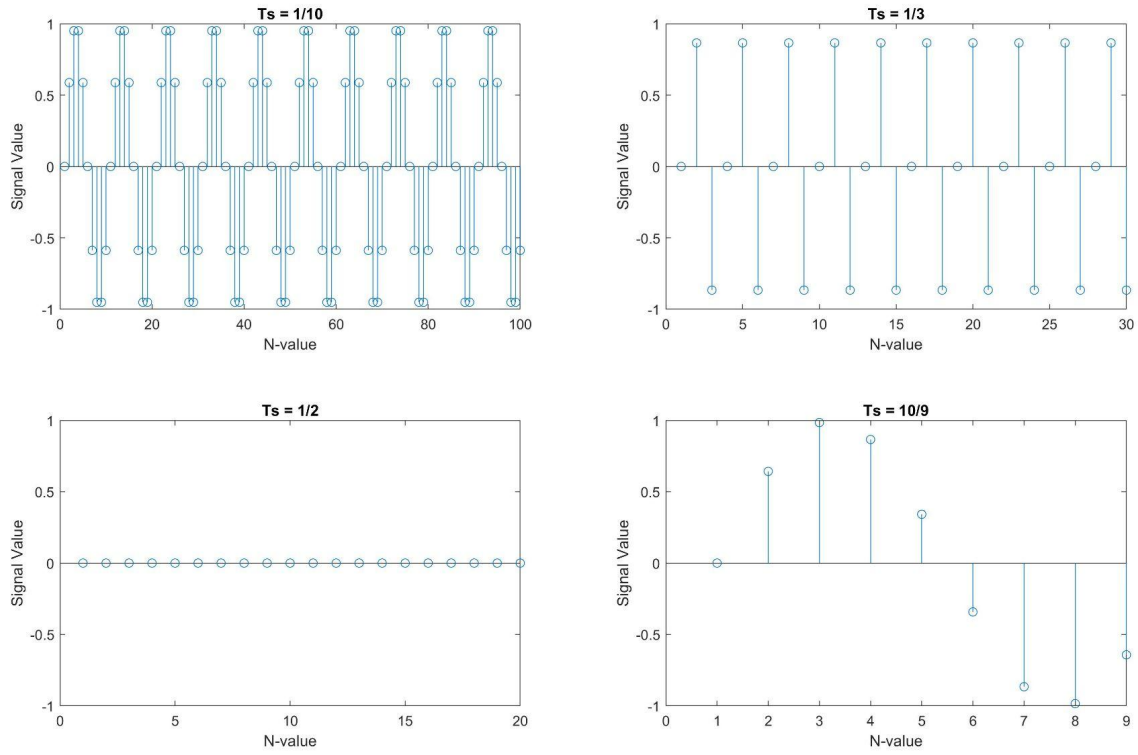

Plot of the Signal cos(ωn) * 0.8^n * u(n) Over the Interval [-1, 10]
cos(ωn) * 0.8^n * u(n)
n
1
0.5
0
-0.5
-2
0
2
4
6
8
10


Plot of the Signal cos(ωn) * 1.0^n * u(n) Over the Interval [-1, 10]
cos(ωn) * 1.0^n * u(n)
n
1
0.5
0
-0.5
-1
-2
0
2
4
6
8
10


Plot of the Signal cos(ωn) * 1.5^n * u(n) Over the Interval [-1, 10]
cos(ωn) * 1.5^n * u(n)
n
30
25
20
15
10
5
0
-5
-10
-2
0
2
4
6
8
10

Section 2.7: Sampling

When Ts = 1/10, there are 10 full periods of something that looks like a sine wave visible in the graph. With Ts = 1/3, there are still 10 full periods, but instead of having multiple points per wave, the only visible points are the min, max, and the zeros of the function. Because of the different sampling size, not as much of the sine wave is visible within the graph. When Ts =1/2, the graph only shows the zeroes of the sine wave. This is because Ts = 1/2 means that the input to the sine wave is always a multiple of pi, which is always 0 for sine. Ts = 10/9 plots something that looks approximately like one cycle of a sine wave. If we had to choose a sampling value, we would choose Ts = 1/10 because it kept most of the original function and minimized the amount of missing information from the continuous time function.



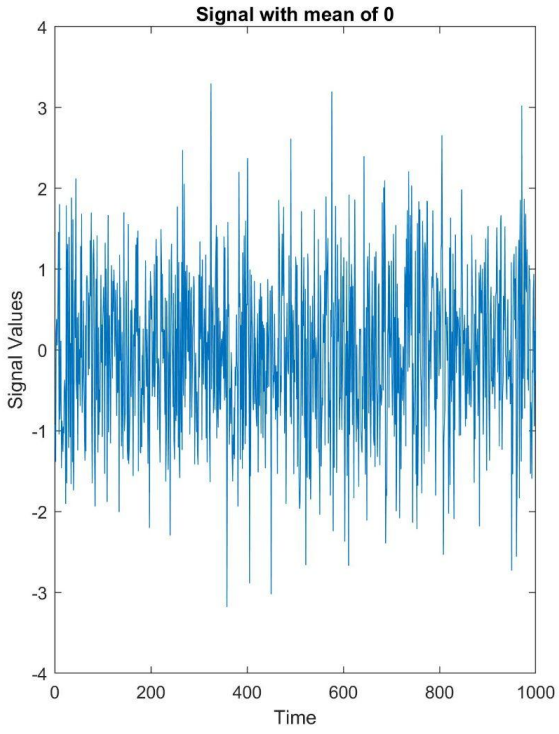
MATLAB Code for Section 2.7:

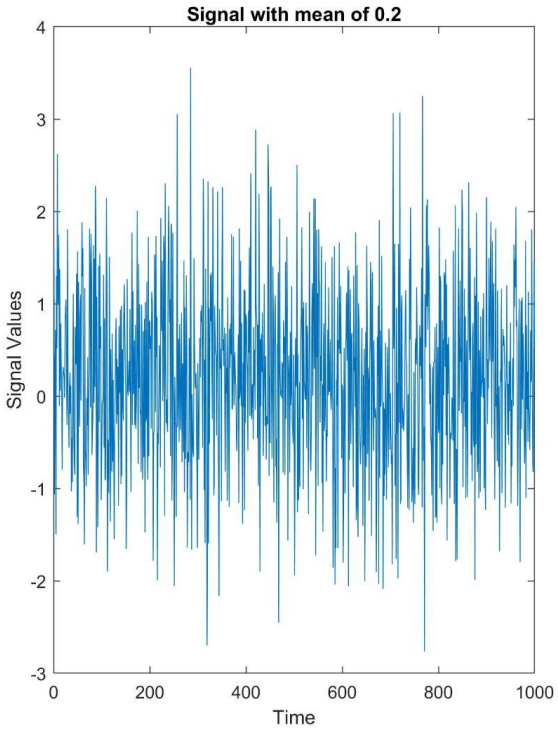
```
%% 2.7 Sampling
% Plots a signal 4 times, using different sampling rates to show the differences

% First plot- Ts of 1/10 from 0 to 100
Ts1 = 1/10;
n1 = 0:100;
signal1 = sin(2*pi*Ts1*n1);
subplot(2,2,1);
```

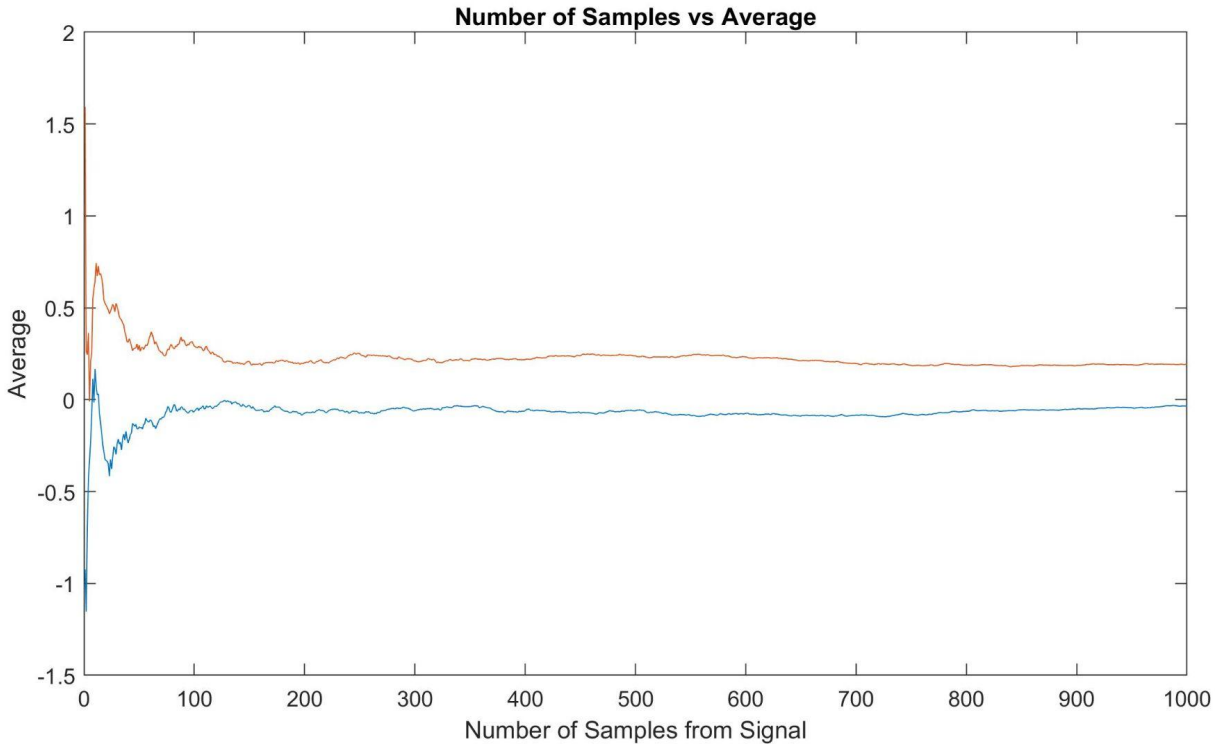
```
one_tenth_plot = stem(signal1);
axis([0,100,-1,1])
xlabel("N-value");
ylabel("Signal Value");
title('Ts = 1/10');

% Second plot- Ts of 1/3 from 0 to 30
Ts2 = 1/3;
n2 = 0:30;
signal2 = sin(2*pi*Ts2*n2);
subplot(2,2,2);
one_third_plot = stem(signal2);
axis([0,30,-1,1]);
xlabel("N-value");
ylabel("Signal Value");
title('Ts = 1/3');

% Third Plot- Ts of 1/2 from 0 to 20
Ts3 = 1/2;
n3 = 0:20;
signal3 = sin(2*pi*Ts3*n3);
subplot(2,2,3);
one_half_plot = stem(signal3);
axis([0,20,-1,1]);
xlabel("N-value");
ylabel("Signal Value");
title('Ts = 1/2');

% Fourth Plot- Ts of 10/9 from 0 to 9
Ts4 = 10/9;
n4 = 0:9;
signal4 = sin(2*pi*Ts4*n4);
subplot(2,2,4);
ten_ninths_plot = stem(signal4);
axis([0,9,-1,1]);
xlabel("N-value");
ylabel("Signal Value");
title('Ts = 10/9');
saveas(gcf, "ecse313_lab1_plot2.7.jpg")
```

Section 2.8: Random Signals

Both signals look approximately the same when plotted. In the average functions, both average values started off fluctuating extremely, but quickly settled to an average value of whichever the mean was. In both cases, the graph of the average had a horizontal asymptote at x = 0 or x =0.2, corresponding to the mean of the randomly-generated signal. As a result, these average values can be used to distinguish between two signals that both have random noise but different means. The mean of the signal is the value that ave(n) goes to as n increases.







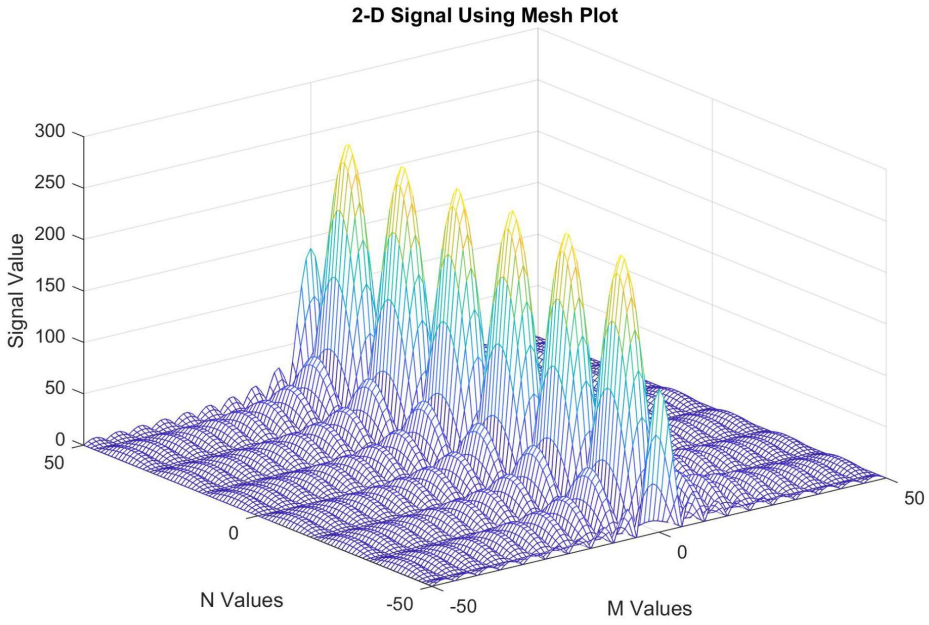
MATLAB Code for Section 2.8:

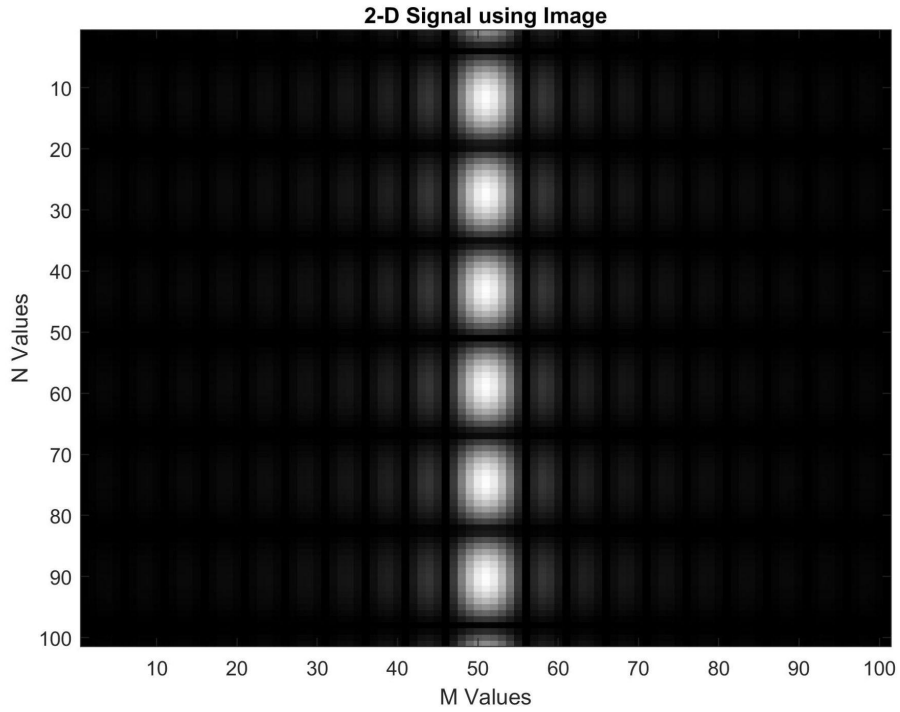
%% 2.8 Random Signals

```
% Generates a random signal and shows how the signal's average can be used to distinguish
between two random signal

% generate signals
sig1 = random('norm', 0, 1, 1, 1000);
sig2 = random('norm', 0.2, 1, 1, 1000);

% plot signals in the same figure
figure
subplot(1, 2, 1);
plot(sig1);
title("Signal with mean of 0");
xlabel("Time")
ylabel("Signal Values")
subplot(1, 2, 2);
plot(sig2);
title("Signal with mean of 0.2");
xlabel("Time")
ylabel("Signal Values")

% find averages using a for loop
for n=1:1000
    ave1(n) = mean(sig1(1:n));
    ave2(n) = mean(sig2(1:n));
end

n= 1:1000;
figure
plot(n,ave1);
hold on
plot(n,ave2);
title("Number of Samples vs Average");
xlabel("Number of Samples from Signal")
ylabel("Average")
```

Section 2.9: 2-D Signals

The surface plot works better to visualize the signal in three dimensions. One possible application would be a plot of a signal in 2 dimensions throughout time, or other applications where information in all three dimensions needs to be conveyed. The image plot works better when the most important information is the value of the signal, and only 2 dimensions need to be seen. The image emphasizes only the value of the signal, and as a result is not suitable for applications where a more comprehensive view of the signal is necessary.





MATLAB Code for Section 2.9:

```
%% 2.9 2-D Signals
% Plots a 2-D Signal on a meshplot and as an image

% generate input
[X Y] = meshgrid(-50:50);

% generate signal
mesh_signal = 255.*(abs((sinc(0.2.*X)).*(sin(0.2.*Y))));

% plot the signal with mesh
mesh(X, Y, mesh_signal);
xlabel("M Values")
ylabel("N Values")
zlabel("Signal Value")
title("2-D Signal Using Mesh Plot");

% display as image
figure
image(mesh_signal);
title("2-D Signal using Image")
xlabel("M Values")
```

```
ylabel("N Values")
colormap(gray(256));
```