2021

I, Metkel Harshkumar (202001154), declare that the work that I am presenting it my own work.

I have not copied the work (Matlab code, results, etc.) that someone else has done.

Concepts understanding and insights I will be describing are my own. Wherever I have relied on an existing work that is not my own, I have provided a proper reference citation.

I make this pledge truthfully. I know that violation of this solemn pledge can carry grave consequences.

[illegible]

Lab Report-Lab 9

CT111

METKEL HARSHKUMAR ANANDBHAI

Name: Metkel Harshkuamr Anandbhai;

Student id: 202001154;

Course: CT111;

Lab no: 9;

Lab group: 3.

Practice part 0:

Here we experiment with a **powerful and generalized way of generating a type of (complex-valued) orthonormal matrices -** the DFT matrices of size N samples, where N is an arbitrary integer.

```
clearvars;


N = 64;


n = 0:N-1; % time variable (discrete) can be thought of no of columns
of matrix
k = (0:N-1)/N; % frequency variable (discretized), varies from 0 to 1 -
(1/N)


n = n'; % turn the vector of time sample index to a column vector


G = exp(1i*2*pi*n*k); % N x N Matrix of Discrete Fourier Transform
% G = cos(2*pi*n*k)+1i*sin(2*pi*n*k);
% for any particular column the value of k will be same, the behaviour
will
% be managed by n (frequecny value only)
column_sel = 5;
% n = 1, will produce two cycles, means two frequecy


plot(n,[real(G(:,column_sel)) imag(G(:,column_sel))],'linewidth',2);
xlabel('Discretized Time Sample n');
ylabel('Amplitude');
title(sprintf('Complex Exponential Signal on the row %d of the DFT
matrix',column_sel));
legend('Real','Imaginary','location','best'); grid;
xlim([0 N-1])
```

Here we n represents the frequency. Which varies from 0 to N-1.

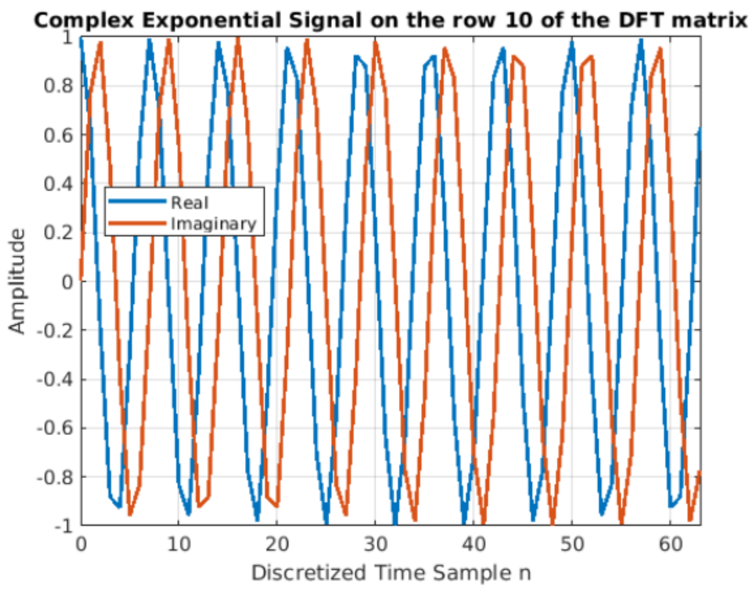
for any particular column of **G**$_{\text{matrix}}$ the value of k will be same, the behaviour will be managed by n (frequecny value only)

Observation:

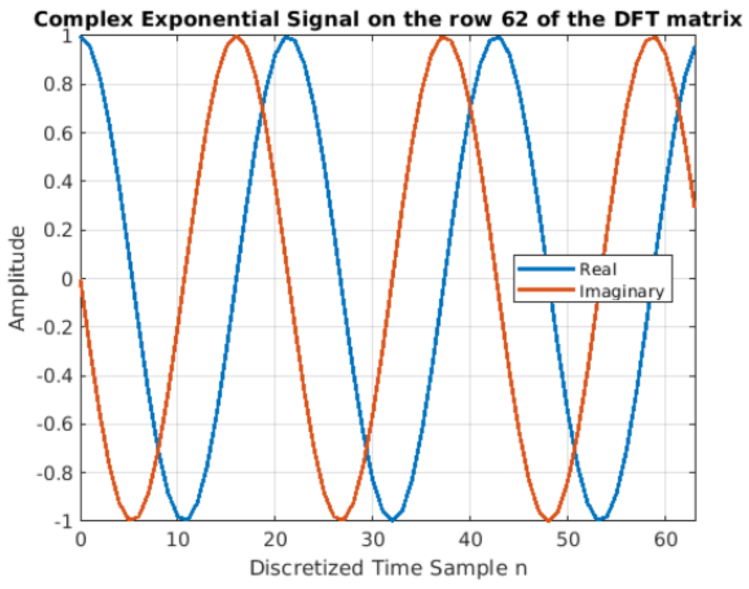
As we increase the value of n we observe the plot becomes more and more dense. Which means that no of cycles increases (as expected because n represents the frequency with which sin and cosine of $\mathbf{G}_{\text{matrix}}$ changes.

Also we observe that the frequency will keep on increasing until we reach n = 32. After that no of cycles again decreases this gives the analogy that the $\mathbf{G}_{\text{matrix}}$ is symmetric.

Plot for n = 10:

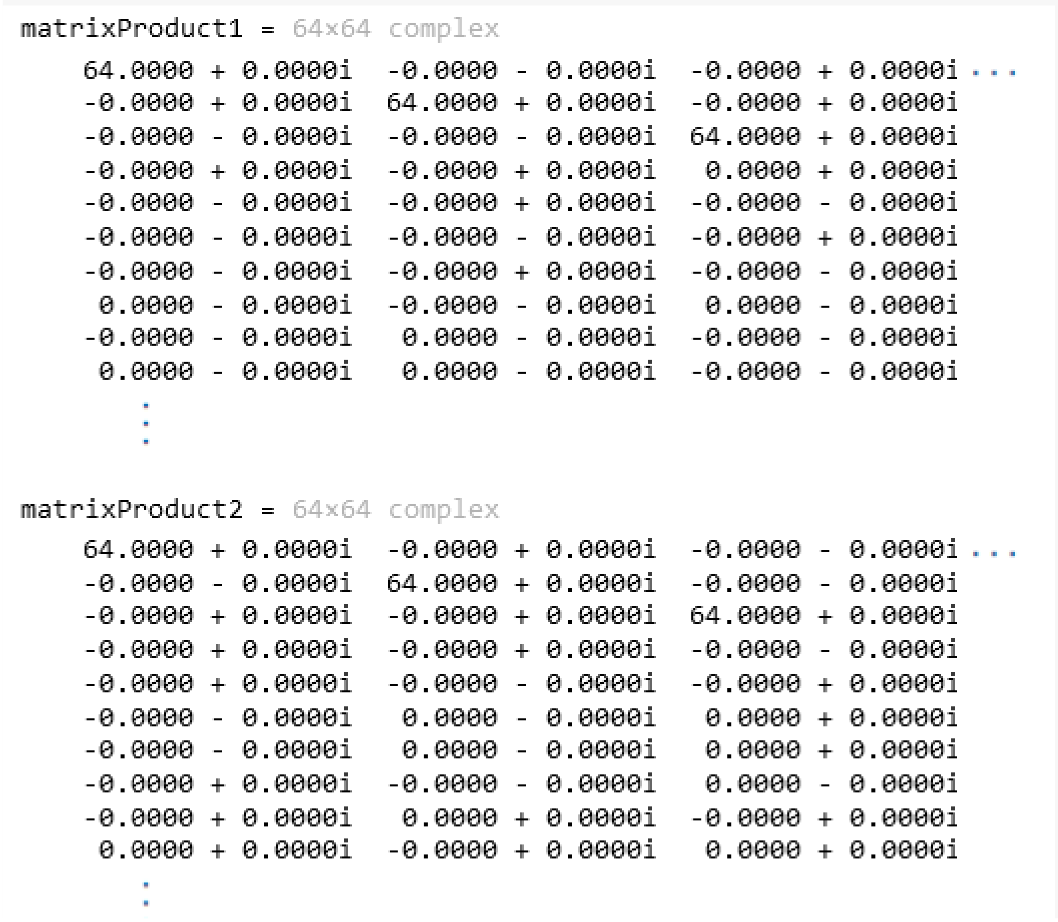


Plot for n = 62 (will have only three cycles)



Here we **check the orthonormality of the generated matrix.**

```
matrixProduct1 = G'*G


matrixProduct2 = G*G'
```



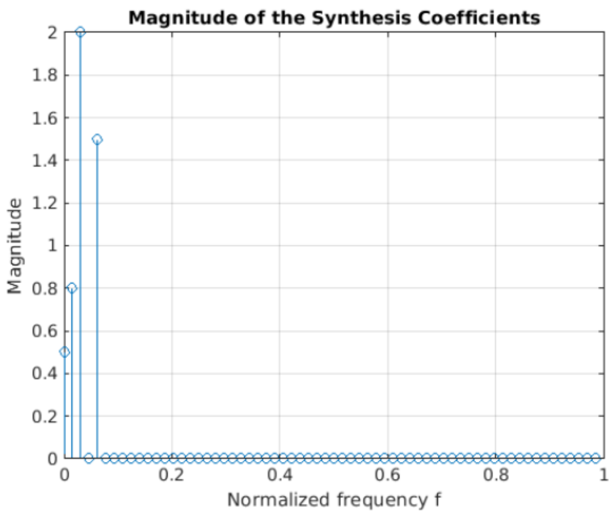
```
matrixProduct1 = 64x64 complex

    64.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i ...
    -0.0000 + 0.0000i  64.0000 + 0.0000i  -0.0000 + 0.0000i
    -0.0000 - 0.0000i  -0.0000 - 0.0000i  64.0000 + 0.0000i
    -0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i
    -0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i
    -0.0000 - 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i
    -0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i
     0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i
    -0.0000 - 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i
     0.0000 - 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i
        ⋮

matrixProduct2 = 64x64 complex

    64.0000 + 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i ...
    -0.0000 - 0.0000i  64.0000 + 0.0000i  -0.0000 - 0.0000i
    -0.0000 + 0.0000i  -0.0000 + 0.0000i  64.0000 + 0.0000i
    -0.0000 + 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i
    -0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i
    -0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i
    -0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i
    -0.0000 + 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i
    -0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i
     0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i
        ⋮
```

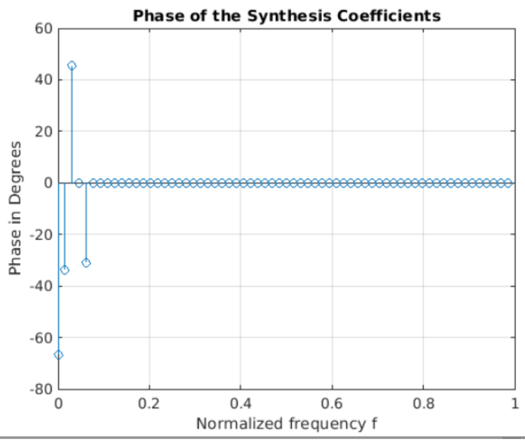
In either case we get the identity matrix this confirms the **orthonormality of the generated matrix.**

Now we synthesize a time-domain signal by specifying the weights (which can be complex-valued) of the frequencies that we would like to "mix-up" in the synthesized "cocktail".

```
S_f = zeros(N,1); % Initialize the frequency-domain "weights" with all
zeros
column_sel = [1 2 3 5];
S_f(column_sel) = [0.5*exp(1i*rand*2*pi) 0.8*exp(1i*rand*2*pi)
2*exp(1i*rand*2*pi)  1.5*exp(1i*rand*2*pi) ];
stem(k,abs(S_f)); % plot the magnitude of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Magnitude'); title('Magnitude
of the Synthesis Coefficients'); grid
stem(k,angle(S_f)*180/pi); % plot the phase of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the Synthesis Coefficients'); grid
```

here we are doing synthesis by performing linear superposition of some of four columns mentioned in column_sel. And then what we observe is that output will have that weights only at the columns indexes and rest will be zero.

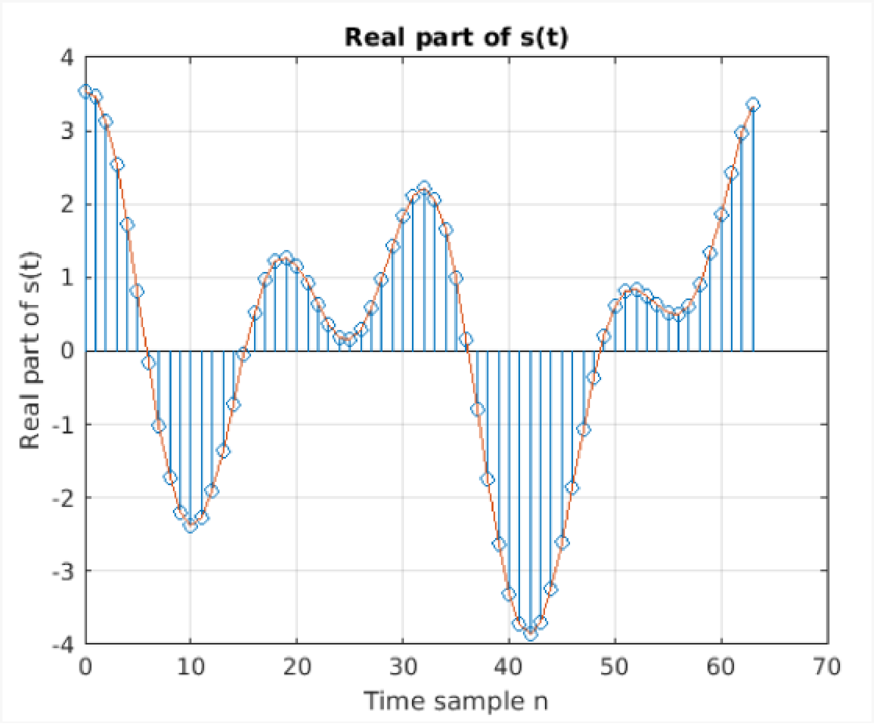


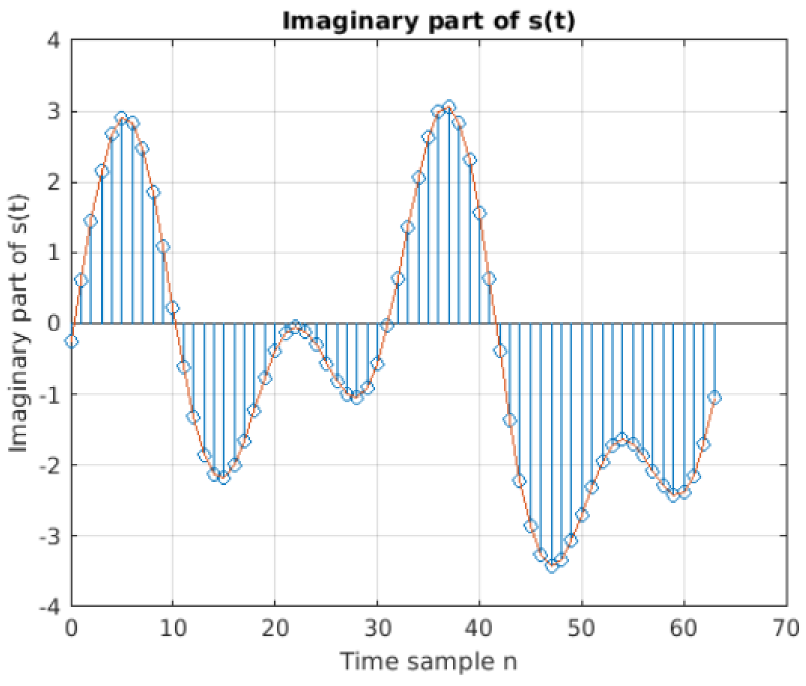


Here the second image shows the random generated the phase for the s(t).

The synthesis of the time-domain signal follows given the above frequency spectrum:

```
s_t = G*S_f; % This is the Inverse Fourier Transform
figure; stem(n,real(s_t)); hold on; plot(n,real(s_t))
xlabel('Time sample n'); ylabel('Real part of s(t)'); title('Real part
of s(t)'); grid


figure; stem(n,imag(s_t)); hold on; plot(n,imag(s_t))
xlabel('Time sample n'); ylabel('Imaginary part of s(t)');
title('Imaginary part of s(t)'); grid
```

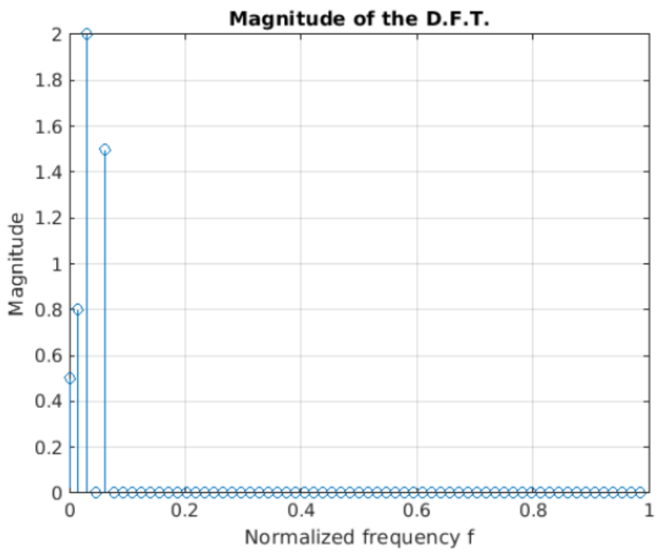




Here having the frequency based signal performing **Inverse-Fourier Transform** we get **Time-domain signal.**

The synthesis of the time-domain signal follows given the above frequency spectrum:

```
s_t = G*S_f; % This is the Inverse Fourier Transform
figure; stem(n,real(s_t)); hold on; plot(n,real(s_t))
xlabel('Time sample n'); ylabel('Real part of s(t)'); title('Real part
of s(t)'); grid


figure; stem(n,imag(s_t)); hold on; plot(n,imag(s_t))
xlabel('Time sample n'); ylabel('Imaginary part of s(t)');
title('Imaginary part of s(t)'); grid
```



here on performing **Analysis (Fourier Transform)** we are able to again generate the s(t). This shows the application of Fourier Transform in Frequency upconversion and Downconversion. As the signal is never affected but we become able to transmit the signal in RF range.

Conclusionn:

If we change, frequency variable k as then it won't affect analysis and synthesis portion. The only apparent difference would be in dimension of $\mathbf{G}_{\text{matrix}}$**.** This is apparent by observing following thing. In first case as we decreases no of samples the graph would now changes in smaller range, like for 1st case the cycles increases form 0 to 16 and then start decreasing. Similarly in second case it first increases from 0 to 63 and then decreases afterwards.

Practice part 1:

Here we **analyze** different types of signals encountered in practice. **The purpose is to observe the spectral characteristics of these signals**, i.e., which frequencies are dominant and which frequencies are absent (or, more accurately, the *complex exponentials at* which frequencies are present)

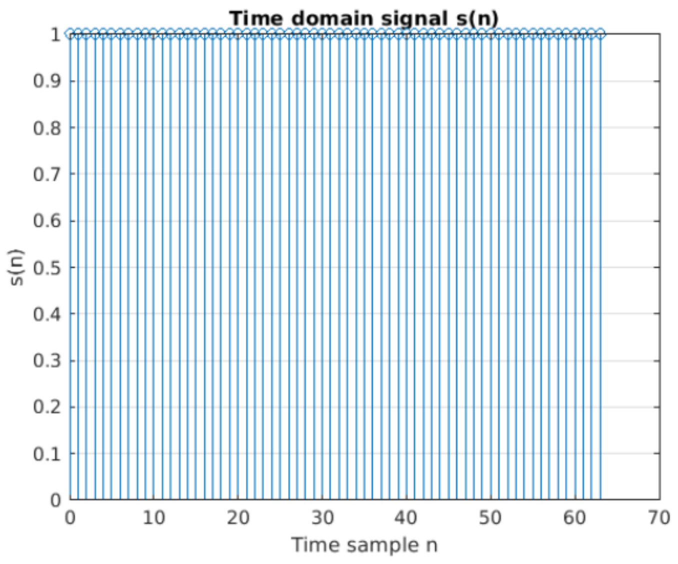
```
clearvars;


N = 64;


n = 0:N-1; % time variable (discrete)
% frequency variable (discretized), varies from -N/2 to +N/2-(1/N)
k = (-N/2:N/2-1)/N;
% note the above is different from Matlab's fft function, which uses k as 0 to
% 1-(1/N). However, Matlab has fftshift function which converts the fft output
% to the form in which k varies from -N/2 to +N/2-(1/N)


n = n'; % turn the vector of time sample index to a column vector


G = exp(1i*2*pi*n*k); % N x N Matrix of Discrete Fourier Transform
% G = cos(2*pi*n*k)+1i*sin(2*pi*n*k);
```

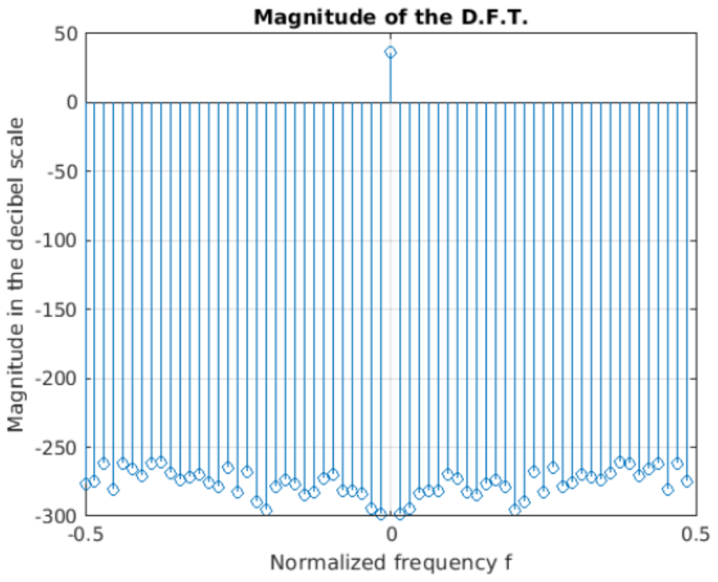
Generate a **DC time domain signal** with length N samples

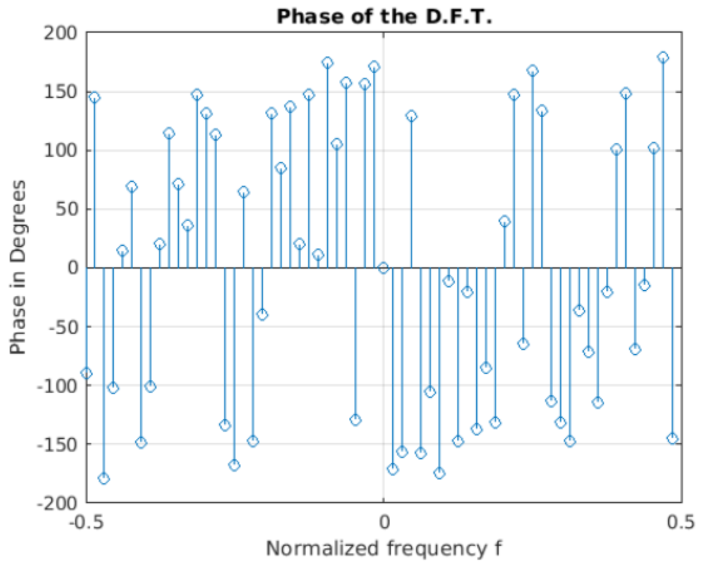
```
s_n = ones(N,1); % This is the time-domain DC signal
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal s(n)');
grid
```

The analysis of the time-domain signal to see which frequency components are present

```
S_f_DFT = G'*s_n; % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in the decibel scale');
title('Magnitude of the D.F.T.'); grid


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees'); title('Phase of
the D.F.T.'); grid
```


Time domain signal s(n)
s(n)
Time sample n
0
10
20
30
40
50
60
70
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Magnitude of the D.F.T.
Magnitude in the decibel scale
Normalized frequency f
50
0
-50
-100
-150
-200
-250
-300
-0.5
0
0.5


Phase of the D.F.T.
Phase in Degrees
Normalized frequency f
200
150
100
50
0
-50
-100
-150
-200
-0.5
0
0.5

Observation:

Here this signal is DC. Which means that the frequency of signal is 0. And this is apparent also in above curves.

In plot showing amplitude of the analysed signal we see that there in only one peak raising at f= 0Hz. And rest all values are having large negative dB values.

However the plot shows large negative values which means that that value are very small. Means close to negative infinity.

While 3rd plot is simply the plot of phases angles. Which is generated randomly. But important point is that at f = 0. We get phase 0.

Generate a **triangular-shaped time domain signal** with length N samples

```
s_n = [linspace(0,1,N/2) linspace(1,0,N/2)]'; % triangular signal
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal s(n)');
grid
```

The analysis of the time-domain signal:

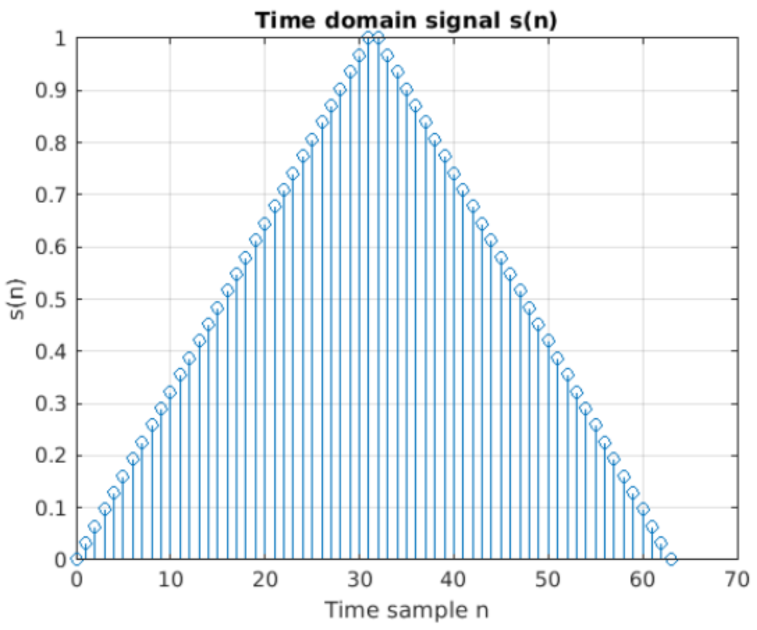
```
S_f_DFT = G'*s_n; % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 32])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees'); title('Phase of
the D.F.T.'); grid
```

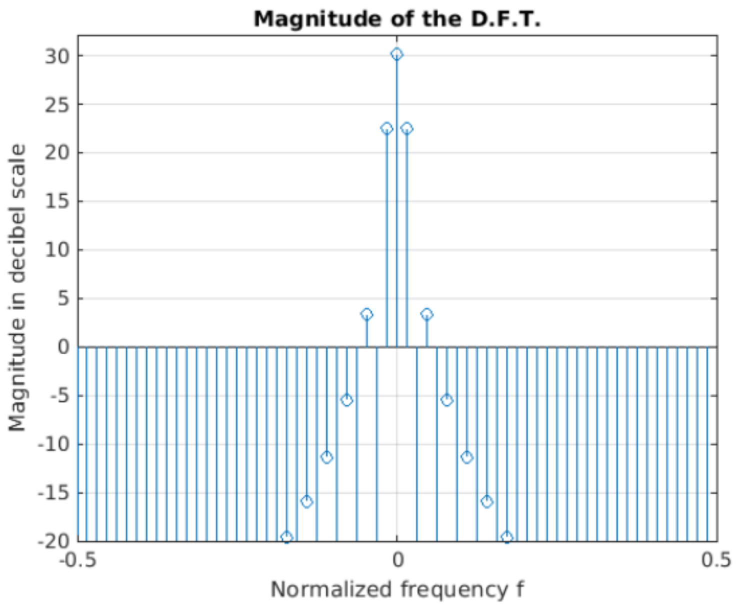
Now in triangular plot we will be having real value plots. And as the frequency varies different magnitudes are generated.

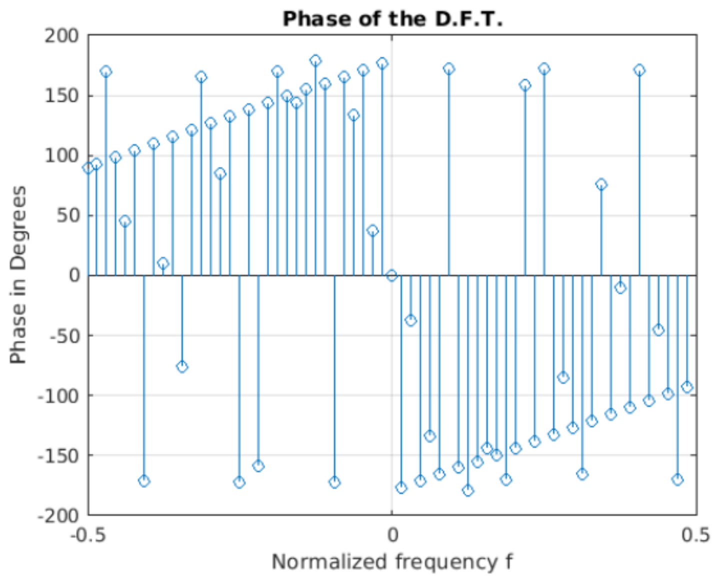
For the real valued signal, its Fourier Transform is Complex Conjugate Symmetry on frequency axis.

This is apparent in magnitude-frequency plot. We see that both right and left side of zero are exactly mirror images of each other. Magnitude is symmetric.

While if we see phase-frequency plot. We see that, still there is negative-symmetry. But not same as that in case of magnitude. Here we have symmetry around negative axis. Take the positive frequency axis multiply it by -1 then it will be plot of negative frequency axis.


Time domain signal s(n)
s(n)
Time sample n
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70


Magnitude of the D.F.T.
Magnitude in decibel scale
Normalized frequency f
30
25
20
15
10
5
0
-5
-10
-15
-20
-0.5
0
0.5


Phase of the D.F.T.
Phase in Degrees
Normalized frequency f
200
150
100
50
0
-50
-100
-150
-200
-0.5
0
0.5

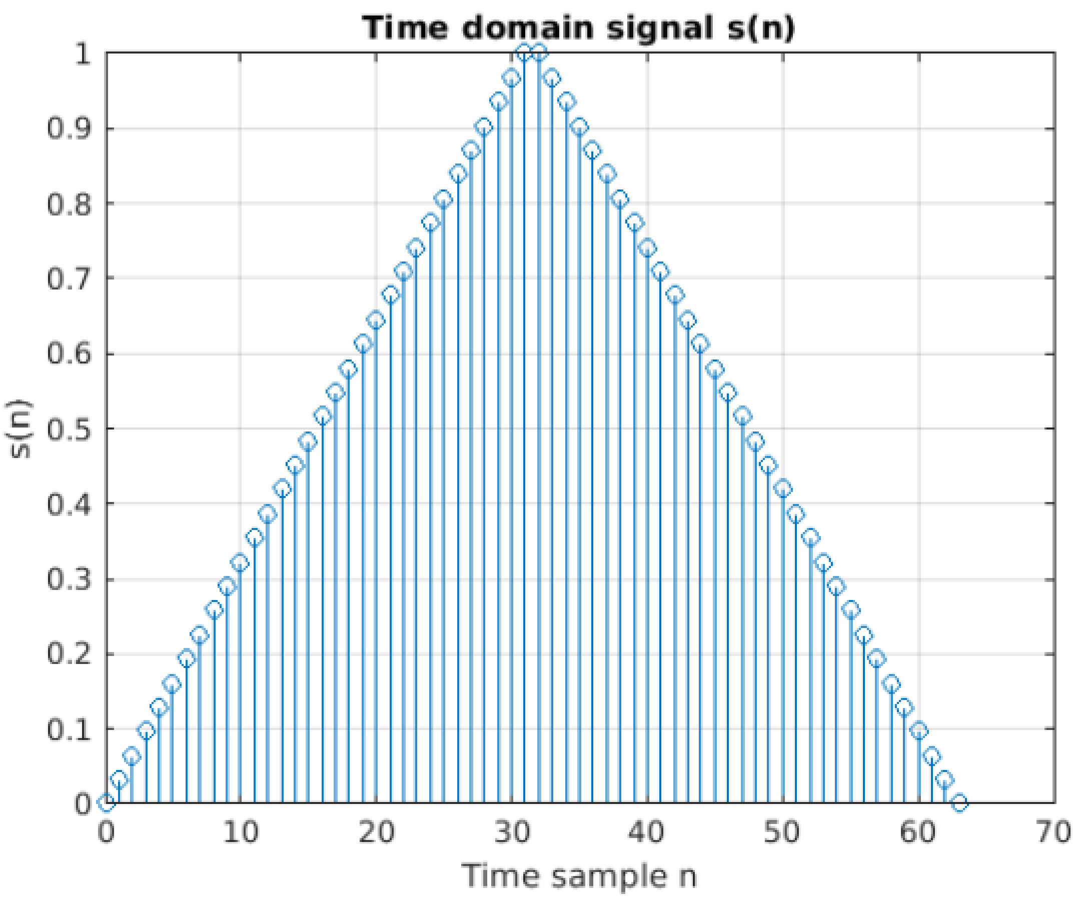
Generate a **triangular-shaped time domain signal** with length N samples

```
s_n = [linspace(0,1,N/2) linspace(1,0,N/2)]'; % triangular signal
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid
```

```
The analysis of the time-domain signal:
```

```
S_f_DFT = G'*s_n; % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 32])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the D.F.T.'); grid
```

now when we perform inverse F.T. we again get the Triangular plot same as generated previously. This shows the symmetry of Fourier Transform.



Generate a wavy pulse signal with length N samples - this is called **the sinc pulse**

```
s_n = sinc(linspace(-5,5,N)); s_n = s_n'; % SINC pulse
figure; stem(n-N/2,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid
```

```
The analysis of the time-domain signal:
```

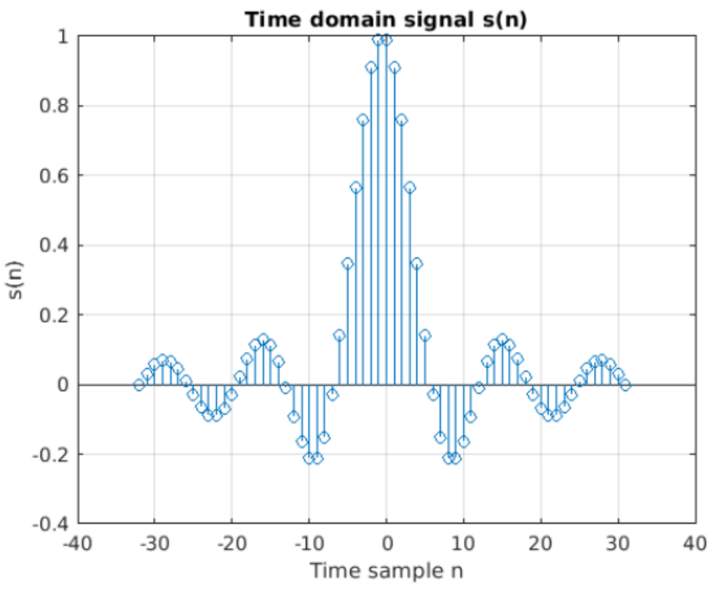
```
S_f_DFT = G'*s_n; % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 20])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the D.F.T.'); grid
```

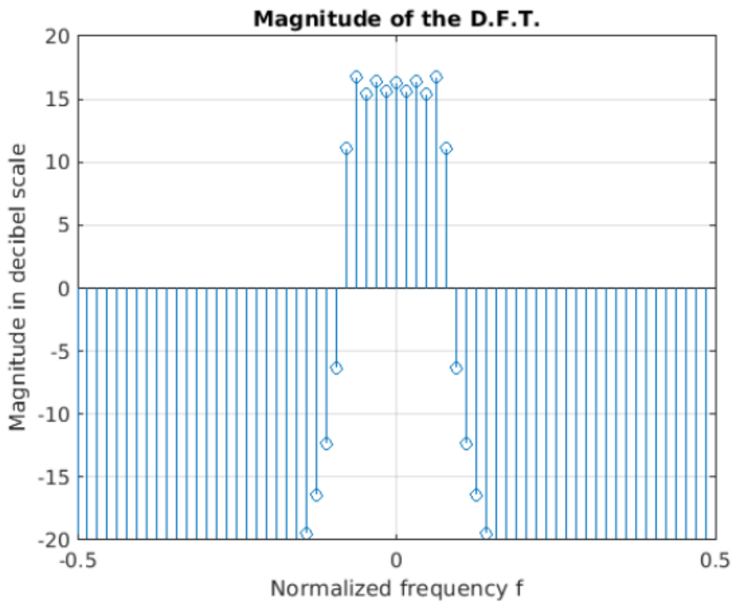
Here sinc is wavy curve whose wavy part kind of tapers down as it moves away from origin. But this never becomes exactly and moves to –infinity to + infinity.

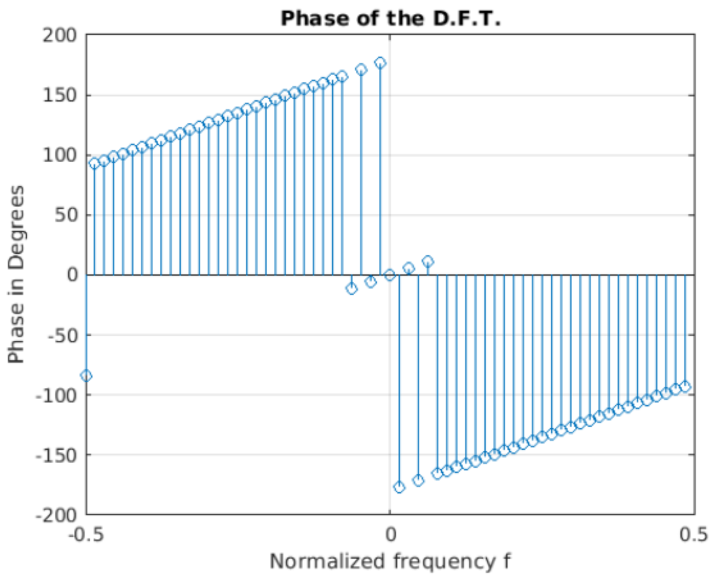
Here also we observe the mirror image symmetry in magnitude of analyses signal. And negative axis symmetry in phase plot-frequency plot. As expected.

Here we observe that magnitude plot looks something like rectangular pulse.

So now if we apply concept of duality, to it. Take this rectangular plot as the time domain signal and perform analysis of that we would get sinc function same as we generated earlier but in frequency domain.


Time domain signal s(n)
s(n)
Time sample n
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-40
-30
-20
-10
0
10
20
30
40


Magnitude of the D.F.T.
Magnitude in decibel scale
Normalized frequency f
20
15
10
5
0
-5
-10
-15
-20
-0.5
0
0.5


Phase of the D.F.T.
Phase in Degrees
Normalized frequency f
200
150
100
50
0
-50
-100
-150
-200
-0.5
0
0.5

Repeat the above analysis using Matlab's fft function. As mentioned above, you will need to use fftshift function to obtain the same analysis result as obtained using the G matrix above

```
s_n = ones(N,1); % This is the time-domain DC signal
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid
```

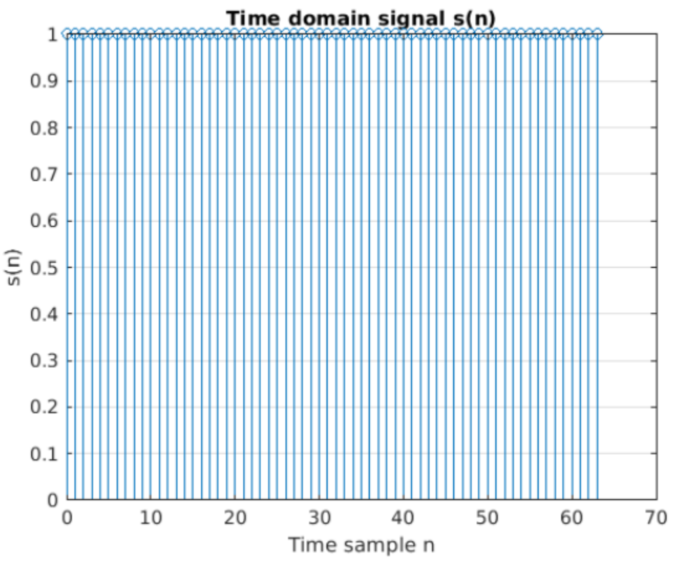
Analysis using matlab's fft function

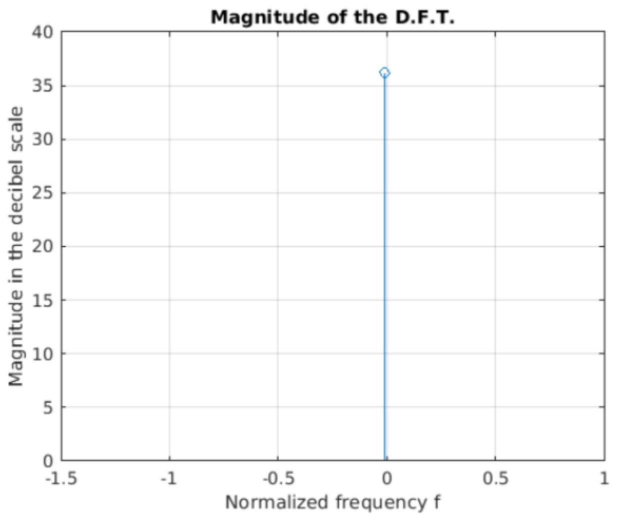
```
S_f_DFT = fftshift(fft(s_n)); % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in the decibel
scale');
title('Magnitude of the D.F.T.'); grid


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the D.F.T.'); grid
```

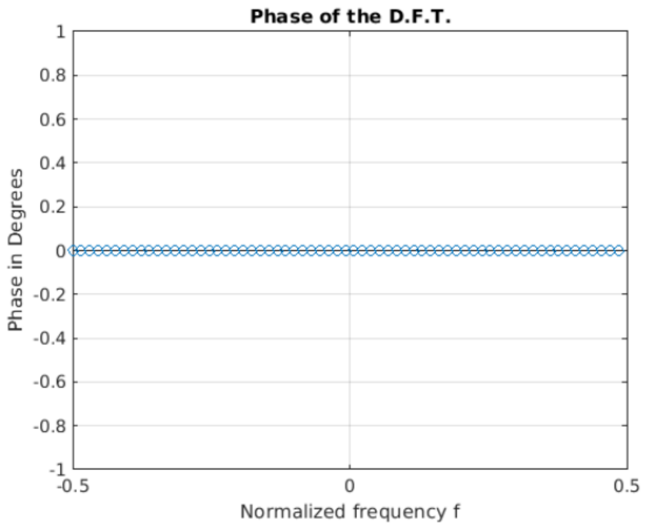
Here simply we need to use the fft function of Matlab, this will perform the Fourier Transform for us. Rest all the code remains same.

This time one interesting point to notice is that, in F.T plot we only have the signals in frequency spectrum of that signal. Because Matlab's itself removes all the signal all the signals which are technically going to negative infinity on Fourier Transform.

And also as we are not using any random frequency the frequency plot is coming exactly zero. Point to note is, even this maintains the negative frequency symmetry.


Time domain signal s(n)
s(n)
Time sample n
0
10
20
30
40
50
60
70
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Magnitude of the D.F.T.
Magnitude in the decibel scale
Normalized frequency f
-1.5
-1
-0.5
0
0.5
1
0
5
10
15
20
25
30
35
40


Phase of the D.F.T.
Phase in Degrees
Normalized frequency f
-0.5
0
0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

Now for triangular pulse:

Generate a **triangular-shaped time domain signal** with length N samples

```
s_n = [linspace(0,1,N/2) linspace(1,0,N/2)]'; % triangular signal
figure; stem(n,s_n);
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid
```

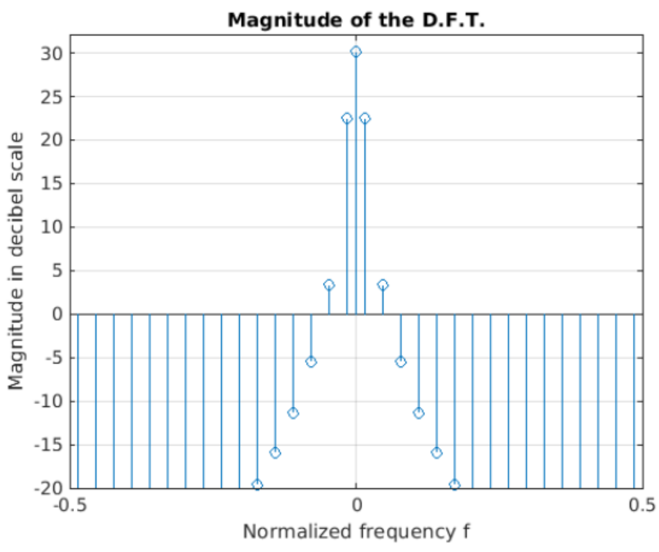
This time inorder to understand concept of duality I am treating this wave as in frequency domain

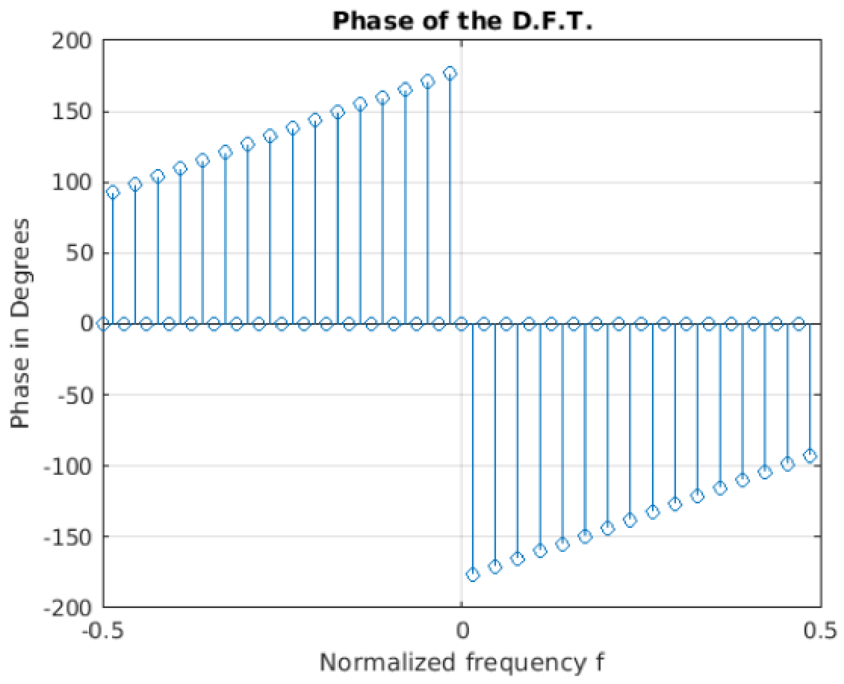
```
S_f_DFT = fftshift(fft(s_n)); % This is the Inverse-Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 32])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the D.F.T.'); grid
```

now performing analysis of S_F_DFT we get the trangular back

```
S_f_DT = ifft(S_f_DFT); %inverse f.t.
figure; stem(n, S_f_DT);
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal s(n)
generated back by Inverse Fourier Transform'); grid
```

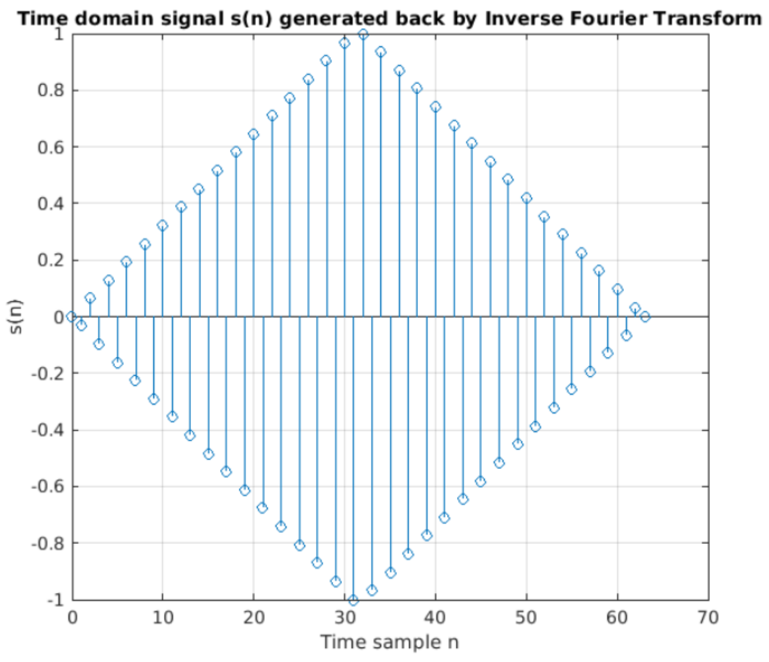
So plot obtained using Matlab's function are:





Below is the image of synthesis (Inverse Fourier Transform) for the above frequency domain signal:

Note here because of using fftshift we are getting the mirror image symmetry of curve.

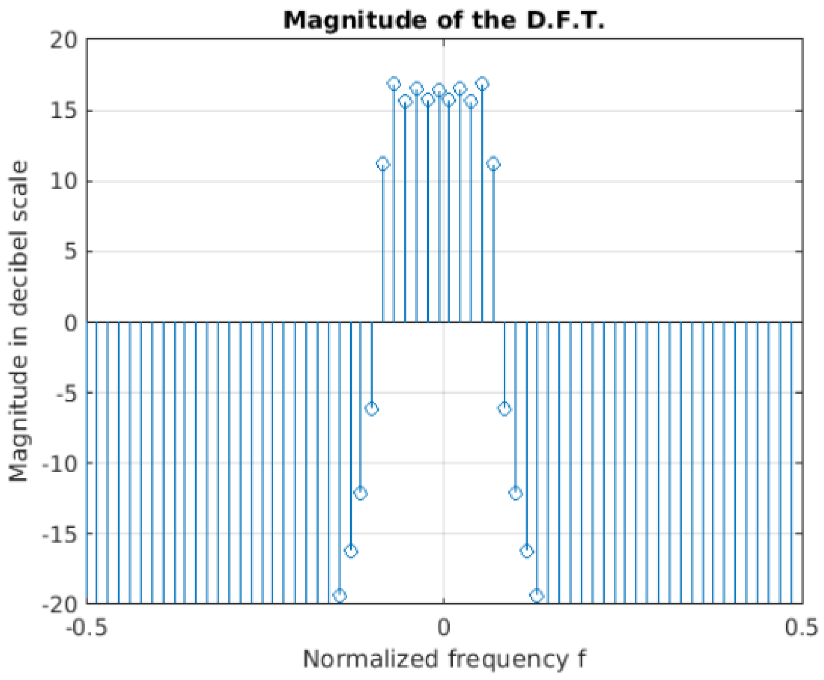


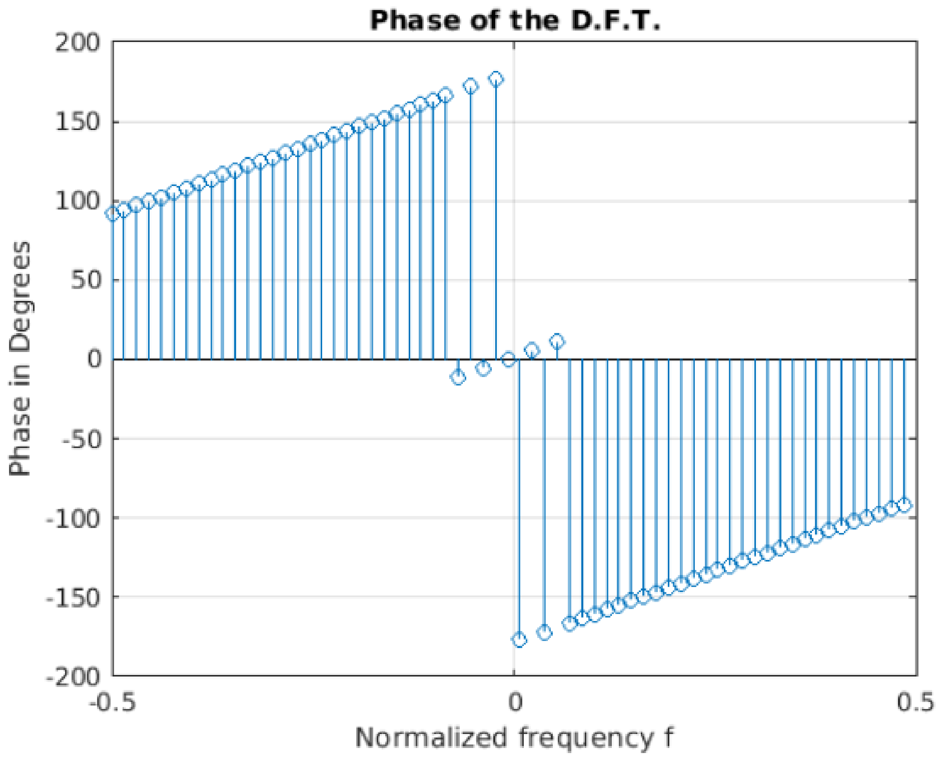
Generate a wavy pulse signal with length N samples - this is called **the sinc pulse**

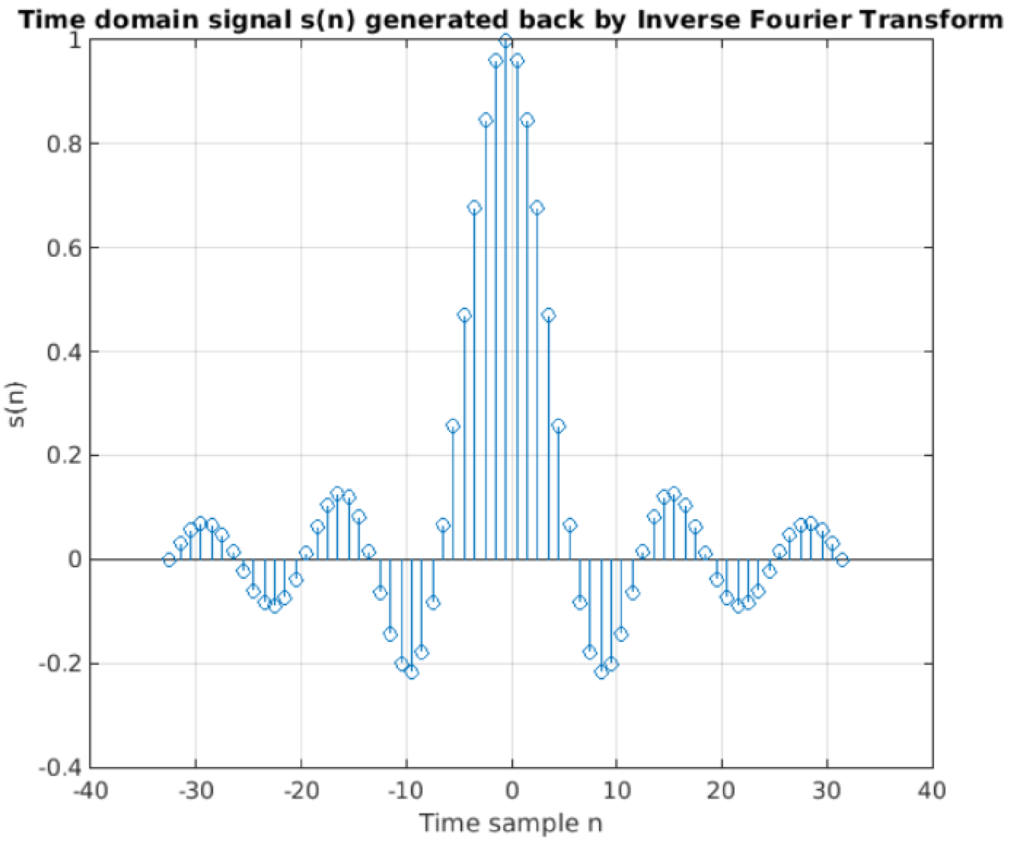
```
s_n = sinc(linspace(-5,5,N)); s_n = s_n'; % SINC pulse
figure; stem(n-N/2,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid


S_f_DFT = fftshift(fft(s_n)); % This is the Fourier Transform


stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 20])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of D.F.T.'); grid
```

now performing analysis of S_F_DFT we get the trangular back

```
S_f_DT = ifft(S_f_DFT); %inverse f.t.
figure; stem(n-N/2,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal s(n)
generated back by Inverse Fourier Transform'); grid
```




Phase of the D.F.T.
Phase in Degrees
Normalized frequency f
200
150
100
50
0
-50
-100
-150
-200
-0.5
0
0.5

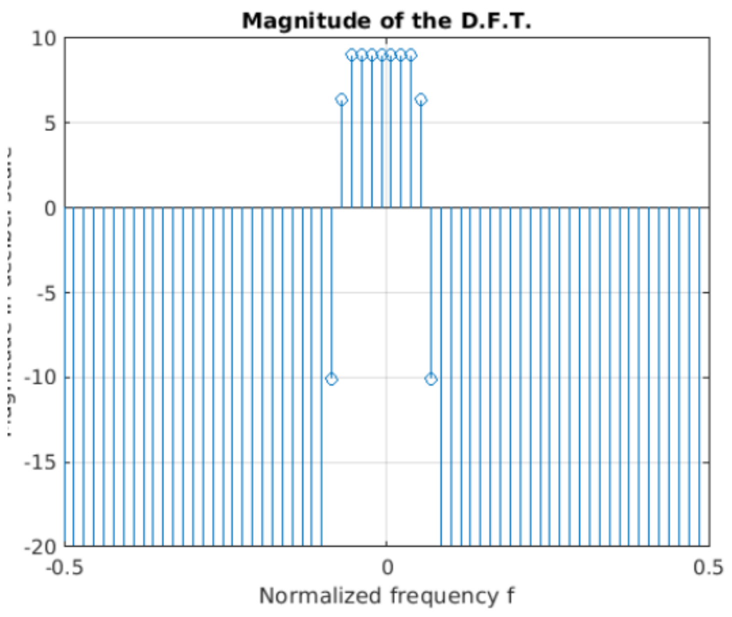

Time domain signal s(n) generated back by Inverse Fourier Transform
s(n)
Time sample n
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-40
-30
-20
-10
0
10
20
30
40

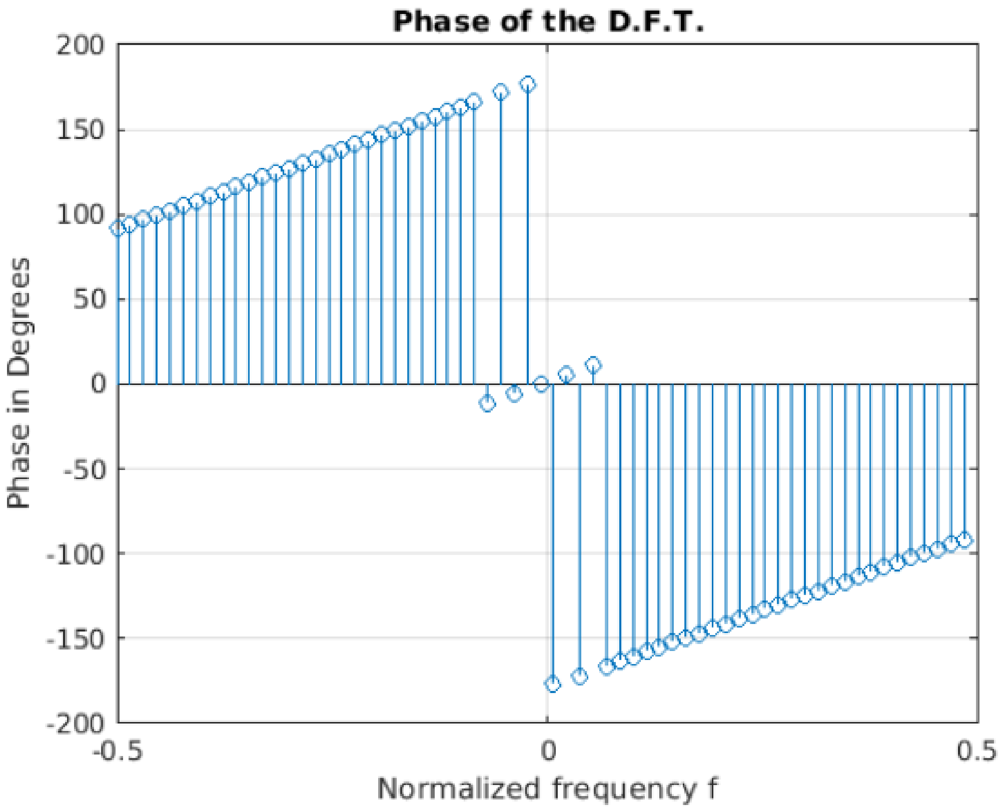
Generate a wavy pulse signal with length N samples - this is called **the (square-root) raised-cosine pulse**

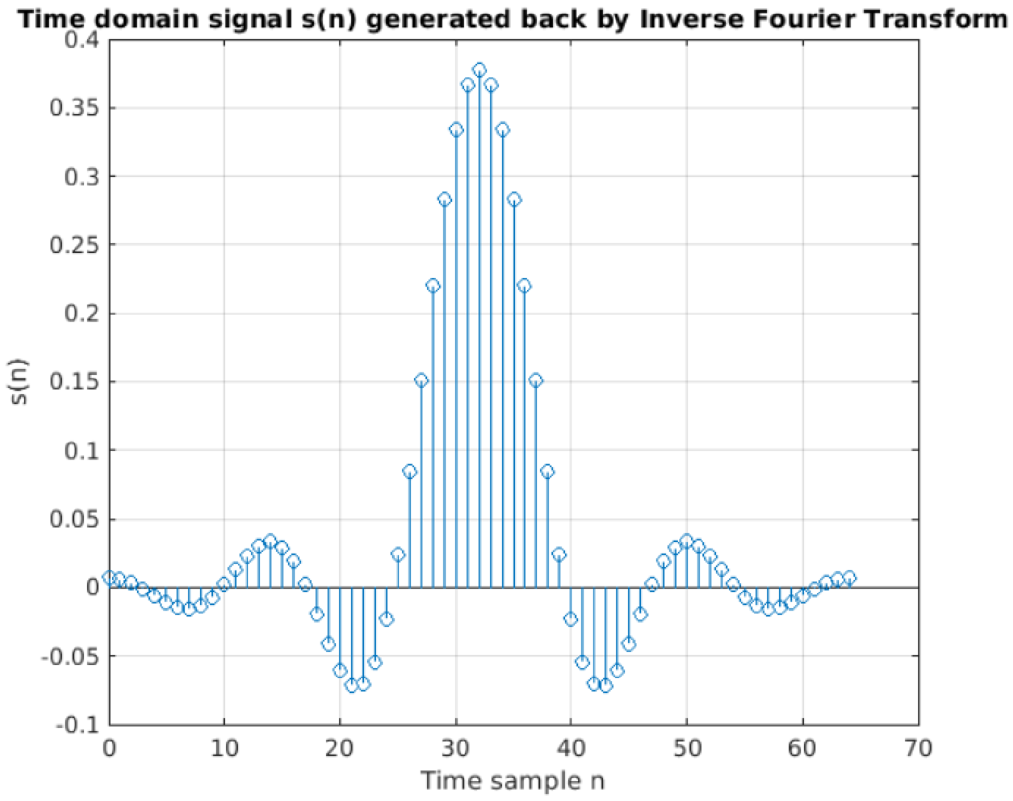
```
s_n = rcosdesign(0.25, 8, 8,'sqrt'); s_n = s_n'; % SRRC pulse
N = length(s_n); n = 0:N-1; k = (-N/2:N/2-1)/N; n = n';
G = exp(1i*2*pi*n*k); % N x N Matrix of Discrete Fourier Transform
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal
s(n)'); grid
```

```
The analysis of the time-domain signal:
```

```
S_f_DFT = fftshift(fft(s_n)); % This is the Fourier Transform
stem(k,20*log10(abs(S_f_DFT))); % plot the magnitude of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in decibel scale');
title('Magnitude of the D.F.T.'); grid
ylim([-20 10])


stem(k,angle(S_f_DFT)*180/pi); % plot the phase of the Fourier
Transform
xlabel('Normalized frequency f'); ylabel('Phase in Degrees');
title('Phase of the D.F.T.'); grid


S_f_DT = ifft(S_f_DFT); %inverse f.t.
figure; stem(n,s_n)
xlabel('Time sample n'); ylabel('s(n)'); title('Time domain signal s(n)
generated back by Inverse Fourier Transform'); grid
```




Phase of the D.F.T.
Phase in Degrees
200
150
100
50
0
-50
-100
-150
-200
-0.5
0
0.5
Normalized frequency f


Time domain signal s(n) generated back by Inverse Fourier Transform
s(n)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
10
20
30
40
50
60
70
Time sample n

Practice part 2:

Here we experiment with **the convolution operation** and study its several **applications at the communication transmitter and receiver.**

Assuming that you have already experimented with Matlab's fft function, here we directly use it instead of defining the G matrix and using it to perform the Discrete Fourier Transform (DFT)

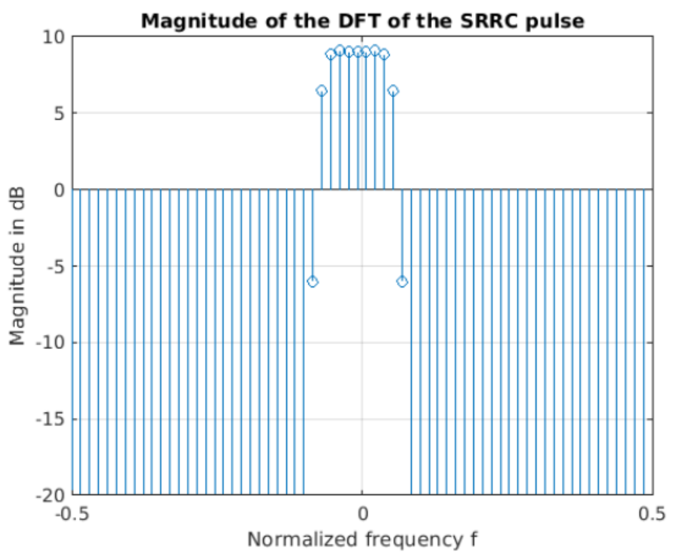
```
clearvars;


N = 65;
n = 0:N-1; % time variable (discrete)
k = (-N/2:N/2-1)/N; % frequency variable (discretized), varies from -N/2 to
+N/2-(1/N)
```

Step 1: Generate the square-root raised-cosine (SRRC) pulse

- We directly use an in-built Matlab function rcosdesign to generate this pulse
- This function requires several inputs: beta defines the shape of the wavy pulse, pulse_span defines the number of symbols over which the wavy pulse extends, and samples_per_symbol specifies the number of samples of this wavy pulse that are generated for each symbol duration

```
pulse_span = 8; % number of symbols
samples_per_symbol = 8; % number of samples per symbol
beta_srrc = 0.3;
h_n = rcosdesign(beta_srrc, pulse_span, samples_per_symbol,'sqrt'); h_n =
h_n'; % SRRC pulse
H_f_DFT = fftshift(fft(h_n)); % the Fourier Transform
stem(k,20*log10(abs(H_f_DFT))); % plot the magnitude of the Fourier Transform
xlabel('Normalized frequency f'); ylabel('Magnitude in dB');
title('Magnitude of the DFT of the SRRC pulse'); grid
ylim([-20 10])
```

Step 1: it is all about generating raised cosine function. And performing its Fourier Transform which will generate the rectangular pulse.



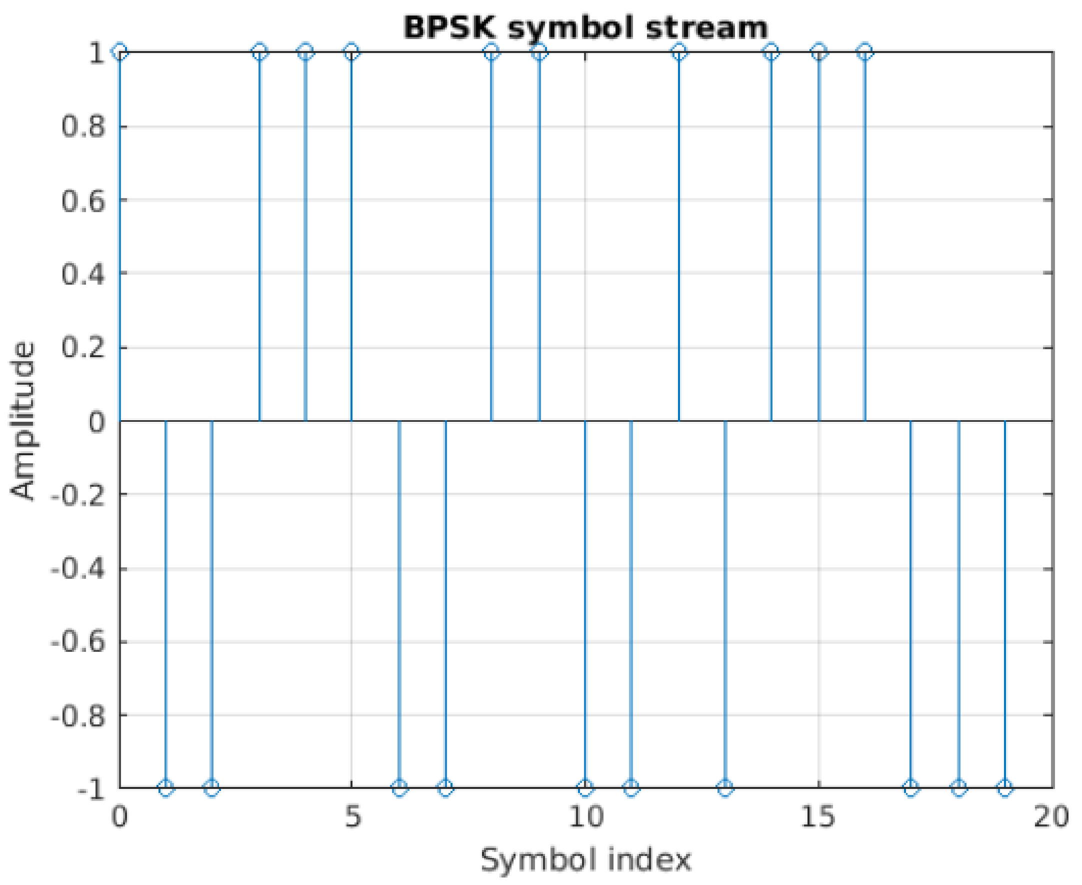
Step 2a Pulse Shaping: Generate BPSK symbol stream and observe its shape in the time domain and in the frequency domain

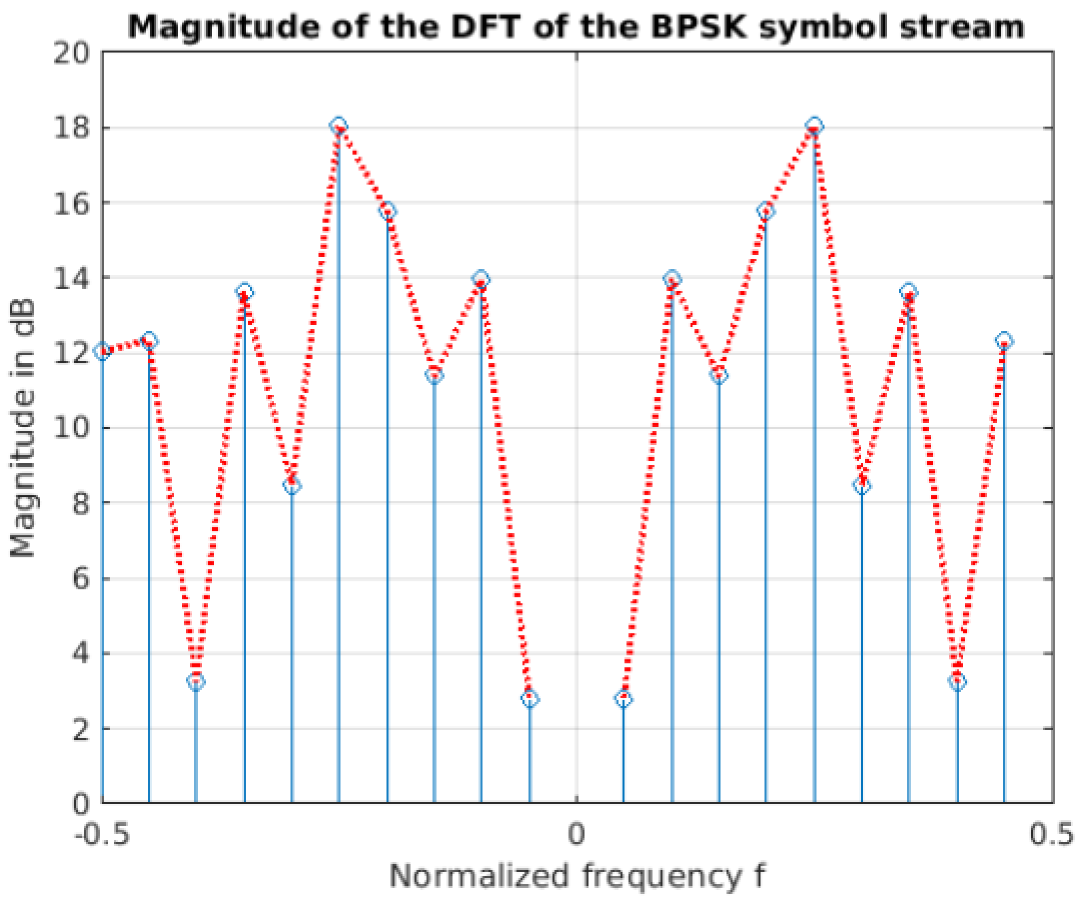
```
Nbpsk = 20;
b_n = randi([0 1],Nbpsk,1); % Generate Nbpsk bits out of a Bernouilli(q = 0.5)
source
s_n = 2*b_n - 1; % BPSK modulated symbol stream of length Nbpsk symbols


stem(0:Nbpsk-1,s_n); % plot the BPSK symbol stream
xlabel('Symbol index'); ylabel('Amplitude');
title('BPSK symbol stream'); grid


S_f = fftshift(fft(s_n)); % the Fourier Transform
kVar = (-Nbpsk:2:Nbpsk-1)/2/Nbpsk;
plotVariable = 20*log10(abs(S_f)); stem(kVar,plotVariable)
hold on; plot(kVar,plotVariable,'r:','linewidth',2); hold off
xlabel('Normalized frequency f'); ylabel('Magnitude in dB');
title({'Magnitude of the DFT of the BPSK symbol stream'}); grid
```

Step 2:

This step first produces the Nbpsk symbols and then perform pulse shaping of that pulse.

Which is essentially a Fourier Transform.


BPSK symbol stream
Amplitude
Symbol index
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
5
10
15
20

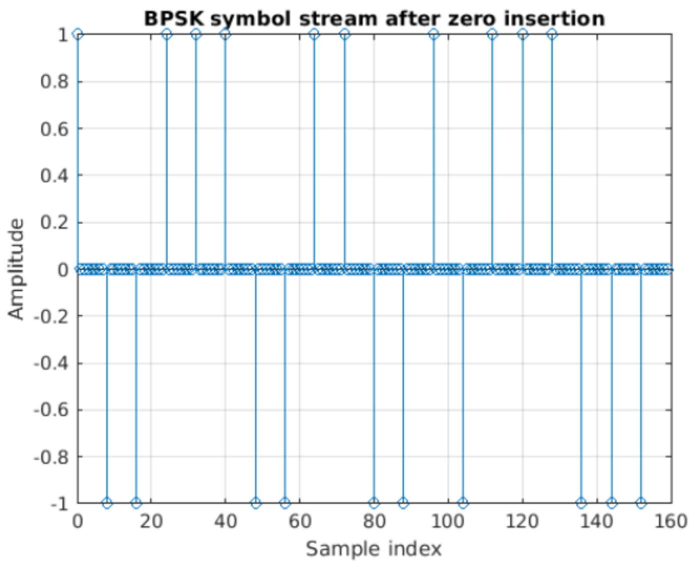

Magnitude of the DFT of the BPSK symbol stream
Magnitude in dB
Normalized frequency f
20
18
16
14
12
10
8
6
4
2
0
-0.5
0
0.5

Step 2b Pulse Shaping: Insert zeros in the generated BPSK symbol stream and convolve with the SRRC pulse

Understand the following:

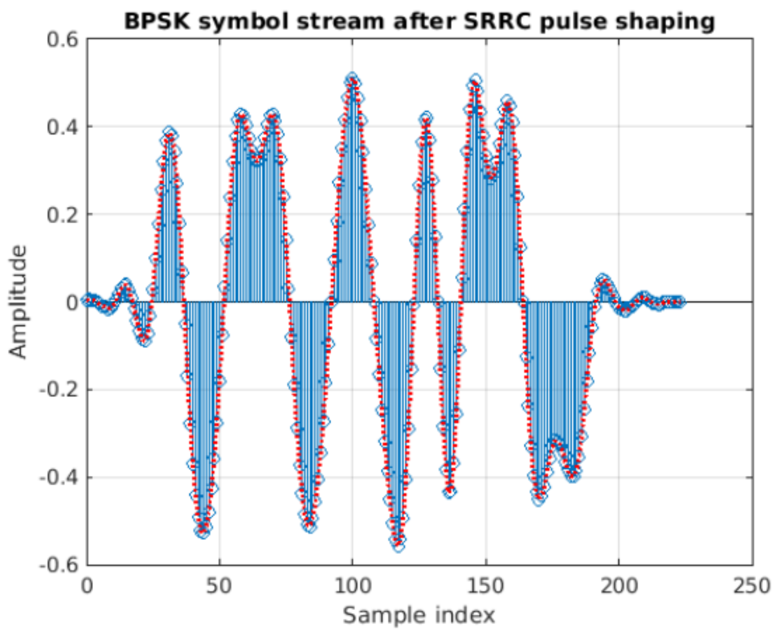
1. Why we are inserting samples_per_symbol-1 zeros in between two BPSK symbols?

   We are inserting the samples_per_symbol-1 zeros in between two BPSK because we want the signals to convolve only at frequencies which are in frequency spectrum of in impulse response of the signal used to convolve it. Otherwise we get some stray signals also.



2. What is the expected output when we convolve this zero-inserted sample-stream with the SRRC pulse?

   As we have made all the symbols between two bpsk symbols now on convolving the output should be such that the bpsk symbol will take shape of SRRC pulse. Means whenever we have -1 symbol it will go downside and upside for +1 symbol.

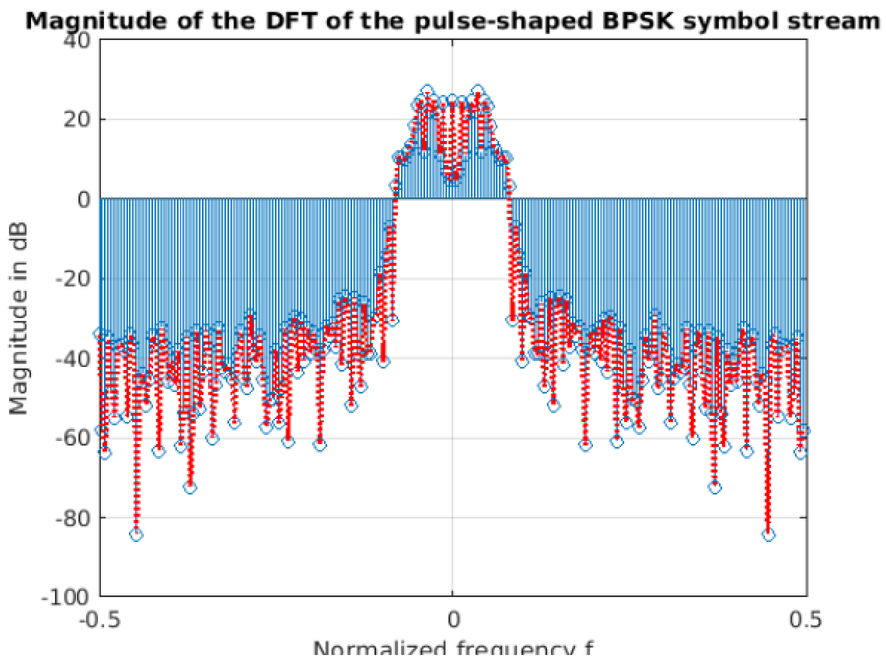


3. Notice that the spikes in the time-domain are replaced by a smoothened signal after the pulse-shaping. In general, the spikes in the transmitted signal are not desirable, they lead to hardware implemenation problems and also are spectrally wasteful. The pulse-shaping smoothens the BPSK symbol stream with the spikes and resolves these issues.

   Smoothness can be seen in above curve, after performing convolution of zero inserted symbol.

4. How does the DFT of the pulse-shaped BPSK symbol stream obtained in this step compare with the DFT of the SRRC pulse plotted in Step 1?

   If we carefully observe this curve obtained by performing Fourier transform of signal generated after pulse shaping then this is quite similar to rectangular pulse shaped SRRC pulse. Both got some bulkiness in smaller values around zero.



5. How does the DFT of the pulse-shaped BPSK symbol steam obtained in this step compare with the DFT of the original in Step 2a?

6. Notice that the spectral occupancy of the pulse-shaped signal is restricted to a small range of the overall spectral width of [-0.5, +0.5]. This was not the case in Step 2a without the pulse-shaping.

   This happens because after convolution now we don't need to see the whole data stream. We will simply look at those values of k, that represent the impulse response of impulse pulse (SRRC in this case).

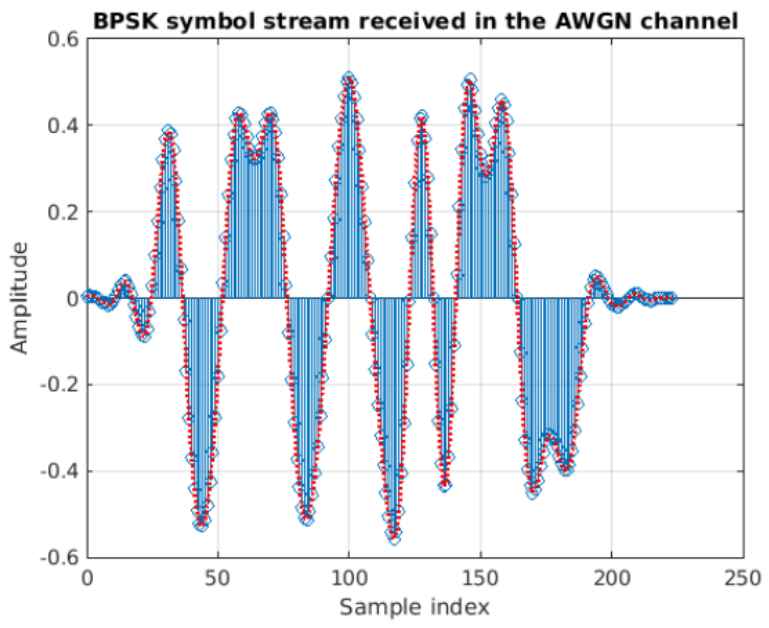
Step 3 AWGN Channel: Add the additive white Gaussian noise and analyze the received signal at the channel output

- Observe the effect of the AWGN on the time-domain and the frequency-domain plots of the received signal.
- Compare these two plots with the corresponding plots obtained in Step 2b for the transmitted signal in the absence of noise.
- Notice that since the noise is spectrally-white, it shows up in the spectral band outside of roughly [-0.1,+0.1] band over which the transmitted signal is located. This out-of-band noise which does not spectrally overlap with the signal can be removed without affecting the desired BPSK portion of the noisy received signal.

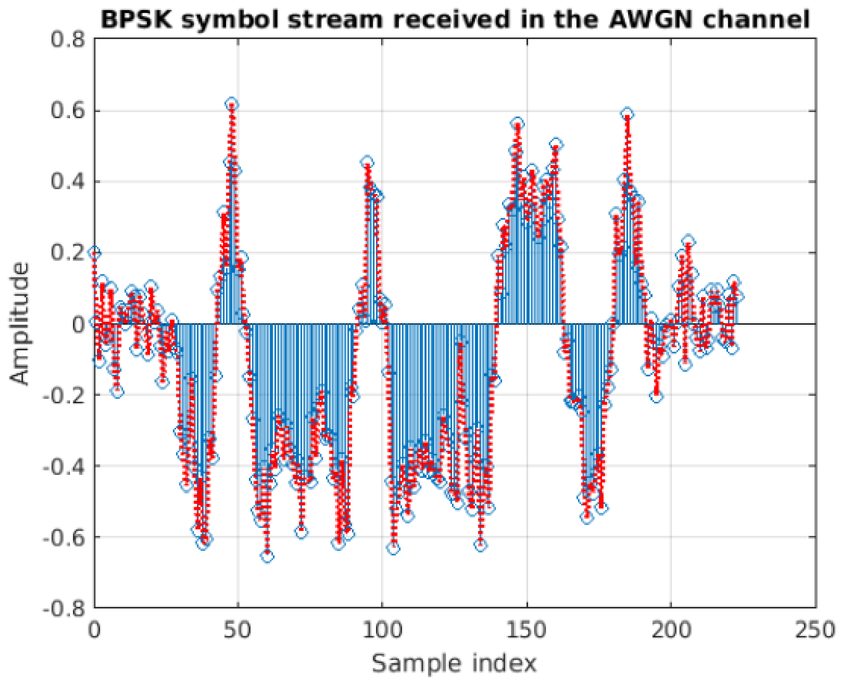
Change the value of sigma_noise to dial up/down the strength of the additive noise

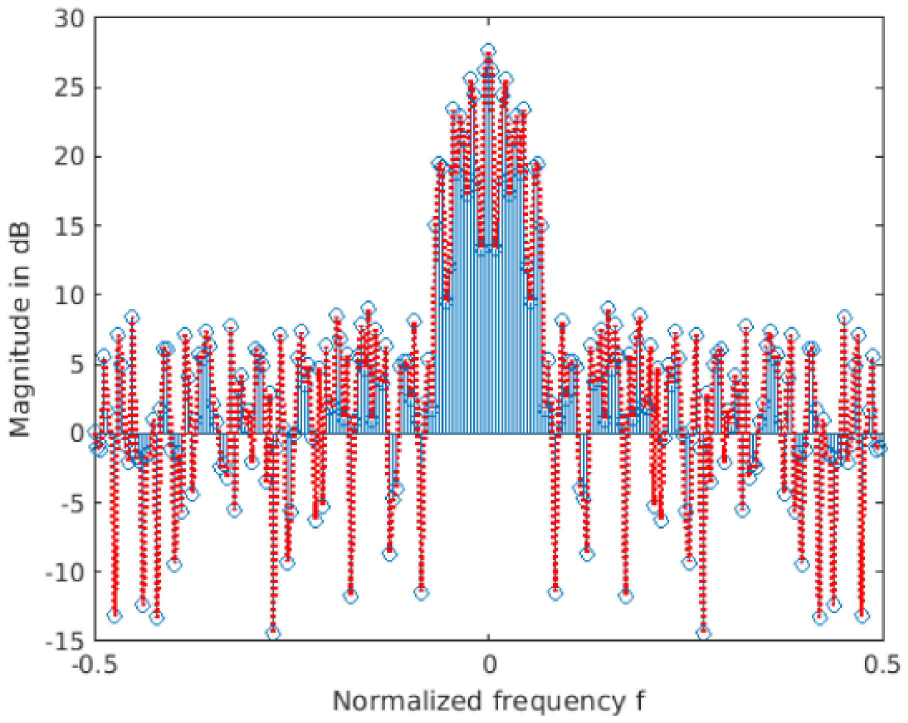
```
sigma_noise = 0; % the standard deviation of the Gaussian noise
s_received = s_pulseShaped + sigma_noise*randn(NTotal,1);
stem(nTotal,s_received); % plot the BPSK symbol stream after pulse shaping
hold on; plot(nTotal,s_received,'r:','linewidth',2); hold off
xlabel('Sample index'); ylabel('Amplitude');
title('BPSK symbol stream received in the AWGN channel'); grid


S_f_received = fftshift(fft(s_received)); % the Fourier Transform
plotVariable = 20*log10(abs(S_f_received)); stem(kTotal,plotVariable)
hold on; plot(kTotal,plotVariable,'r:','linewidth',2); hold off
xlabel('Normalized frequency f'); ylabel('Magnitude in dB');
title({'Magnitude of the DFT of the pulse-shaped BPSK symbol stream',...
    'received in the AWGN channel'}); grid
```

For noise = 0. We get the same output as in previou case:



for noise = 0.1





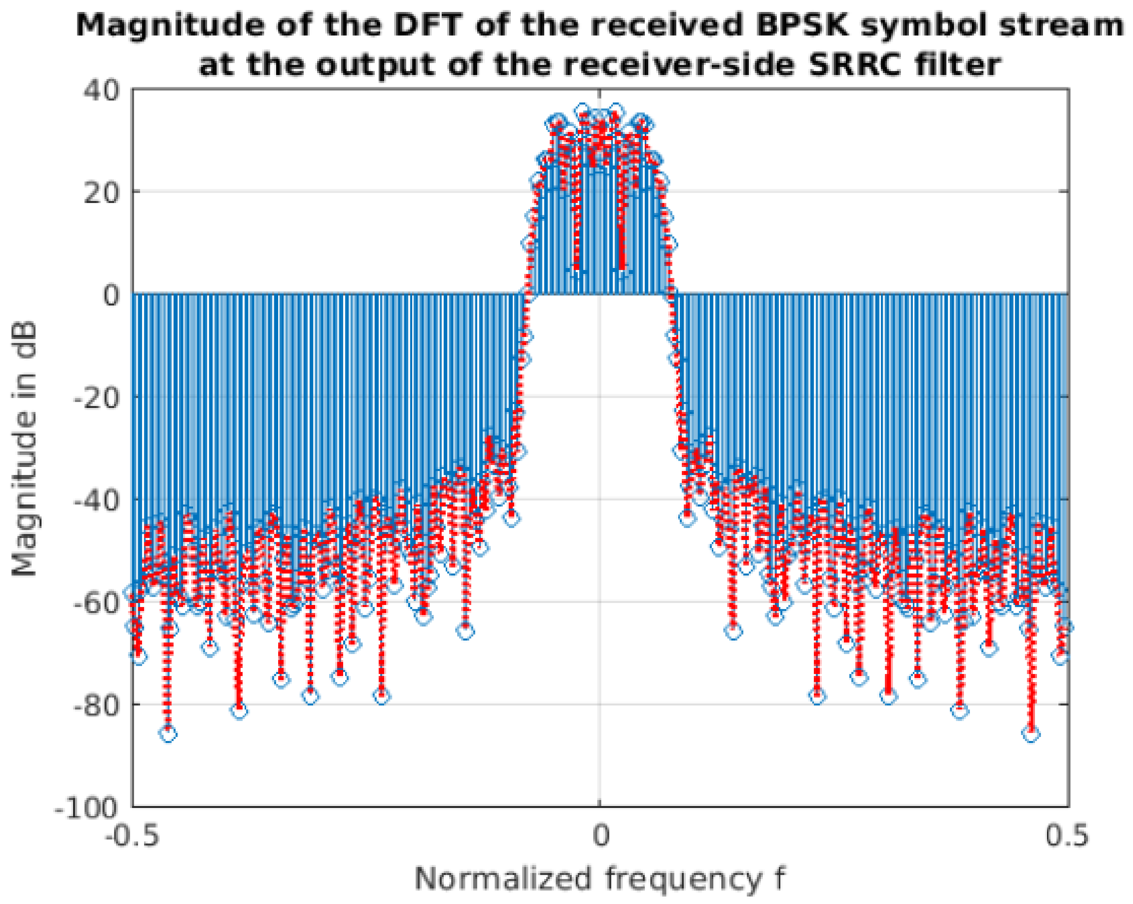
Here we observe that the frequency domain is affected more by noise than compared to time domain.

Step 4 Receiver Side SRRC Filtering

Pass the received signal through the receiver-side SRRC noise-removing filter. This filter is nothing but the convolution operation, i.e., the received noisy signal is convolved with the SRRC pulse. The SRRC pulse, in this case, acts as a filter. Its effect is to "filter-out" the noise which is outside of the spectral band of interest

- The objective of this receiver-side SRRC filtering is to remove the out-of-band noise which does not spectrally overlap with the signal.
- Observe the realization of this objective. The band corresponding to the transmitted pulse-shaped BPSK signal is preserved in the magnitude, however, the out-of-band noise is signficantly reduced compared to the corresponding spectral plot obtained in Step 3
- This occurs because convolution in the temporal domain is equivalent to the product in the frequency domain. The spectral shape of the SRRC filter (plotted in Step 1) is multiplied with the spectrum of the received signal (plotted in Step 3). Since the SRRC filter magnitude outside of the band [-0.1, +0.1] of interest is extremely small (almost zero), the effect of this multplication operation is to zero-out the out-of-band AWGN component.

```
s_received_filtered = conv(h_n,s_received);
NTotal_filtered = length(s_received_filtered); nTotal_filtered =
0:NTotal_filtered-1;
stem(nTotal_filtered,s_received_filtered); % plot the BPSK symbol
stream after pulse shaping
hold on; plot(nTotal_filtered,s_received_filtered,'r:','linewidth',2);
hold off
xlabel('Sample index'); ylabel('Amplitude');
title('BPSK symbol stream received in the presence of additive White
Gaussian noise'); grid


S_f_received_filtered = fftshift(fft(s_received_filtered)); % the
Fourier Transform
kTotal_filtered =  (-NTotal_filtered/2:NTotal_filtered/2-
1)/NTotal_filtered;
plotVariable = 20*log10(abs(S_f_received_filtered));
stem(kTotal_filtered,plotVariable)
hold on; plot(kTotal_filtered,plotVariable,'r:','linewidth',2); hold
off
xlabel('Normalized frequency f'); ylabel('Magnitude in dB');
title({'Magnitude of the DFT of the received BPSK symbol stream','at
the output of the receiver-side SRRC filter'}); grid
```


Magnitude of the DFT of the received BPSK symbol stream
at the output of the receiver-side SRRC filter
Magnitude in dB
Normalized frequency f
40
20
0
-20
-40
-60
-80
-100
-0.5
0
0.5