California State University Long Beach

College Of Engineering

Computer Engineering and Computer Science Department

CECS 463 – Digital Signal Processing

Instructor: Haney W. Williams

Project 2

By: Aishwarya Ravishankar, Alina Suon, Antoine Gannat, Allura Jackson

Table of Contents

Part 1

Problem Description: Find the expression for the complex Fourier coefficients for the square wave signal f(t) from: g(t) if the reconstruction of f(t) using the coefficients. Using the coefficients (cn), plot two square waves of period T=1 over interval -1.1 to 1.1. Plot the absolute value of the error between the points in f(t) and its estimate. Also calculate between the two

1

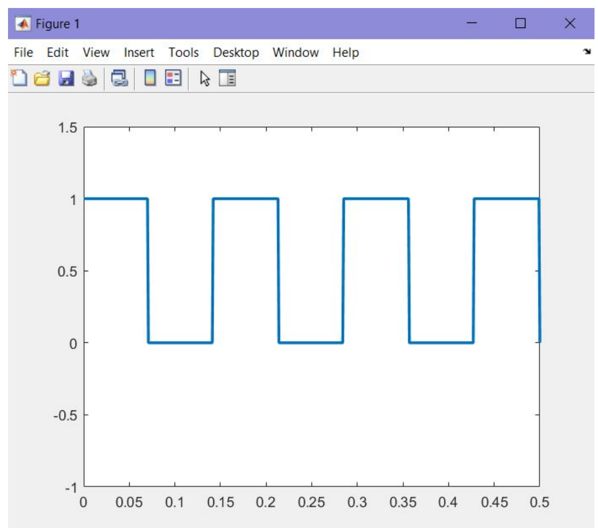
a)

Solution:

```
close all; clear all;
W_n = 7;
c_n = 1000;
cn = 0.5;
Wn = [0:1/c_n:cn];
cycles = cn*W_n;
c0 = ones(1,length(Wn));
duty = 50;
oc_samp = c_n/W_n;
on_samp = (oc_samp * duty)/100;
off_samp = oc_samp - on_samp;
gN = 0;
for i = 1 : ceil(cycles);
    c0(gN+on_samp+1:i*oc_samp) = 0;
    gN = gN + oc_samp;
end
plot(Wn,c0(1:length(Wn)),'LineWidth',2);y
lim([-1 1.5]);
% stem(Wn,c0); ylim([-1.5 1.5]);
```

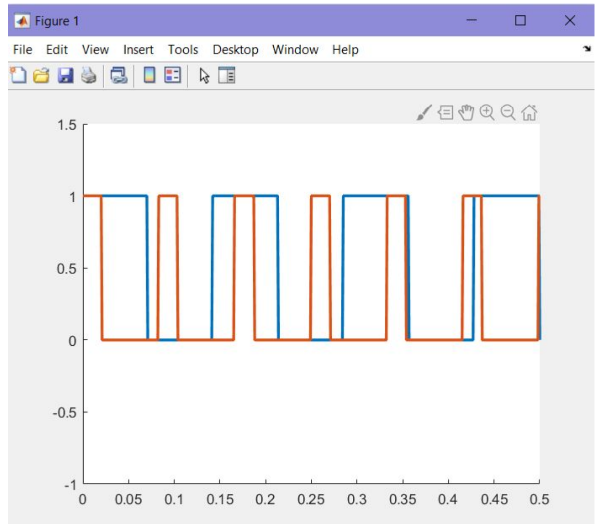
Expressions for Complex Fourier Coefficients:

Results:



b)

```
close all; clear all;
hold on
W_n = 7;
c_n = 1000;
cn = 0.5;
Wn = [0:1/c_n:cn];
cycles = cn*W_n;
c0 = ones(1,length(Wn));
duty = 50;
oc_samp = c_n/W_n;
on_samp = (oc_samp * duty)/100;
off_samp = oc_samp - on_samp;
gN = 0;
for i = 1 : ceil(cycles);
    c0(gN+on_samp+1:i*oc_samp) = 0;
    gN = gN + oc_samp;
end
plot(Wn,c0(1:length(Wn)),'LineWidth',2);y
lim([-1 1.5]);
% stem(Wn,c0); ylim([-1.5 1.5]);

W_n = 12;
c_n = 1000;
cn = 0.5;
Wn = [0:1/c_n:cn];
cycles = cn*W_n;
c0 = ones(1,length(Wn));
duty = 25;
oc_samp = c_n/W_n;
on_samp = (oc_samp * duty)/100;
off_samp = oc_samp - on_samp;
gN = 0;
for i = 1 : ceil(cycles);
    c0(gN+on_samp+1:i*oc_samp) = 0;
    gN = gN + oc_samp;
end
plot(Wn,c0(1:length(Wn)),'LineWidth',2);y
lim([-1 1.5]);
% stem(Wn,c0); ylim([-1.5 1.5]);
hold off
```

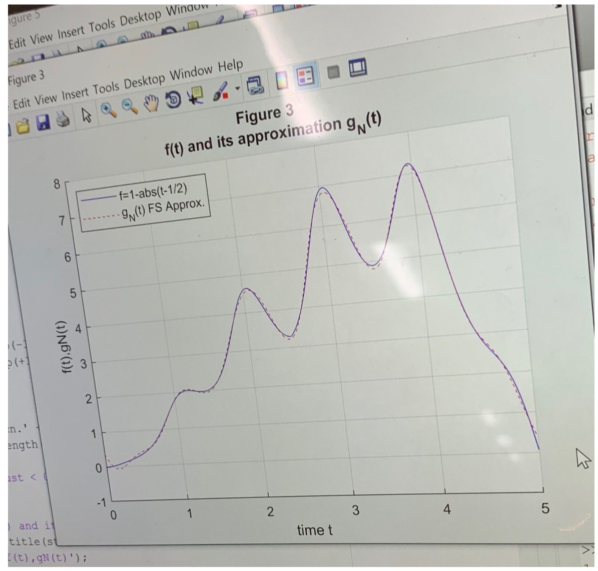


2.

Solution:

```
% number 3

plot(t,f,'b'); grid on;

%find approximate
cn=1/T*sum(f*dT);
A=1;
for n=1
        cn(n) = dT/T *
sum(f.*exp(-1j*2*pi*n*t/T));
        c_n(n)=dT/T
*sum(f.*exp(1j*2*pi*n*t/T));
end
n=1:N;
Zn=exp(1k*2*pi/T*t'*n);
Z_n=conj(Wn); gN = (c0+Zn*cn.' +Z_n*c_n.'
)';
rmse = sqrt(sum(abs(f-gN).^2) /
length(f));

str=sprintf('3. RMSE = %8.6f just < 0.075
using %i coefficients', rmse, N);
disp(str);

str=sprintf('Figure 3/n f(t) and its
approximation g_N(t)');
plot(t,GN,'---r');

grid on; title(str);
xlabel('time t');
ylabel ('f(t),gN(t)');
legend('f=1-abs(t-1/2)','g_N(t) Approx.',
'Location','northwest');
```

Results:



4.

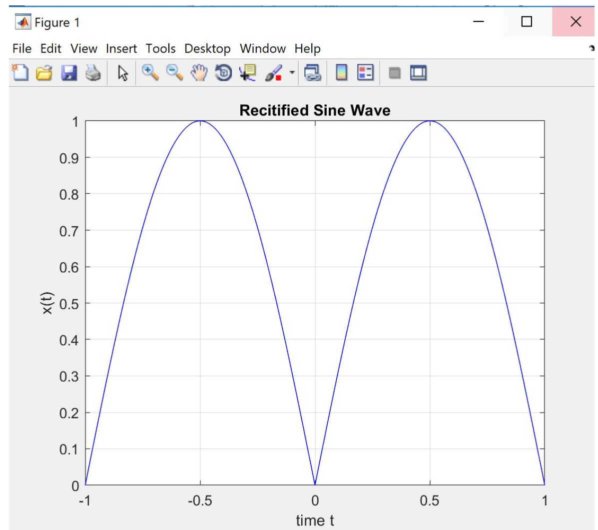
Solution:	***Results:***
`` % number 4 figure() t = -1/1:.01:1.1; T = 2*pi; n = 0:20; c0 = 01; cn = zero([1 21]); for k = 1:2:20 cn(k+1) = -(1/pi/k)*((-1)/^((k-1)/2)); end c_n=conj(cn); Zn=exp(1j*2*pi/T*2*pi*(t)'*n); Z_n =conj(Zn); gN= c0+Zn * cn.'+Z_n*c_n.'; f=0,5*sign(sin(2*pi*(t-.25))); f=f.'; hold on plot(t,f,'blue') plot(t,gN,'red') axis([-1.1,1.1,-1,1,1,1]); title('Periodic Square Wave'); xlabel('time t'); ylabel('f(t)'); hold off subplot(3,1,1) plot(t,f_1) subplot(3,1,2) plot(t,f) subplot(3,1,3) plot(t,gN) rmse = sqrt(sum(abs(f-gN).^2)/length(f)); str = sprintf('RMS ERROR x(t)-g_n(t) with RMSE = %6.4f',rmse); title(str); ```	

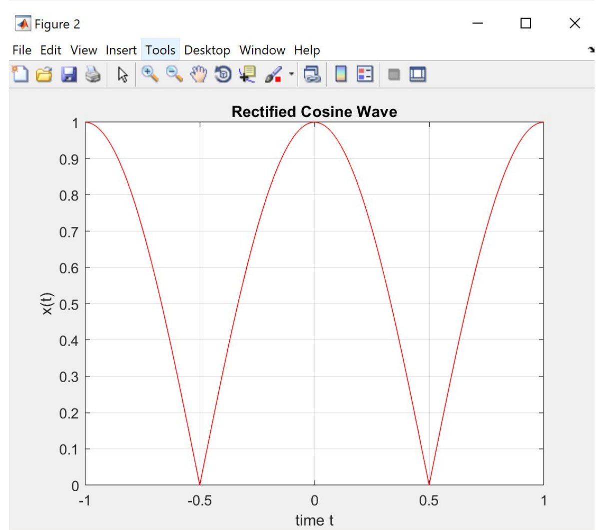
5.

Solution:

```
% number 5

figure()
T=2;
dT=T/1000;
t=-T/2:dT:T/2;

f=abs(sin(2*pi/T*t));
plot(t,f,'b');
grid on;
title('Rectified Sine Wave');
xlabel('time t');
ylabel('x(t)');

t_0= -0.25*T;

p = abs(sin(2*(pi/T*(t-t_0))));

figure()

plot(t,p,'r')
grid on;
title("Rectified Cosine Wave");
xlabel('time t')
ylabel('x(t)');
```

Results Solution:





Encountered Problems: In this section we were having issues with the wording and recognizing how to solve the problem because we haven't seen this before.

Part 2

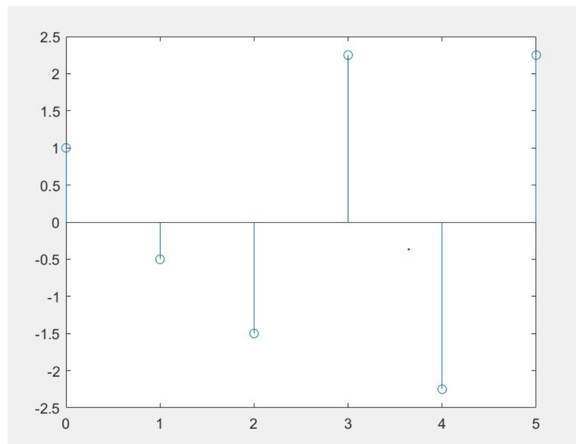
Problem Description: The filter function computes the output of a causal LTI system for a input sequence when the system is specified by a linear constant coefficient difference equation of the form. When x = 1,2,3,4 the output for the following LTI difference equations are below.

1.

Solution:

```
%% Project Part 2 # 1
x  = [1 2 3 4];
a1 = [1 0 0 0];
b1 = [0.5 1 2 0];
z1 = filter(b1,a1,x)
a2 = [1 -0.8 0 0];
b2 = [2 0 0 0];
z2 = filter(b2,a2,x)
a3 = [1 -0.8 0 0];
b3 = [0 2 0 0];
z3 = filter(b3,a3,x)
```

Results:



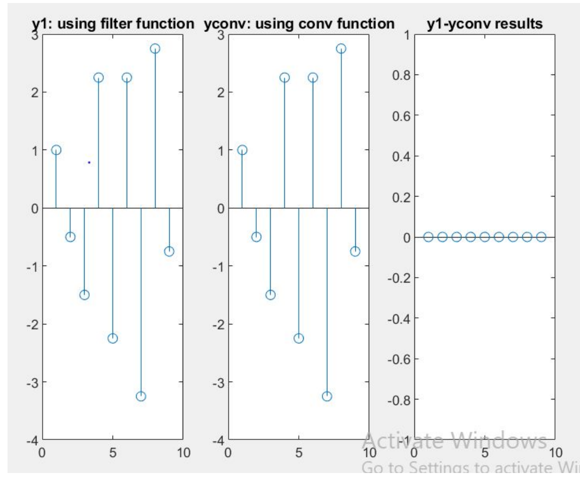
2.

Problem Description: When x[n] = -1.^n over an interval from 0-5 and when y is an set equation, we're trying to find the vector b for the LTI equation. If the filter is to generate the same output as conv by plotting the vector. Then we compare the result to see if the difference is identical zero at all points.

Solution:

```
%% Part 2 # 2
ny = 0:5;
h = [1 0.5 -2 0.75];
x2 = (-1).^ny;
y = filter(h,1,x2)
figure
stem(ny,y)
x1 = [x2 0 0 0];
y1 = filter(h,1,x1)
yconv = conv(h,x2)
ytotal = y1-yconv
figure
subplot(1,3,1)
stem(y1), title('y1: using filter function');
subplot(1,3,2)
stem(yconv), title('yconv: using conv function');
subplot(1,3,3)
```

Results:



Encountered Problems*: Through this section we had big problems trying to understand the wording of the problem but when we did understand what its asking for we were able to finish this section under 30 minutes with the help of the powerpoints.*

Part 3

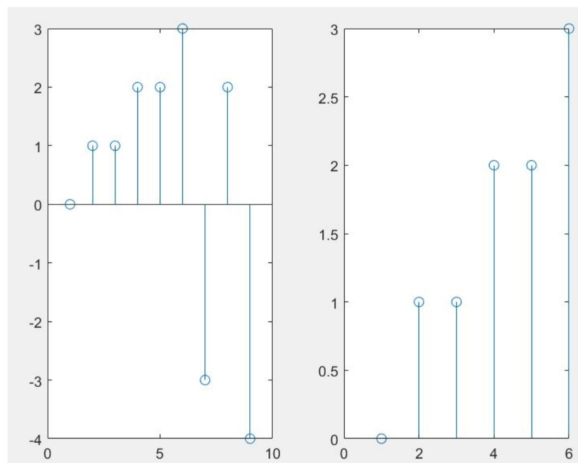
Problem Description: Usually recorded signals are computed by noises, such as echoes. These certain problems attempts to create an echo and then to remove the echo by signal processing by creating an echo signal and plotting them to compare. Then we determine the impulse response and stores the impulse response in a vector. By showing that the equation is indeed an inverse of the first equation by deriving the overall difference equation. Then we store the approximation to the infinite impulse response inthe vector. After that we implemented the echo removal signal and calculated the overall impulse response of the cascaded echo system.

1.

Solution:

```
%%
x = mtlb;
Nx = x + zeros(4001,1);
D = 2750;
a = zeros(2751,1);
a(1,1) = 1;
a(2751,1) = 0.9;
b = 1;
y = filter(b,a,Nx);
sound(y,Fs)
figure
subplot(3,1,1)
plot(x)
subplot(3,1,2)
plot(y)
```

Results:



2.

Solution:	***Results:***
`%%` `n= 0: Nx;` `he = impz(b,a,n);`	No Results just stores the impulse vector to he.

3.

Solution:	***Results:***
`%y[n] = x[n] +ax[n-D]` `%z[n] + az[n-D] = x[n]+ax[n-D]`	No result but from this we can see if there is a valid solution to the overall difference equation.

4.

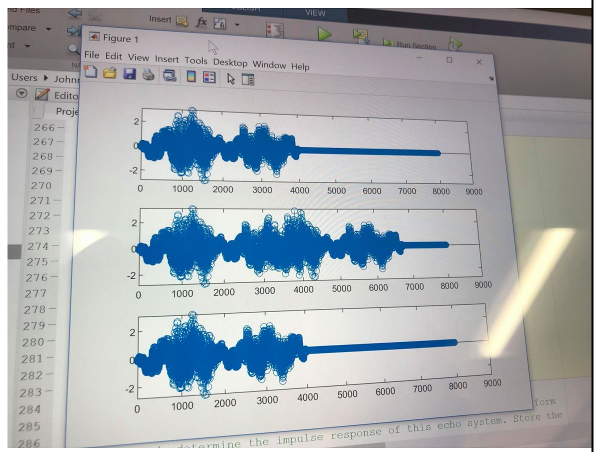
Solution:	***Results:***
`d= [1,50000];` `her = filter(d,a,n);`	No result but the approximation to the infinite impulse response in the vector her.

5.

Solution:

```
a=b;
w= filter(1,a,y);
nw= 0: length(w)-1;
subplot(3,1,3);
stem(nw,w);
```

Results:

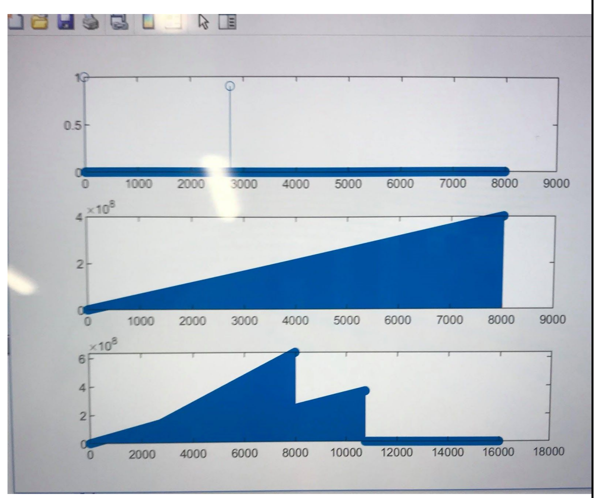


6.

Solution:

```
[hoa,hoan]= conv_m(he,n,her,dn);
figure(2)
subplot(3,1,1);
stem(n,he);
subplot(3,1,2);
stem(dn,her);
subplot(3,1,3);
stem(hoan,hoa);
```

Results:



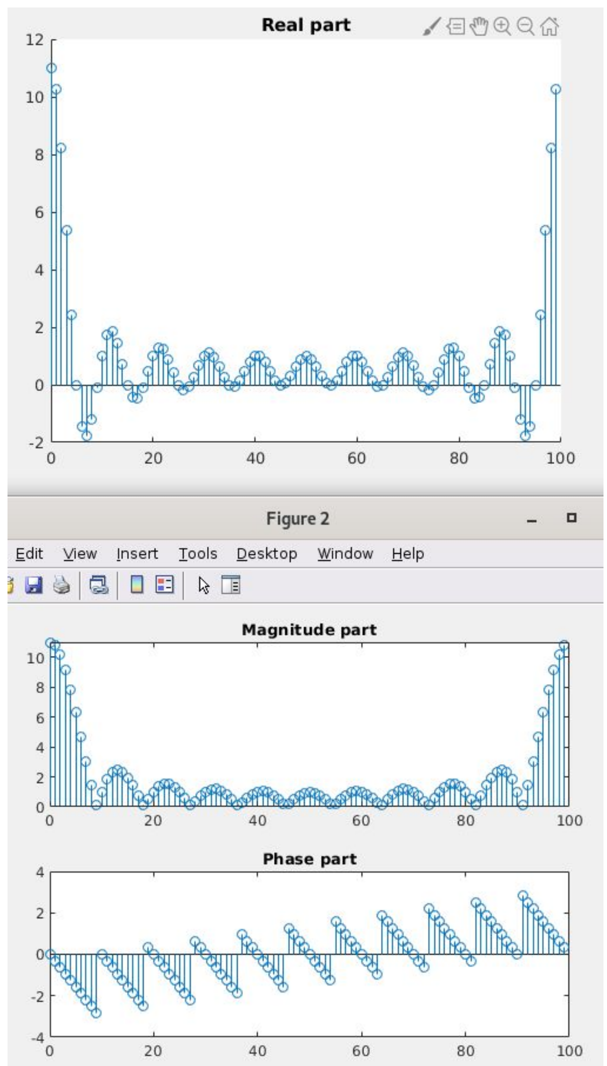
Encountered Problems*: This section was really hard for us because we have never done echo cancellation which made it hard and also we did not know what was right and wrong. We knew how to plot but couldn't verify weather it was right as we were lacking knowledge and skills in this certain topic. With the help of google and learning from the UC Berkeley website we were able to learn while implementing this in matlab.*

Part 4

Problem Description: For this part we're using the DTFT equation to find the h while plotting the results. The plot has the magnitude and phase of a complex quantity. Then we used the DTFT equation to be more easily accomplished by using the matrix capabilities of Matlab. We also repeated that for the designed impulse response sequence in terms of delays in a certain amount of samples.

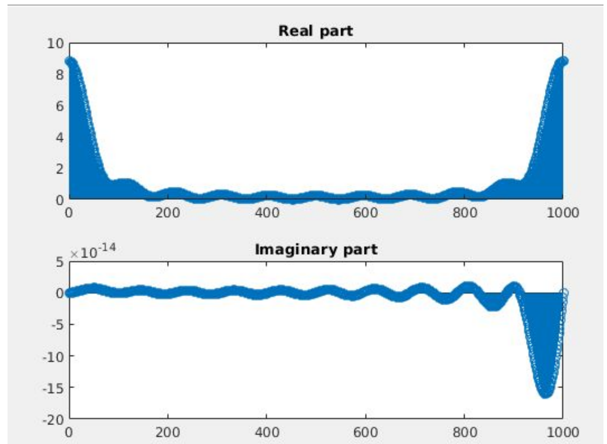
1.

```
hold on;
% The h(n) function
h = @(n) u(n) - u(n-11);

xaxis = [];
yaxis = [];
yaxis_phase = [];
yaxis_mag = [];

N = 100;
M = 100;
% For each k, from 0 to N-1
for k = 0:N-1
   xaxis = [xaxis k];
   X = 0;
   % For each n, from 0 to M-1
   for n = 0:M-1
       % Calcul the DTFT
       X = X + h(n)*exp((-j*2*pi*k*n)/N);
   end
   yaxis = [yaxis real(X)];
   yaxis_mag = [yaxis_mag abs(X)];
   yaxis_phase = [yaxis_phase angle(X)];
end

figure(1);
% Plot the results
stem(xaxis, yaxis);
title("Real part");

figure(2);
% Plot the magnitude
subplot(2, 1, 1);
stem(xaxis, yaxis_mag);
title("Magnitude part");

% Plot the phase
subplot(2, 1, 2);
stem(xaxis, yaxis_phase);
title("Phase part");

hold off;
```



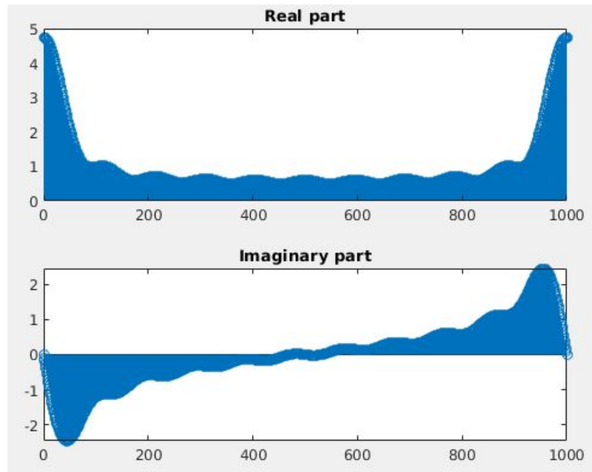
2.

```
N = 1000;

% The x(n) function
x = @(n) exp(-0.2* abs(n));

n1 = -10;
n2 = 10;

xaxis = [];
yaxis_real = [];
yaxis_img = [];

% The loop to calculate the dtft of x(n)
for k = [0:N]
   xaxis = [xaxis k];
   X = 0;
   for n = [n1:n2]
       X = X +
x(n)*(exp(-1j*2*pi/N)).^(n'*k);
   end
   % Save the real and imaginary results
   yaxis_real = [yaxis_real real(X)];
   yaxis_img = [yaxis_img imag(X)];
end

% Display the results
% The real part
subplot(2,1,1);
stem(xaxis, yaxis_real);
title("Real part");

% The imaginary part
subplot(2,1,2);
stem(xaxis, yaxis_img);
title("Imaginary part");
```



3.

```
N = 1000;

% The h(n) function
h = @(n) power(0.81, n) * (u(n) - u(n -
101));

n1 = -10;
n2 = 10;

xaxis = [];
yaxis_real = [];
yaxis_img = [];

% The loop to calculate the dtft of x(n)
for k = [0:N]
   xaxis = [xaxis k];
   X = 0;
   for n = [n1:n2]
       X = X +
h(n)*(exp(-1j*2*pi/N)).^(n'*k);
   end
   % Save the real and imaginary results
   yaxis_real = [yaxis_real real(X)];
   yaxis_img = [yaxis_img imag(X)];
end

% Display the results
% The real part
subplot(2,1,1);
stem(xaxis, yaxis_real);
title("Real part");

% The imaginary part
subplot(2,1,2);
stem(xaxis, yaxis_img);
title("Imaginary part");
```



***Encountered Problems*:** The main problem with this part was the comprehension of the exercice and the usage of the DTFT function.

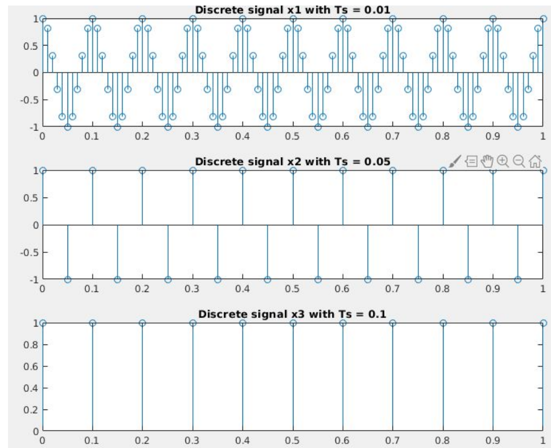
Part 5

Problem Description: The sampling and reconstruction for each of the T's resulting in xk(n). Then we reconstructed the analog signal from the samples using the zero order hold interpolation as well as the first order hold and the cubic spline. The sinc interpolation was constructed by the analog signal given from each of the samples.

1.

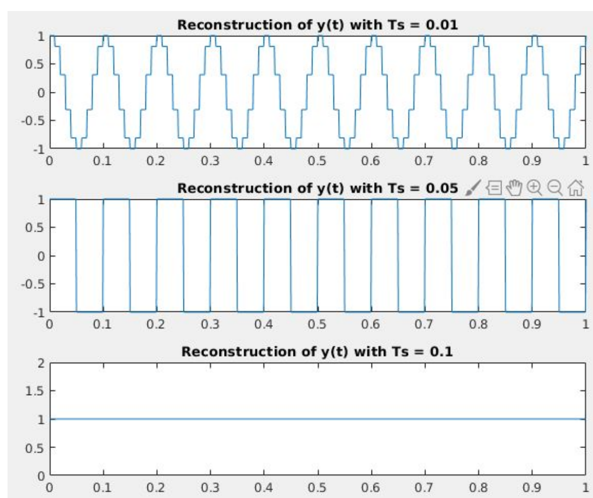
a)

```
x1
plotIndex = 1;
for Ts = [0.01, 0.05, 0.1]
   subplot(3, 1, plotIndex);

   xaxis = [];
   yaxis = [];
   for n = (0:Ts:1)
       xaxis = [xaxis, n];
       yaxis = [yaxis, cos(20 * pi *
n)];
   end
   % Display the values
   stem(xaxis, yaxis);
   % Add a title
   title("Discrete signal x" + plotIndex
+ " with Ts = "+ Ts);
   plotIndex = plotIndex + 1;
end
```



b)

```
% For each interval
plotIndex = 1;
for Ts = [0.01, 0.05, 0.1]
   %% Initialization %%

   % Setting interval of t
   tMax = 1;
   tStep = 0.001;
   tMin = 0;

   % Values of t
   tValues = (tMin:tStep:tMax);

   % Calculating the number of samples
   nbSamples = 1 / Ts;

   % Creating the impulse response
   impResponse = ones(1, Ts / tStep);

   % Creating the analog signal
   x = @(t)(cos(20 * pi * t));
```

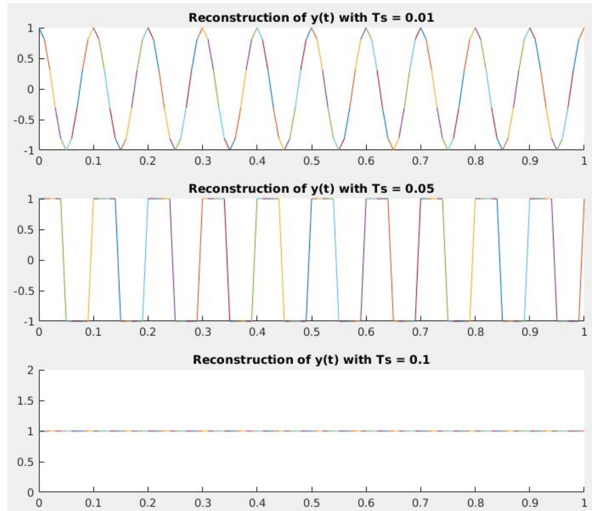


<pre> %% /Initialization %% %% Samples %% % Initialize the samples at length of tValues samples = zeros(1, length(tValues)); % An index used to create samples from the analog signal sampleIndex = 0; for n = 1:Ts / tStep:length(tValues) % Save a new sample samples(n) = x(sampleIndex * Ts); % Increase the index sampleIndex = sampleIndex + 1; end %% /Samples %% %% Treatment %% % Making a convolution between the impulse signal and the samples rSignal = conv(impResponse, samples); % Resize the reconstructed signal to the size of tValues rSignal = rSignal(1:length(tValues)); %% /Treatment %% % Plot the result subplot(3, 1, plotIndex); plot(tValues, rSignal); title("Reconstruction of y(t) with Ts = "+ Ts) plotIndex = plotIndex + 1; end</pre>	

The frequency is 10/1 = 10Hz

c)

```
% For each interval
plotIndex = 1;
for Ts = [0.01, 0.05, 0.1]
   %% Initialization %%

   % Setting interval of t
   tMax = 1;
   tStep = 0.001;
   tMin = 0;

   % Values of t
   tValues = (tMin:tStep:tMax);

   % Calculating the number of samples
   nbSamples = 1 / Ts;

   % Creating the impulse response
   impResponse = ones(1, Ts /
signalStep);

   % Creating the analog signal
   x = @(t)(cos(20 * pi * t));

   %% /Initialization %%

   %% Samples %%

   % Initialize the samples at length of
tValues
   samples = zeros(1, length(tValues));

   % An index used to create samples from
the analog signal
   sampleIndex = 0;

   for n = 1:Ts /
signalStep:length(tValues)
       % Save a new sample
       samples(n) = x(sampleIndex * Ts);
       % Increase the index
       sampleIndex = sampleIndex + 1;
   end


   %% /Samples %%


   %% Treatment %%

   % Making a convolution between the
impulse signal and the samples
   rSignal = conv(impResponse, samples);

   % Resize the reconstructed signal to
the size of tValues

   rSignal = rSignal(1:length(tValues));
```

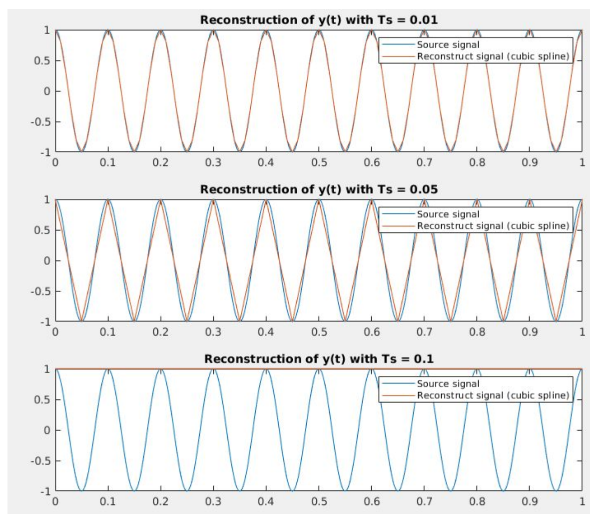


```
    %% /Treatment %%

    % Plot the result
    subplot(3, 1, plotIndex);
    title("Reconstruction of y(t) with Ts
= "+ Ts)
    hold on;
    for index = 1:10:length(tValues)
        if (index > 10)
            plot([tValues(index - 10)
tValues(index)], [rSignal(index - 10)
rSignal(index)]);
        end
    end
    hold off;
    plotIndex = plotIndex + 1;
end
```

The frequency is 10/1 = 10Hz

d)

```
% For each interval
plotIndex = 1;
signal = @(t) cos(20*pi*t);

% Setting interval of t
tMin = 0;
tMax = 1;
tStep = 0.001;
tInterval = (tMin:tStep:tMax);

for Ts = [0.01, 0.05, 0.1]

    % Values of t
    tValues = signal(tInterval);

    % samples
    sampleInterval = (tMin:Ts:tMax);
    samples = signal(sampleInterval);
    % Create spline values
    xSpline = interp1(tInterval, tValues,
sampleInterval,'spline');

    %% /Treatment %%

    % Plot the result
    subplot(3, 1, plotIndex);
    plot(tInterval, tValues,
sampleInterval, samples);
    title("Reconstruction of y(t) with Ts
= "+ Ts)
    legend('Source signal', 'Reconstruct
signal (cubic spline)');
```

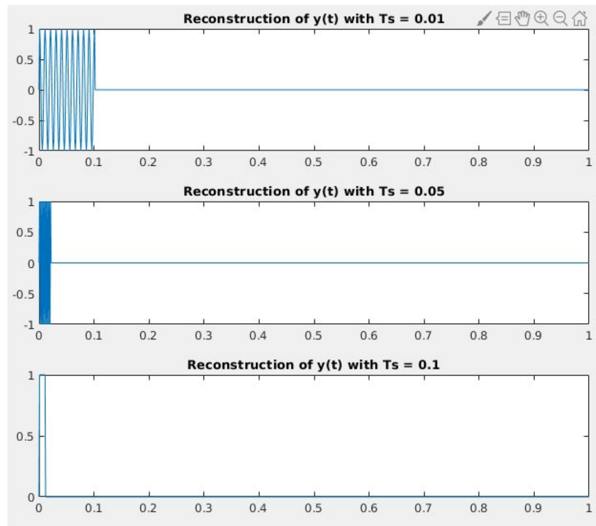


`plotIndex = plotIndex + 1;` `end`	

The frequency is 10/1 = 10Hz

e)

```
% For each interval
plotIndex = 1;
signal = @(t) cos(20*pi*t);

% Setting interval of t
tMin = 0;
tMax = 1;
tStep = 0.001;
tInterval = (tMin:tStep:tMax);

deltaT = 0.001;

for Ts = [0.01, 0.05, 0.1]

   % Values of t
   tValues = signal(tInterval);

   % samples
   sampleInterval = (tMin:Ts:tMax);
   samples = signal(sampleInterval);

   xaxis = [];
   yaxis = [];
   for t = tInterval
       xaxis = [xaxis t];
       y = 0;
       for n = 1:length(samples)
           y = y + samples(n) * sinc((t -
n*deltaT)/deltaT);
       end
       yaxis = [yaxis y];
   end
   %% /Treatment %%

   % Plot the result
   subplot(3, 1, plotIndex);
   plot(xaxis, yaxis);
   title("Reconstruction of y(t) with Ts
= "+ Ts)
   plotIndex = plotIndex + 1;
end
```



Encountered Problems: This part was very hard because of the multiple types of interpolation used. We add to search a lot about each one of them. It was difficult to know if the result we got was correct or not.