Data Communication Lab File

Name :- Rajat Som
Branch :- CSAI-1
Roll No. :- 2021UCA1852
Course Code :- CAECC12

LIST OF EXPERIMENTS

1. To plot the spectrum of a pulse of width 10.

2. To verify following properties of Fourier Transform:
   i. Time Shifting
   ii. Frequency shifting
   iii. Convolutional

3. To generate Uniform random number and plot its density function. Find its mean and variance.

4. To generate Gaussian distributed random numbers and plot its density function. Find its mean and variance.

5. Compute the Signal to quantization Noise ratio of Uniform Quantization.

6. Compute the Signal to quantization Noise ratio of Non-Uniform Quantization. Plot SNR versus Quantization levels.

7. Study of passband digital communication technique BPSK. Calculate the BER of the BPSK modulated signal.

8. Given is a linear block code with the generator matrix G :
   G = 1 1 0 0 1 0 1
   0 1 1 1 1 0 0
   1 1 1 0 0 1 1

a. Calculate the number of valid code words N and the code rate RC. Specify the complete Code set C.

b. Determine the generator matrix G′ of the appropriate systematic (separable) code C’.

9. To generate a M/M/1 Queue having infinite buffer space with parameters (x,y) and plot the average delay per packet vs x/y.

10. To generate a M/M/1 Queue having finite buffer space with parameters (x,y) and plot blocking probability with respect to variation with buffer space.

Exp-1 :- To plot the spectrum of a pulse of width 10.

Theory :- A signal's frequency spectrum is its presentation in the frequency domain based on the Fourier Transform of its time domain function.The Fourier transform is a mathematical formula that transforms a signal sampled in time or space to the same signal sampled in temporal or spatial frequency. In signal processing, the Fourier transform can reveal important characteristics of a signal, namely, its frequency components.

Formula Used :- Fourier Transform of a Signal :

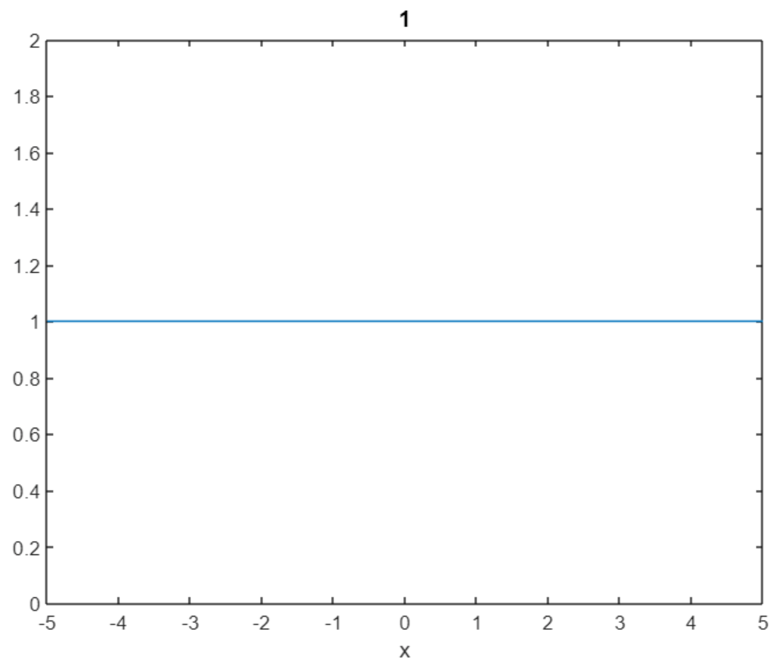
Fourier Transform (FT)

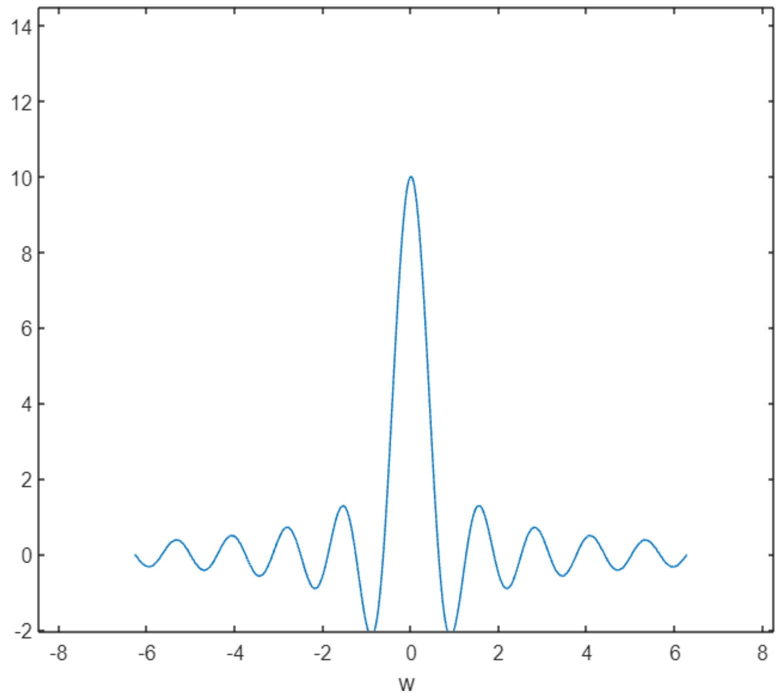
$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$
$$\omega \in (-\infty, +\infty)$$

Code :-

```
t(x1)
yp=yp(1:lex=1;
y=2;
figure(1);
ezplot(int(    x*

clc;
clear all;
close
all;
syms t w;

x=1;
figure(1);
ezplot('1',[-
5,5]);
ft = int(x*exp(-j*w*t),t,-5,5);
figure(2);
ezplot(ft);
```

<u>Output</u> :-





Exp-2 :- To verify following properties of Fourier Transform:

i. Time Shifting
ii. Frequency shifting
iii. Convolutional

Theory :- i).The time shifting property of Fourier transform states that if a signal □(□) is shifted by □0 in time domain, then the frequency spectrum is modified by a linear phase shift of slope (−□□0). Therefore,if □(□) -> □(□) Then, according to the time-shifting property of the Fourier transform.

ii). Frequency shifting property of the Fourier transform states that the multiplication of a time domain signal x(t) by an exponential(e^jω0t) causes the frequency spectrum to be shifted by ω0. Therefore, if □(□) -> □(□) Then, according to the frequency-shifting property.

iii). The convolution of two signals in the time domain is equivalent to the multiplication of their spectra in the frequency domain. Therefore, if h(t)↔H(ω) and x(t)↔X(ω) Then onconvolution.

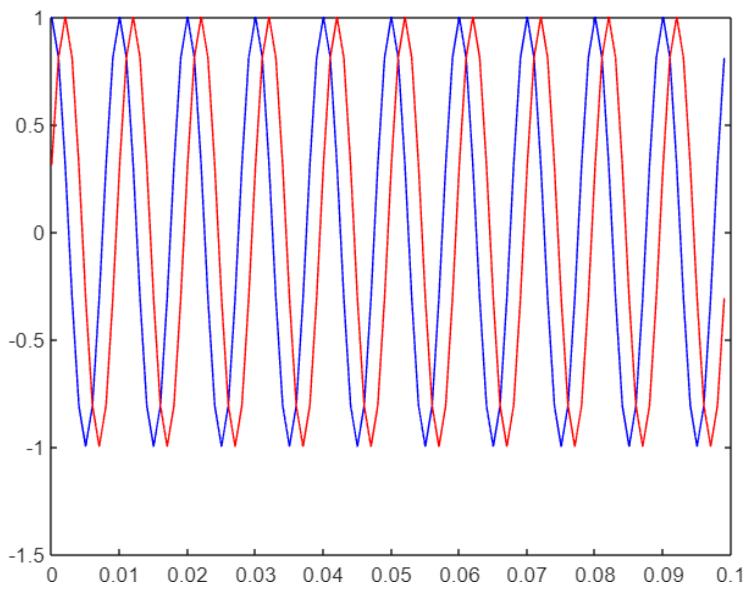
i. Time shifting

Code :-

```
clc
clear all
close all
t=0:0.001:0.1-0.001;
fs=1e3;
freq1=100;
x1=cos(2*pi*freq1*t);
Delay=2
yp=fft(x1)
yp=yp(1:length(x1)/2+1);
f=0:fs/length(x1):500;
yp=yp.*exp(-1i*2*pi*f*Delay*(1/fs))
yp=[yp conj(fliplr(yp(2:end-1)))];
y=ifft(yp,'symmetric');
```

```
plot(t(1:100),x1(1:100),'b');
hold on;
plot(t(1:100),y(1:100),'r');
```

Output :-



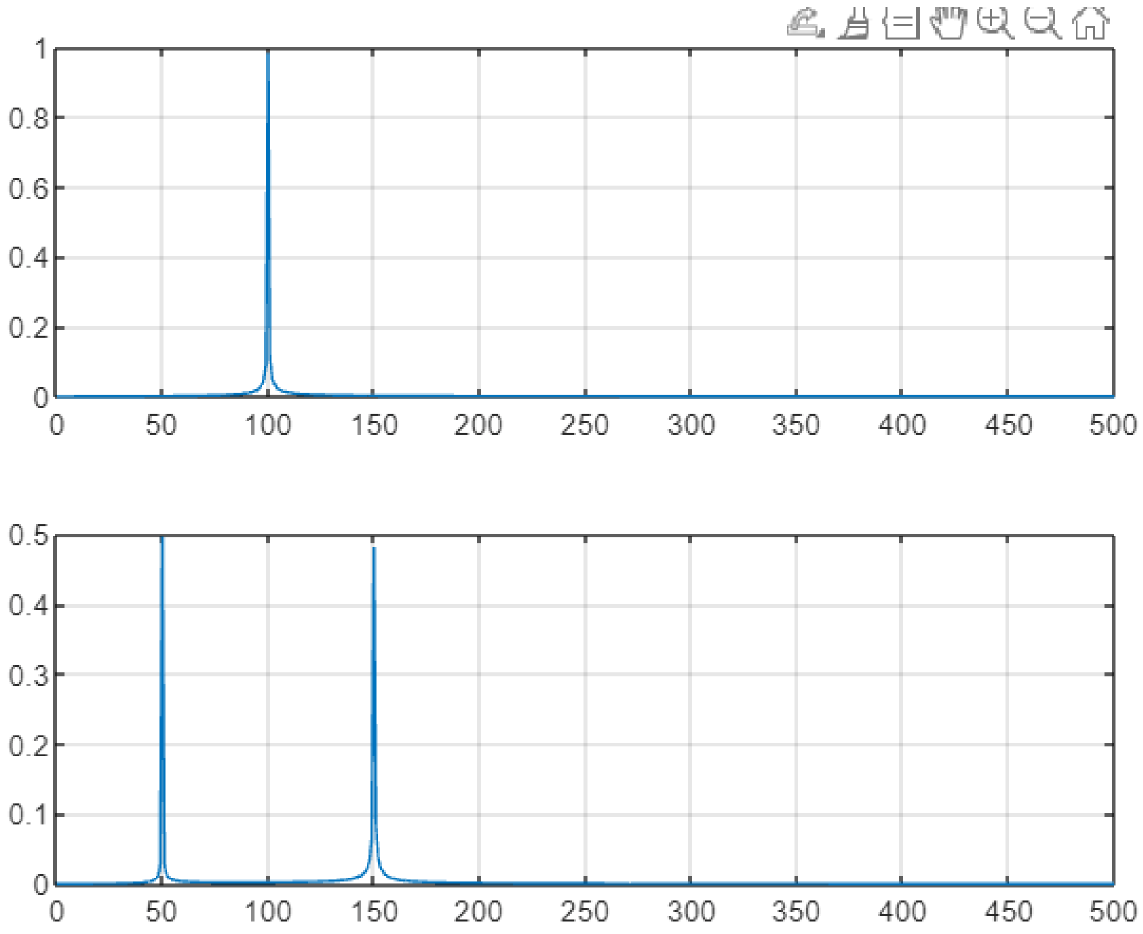
ii. Frequency shifting

Code :-

```
clc
clear all
close all
t=linspace(0,1,1000);
S=sin(2*pi*100*t);
Sc=sin(2*pi*50*t);
HS=S.*Sc;
Fs=1000;
Fn=Fs/2;
FT1=fft(S)/length(S);
FT2=fft(HS)/length(HS);
Fv=linspace(0,1,fix(length(FT1)/2)+1)*Fn;
Iv=1:length(Fv);
```

```
figure(1)
subplot(2,1,1);
plot(Fv,2*abs(FT1(Iv)))
grid
subplot(2,1,2)
plot(Fv,2*abs(FT2(Iv)))
grid
```

<u>Output</u> :-

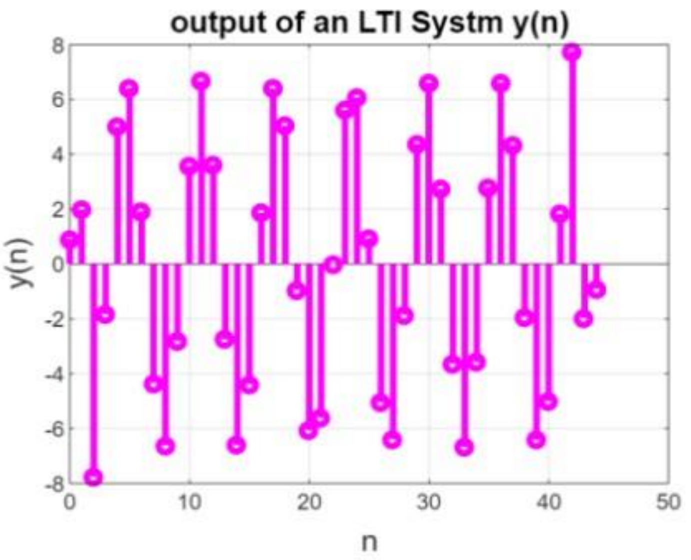


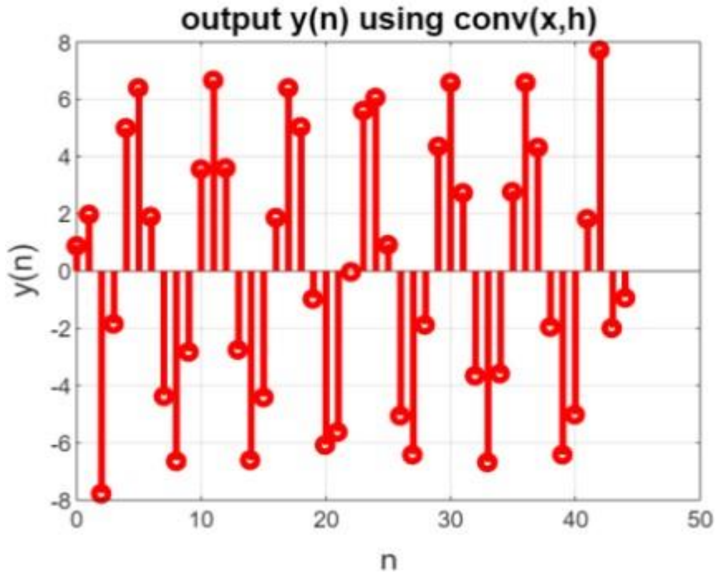
iii. Convolution

Code :-

```
clear variables
close all
n= -20:20;
x=sin(n);
h=[-1,-2,8,-2,-1];
N=length(x);
M=length(h);
Ny=N+M-1;
y=zeroes(1,Ny);
for i=1:N
for k=1:M
y(i+k-1)=y(i+k-1)+h(k)*x(i);
end
end
m=0:Ny-1;
figure
stem(m,y,'linewidth',3,'color','m')
grid;
a=title('output of an LTI Systm y(n)');
set(a,'fontsize',14);
a=ylabel('y(n)');
set(a,'fontsize',14);
a=xlabel('n');
set(a,'fontsize',14);
figure
y2=conv(x,h);
stem(m,y2,'linewidth',3,'color','r')
grid;
a=title('output y(n) using conv(x,h)');
set(a,'fontsize',14);
a=ylabel('y(n)');
set(a,'fontsize',14);
a=xlabel('n');
set(a,'fontsize',14);
```

Output :-





Exp-3 :- To generate Uniform random number and plot its density function. Find its mean and variance.

Theory :- The uniform distribution is also known as rectangular distribution. It is a two-parameter family of curves and it has a constant probability distribution function between its two bounding parameters. The pdf of the uniform distribution is as follows:

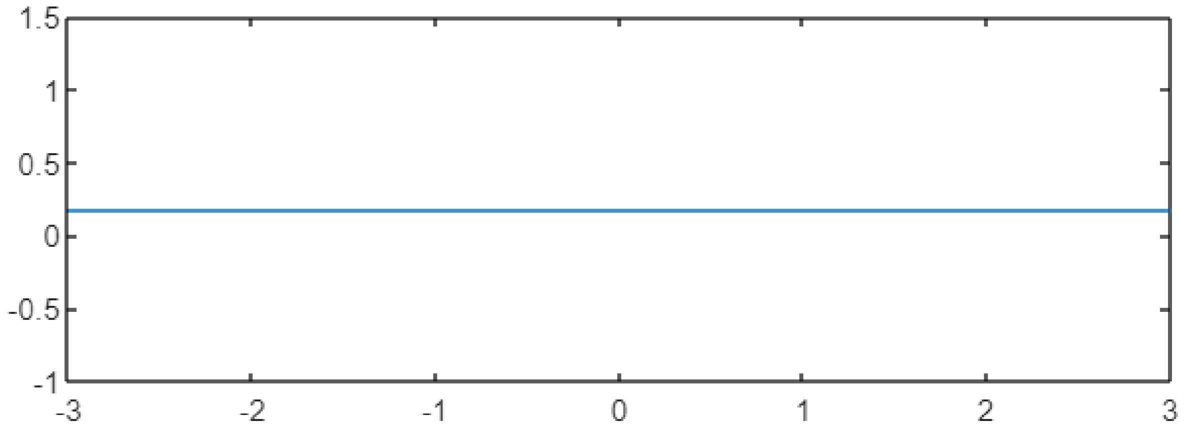
$$P(x) = \begin{cases} 0 & \text{for } x < a \\ \dfrac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{for } x > b \end{cases}$$

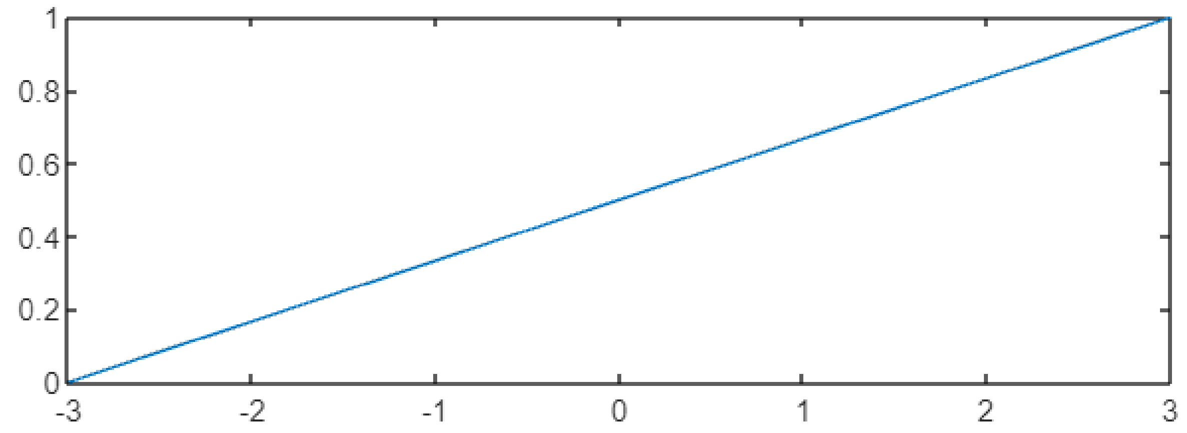
The mean of the uniform distribution is μ = (□ + □)/2 2.
The variance of the uniform distribution is σ = (□ − □)^2/12.

Code :-

```
clear;
close all;
clc;
n=1000;
a=0;
b=6;
mu=(b+a)/2;
v=((b-a)^2)/12;
sigma2=v;
range=-3:0.1:3;
size(range)
pdfu=1/(b-a)*range.^0;
subplot(2,1,1);
plot(range,pdfu);
cdfu=cdf('Uniform',range,-3,3);
subplot(2,1,2);
plot(range,cdfu);
disp(mu);
disp(v);
```

Output :-





Exp-4 :- To generate a Gaussian distributed random number and plot its density function. Find its mean and variance.

Theory :- The normal distribution, also known as the Gaussian distribution, is a two-parameter family of curves. The sum of independent samples from any distribution with finite mean and variance converges to the normal distribution as the sample size goes to infinity. The normal probability density function (pdf) is:

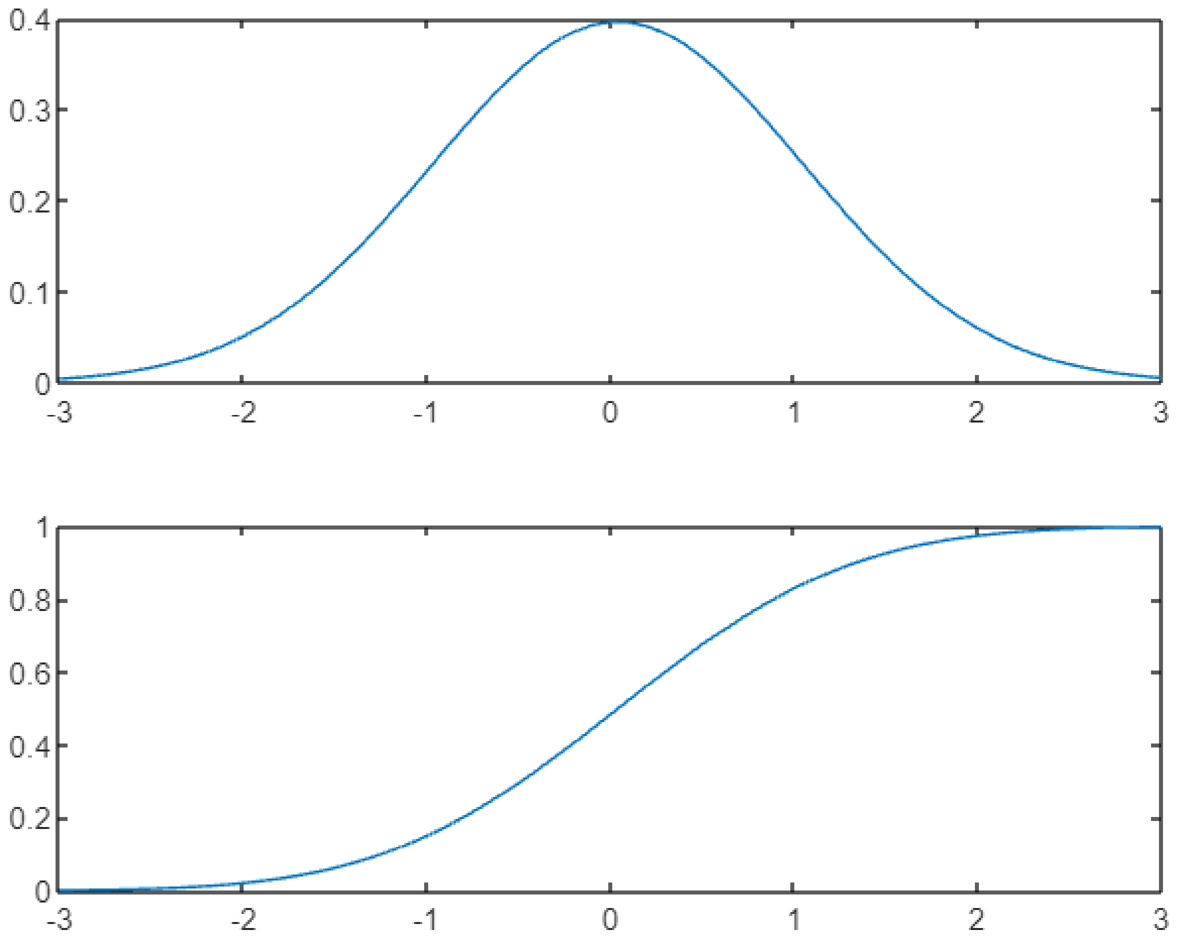
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The mean of the gaussian distribution is μ .
The variance of the gaussian distribution is σ^2.

Code :-

```
clear;
close all;
clc;
n=1000;
r=randn(1,n);
mu=mean(r);
v=var(r);
sigma2=var(r);
range=-3:0.1:3;
pdfn=1/(sqrt(2*pi*sigma2))*exp(-((range-mu).^2)/(2*sigma2));
subplot(2,1,1);
plot(range,pdfn);
cdfn=cdf('Normal',range,mu,sqrt(sigma2));
subplot(2,1,2);
plot(range,cdfn);
```

<u>Output</u> :-



Exp-5 :- Compute the Signal to quantization Noise ratio of Uniform Quantization. Plot SNR versus Quantization levels.

Theory :- The SQNR(signal-to-noise quantization ratio) reflects the relationship between the maximum nominal signal strength and the quantization error (also known as quantization noise) introduced in the analog-to-digital conversion. The SQNR is used asa quick error metric in regard to quantization noise. Equation: Let Ps be the signal powerand Pn be the quantization noise. The SQNR and its decibel version, SQNR (in dB), are defined as follows:

$$\mathrm{SQNR} \triangleq \mathrm{P}_s/\mathrm{P}_n$$

$$\mathrm{SQNR}_{dB} \triangleq 10\log_{10}\mathrm{SQNR}$$

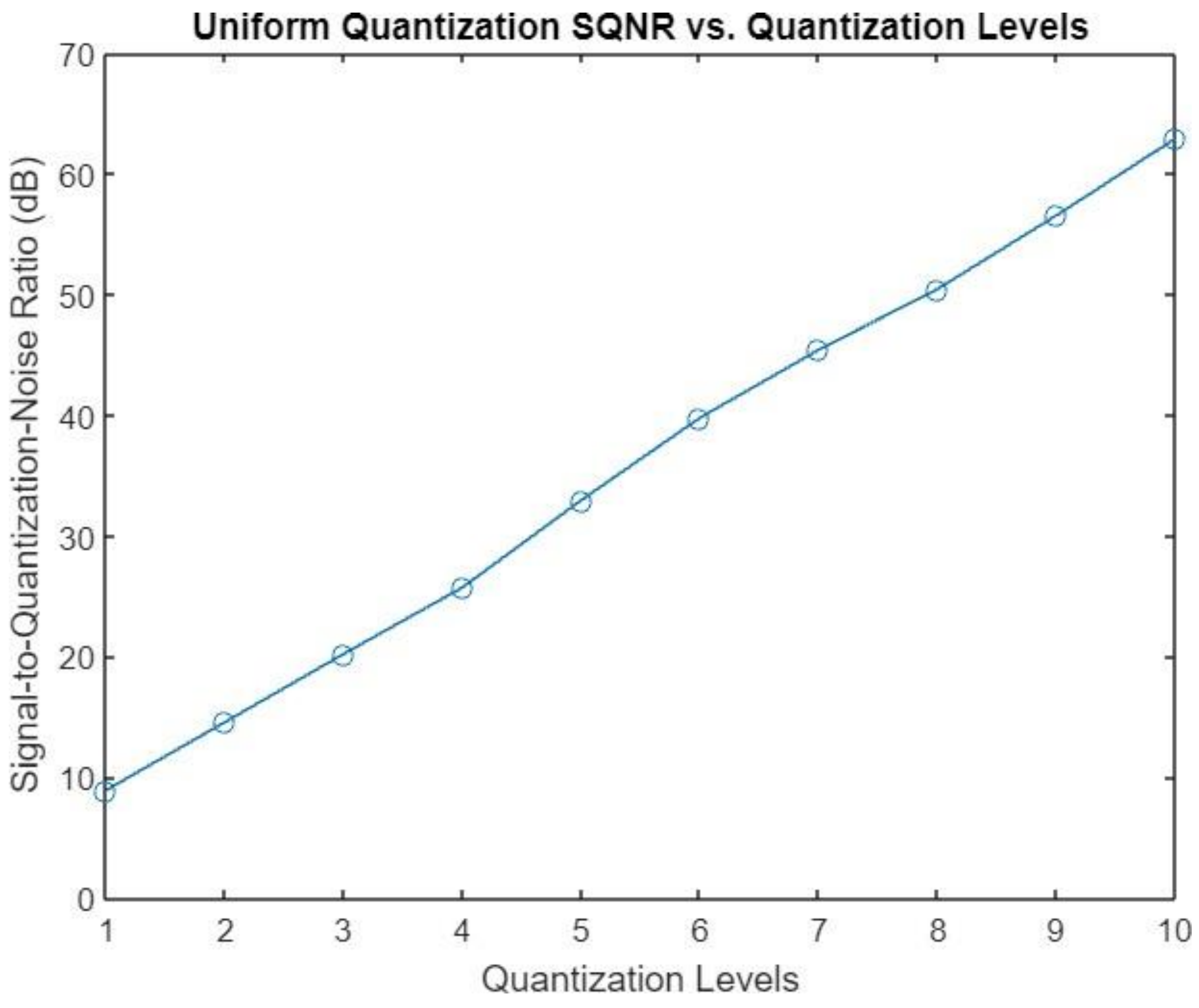
The type of quantization in which the quantization levels are uniformly spaced is termed Uniform Quantization. The type of quantization in which the quantization levels are unequal and mostly the relation between them is logarithmic, is termed as a Non-uniform Quantization.

Code :-

```
N = 1000; % number of samples
fs = 1000; % sampling frequency (Hz)
f = 10; % signal frequency (Hz)
A = 1; % signal amplitude
Q = 10; % maximum quantization level
% Generate the signal
t = (0:N-1)'/fs; % time vector
x = A*sin(2*pi*f*t); % sinusoidal signal
% Compute the SQNR for each quantization level
sqnr = zeros(Q, 1);
for q = 1:Q
% Quantize the signal
xq = round(x*(2^(q-1))/A)*(A/(2^(q-1)));
% Compute the quantization error and the SQNR
```

```
e = x - xq;
sqnr(q) = 10*log10(var(x)/var(e));
end
% Plot the SQNR versus quantization levels
plot(1:Q, sqnr, 'o-')
xlabel('Quantization Levels')
ylabel('Signal-to-Quantization-Noise Ratio (dB)')
title('Uniform Quantization SQNR vs. Quantization Levels')
```

<u>Output</u> :-



Exp-6 :- Compute the Signal to quantization Noise ratio of Uniform Quantization. Plot SNR versus Quantization levels.

Theory :- SQNR stands for Signal-to-Quantization-Noise Ratio. It is a measure of the quality of a digital signal after it has been digitized by a quantizer. In other words, it is the ratio between the power of the original signal and the power of the quantization noise introduced during the digitization process.

The type of quantization in which the quantization levels are unequal and mostly the relation between them is logarithmic, is termed as a Non-uniform Quantization.

Let Psignal be the signal power and P signal be the quantization noise. SQNR (in dB) is:

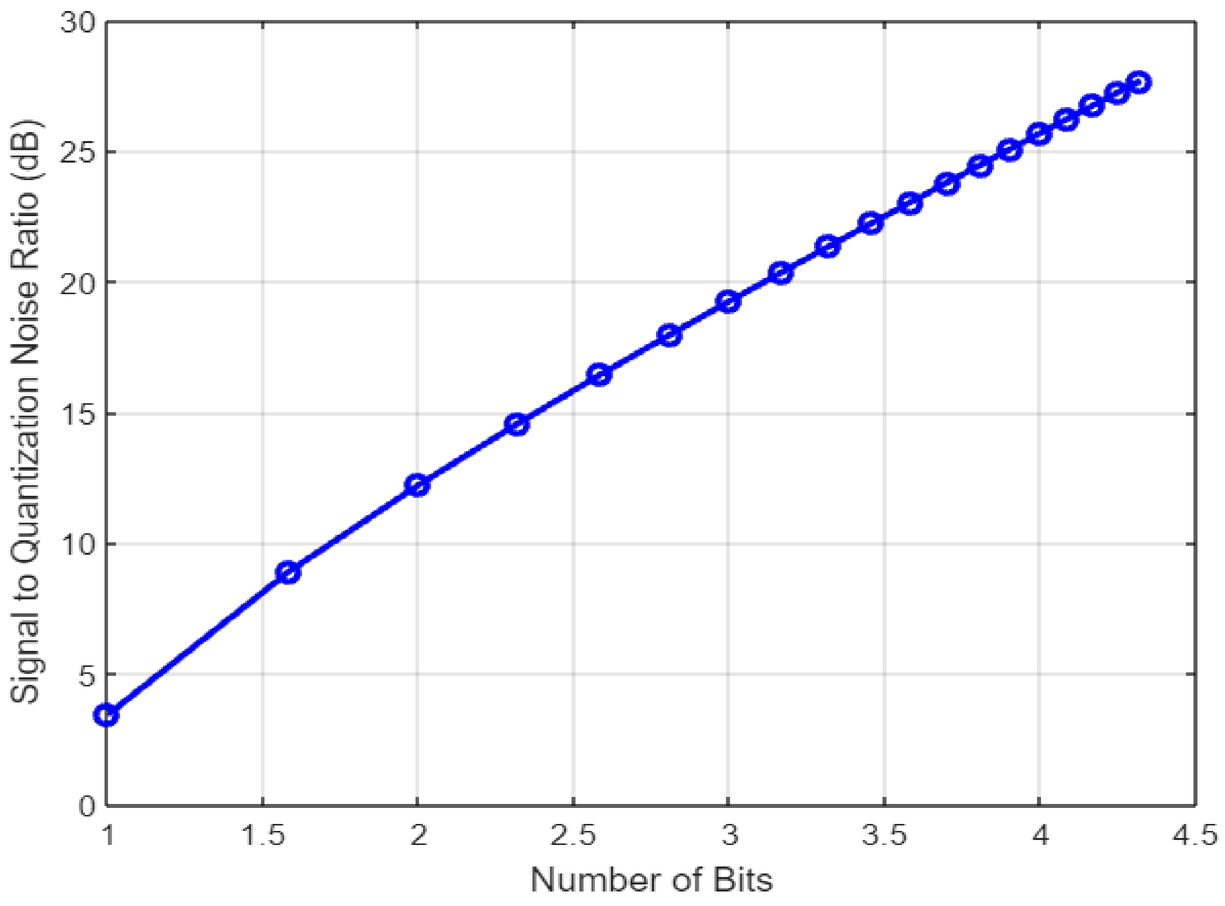
$$SNR(dB) = 10\log\frac{P_s}{P_n}$$

Code :-

```
clear all;
close all;
clc;
N = 10000;
f = 1;
Fs = 1000;
t = (0:N-1)/Fs;
x = sin(2*pi*f*t);
L = 2:20;
b = log2(L);
Delta = 2./(L-1);
SQNR = zeros(length(L),1);
for i=1:length(L)
q = zeros(size(x));
V = [-(L(i)-1)/2 : 1 : (L(i)-1)/2] * Delta(i);
```

```
for j=1:N
[val, index] = min(abs(x(j)-V));
q(j) = V(index);
end
noise = x - q;
signal_power = sum(x.^2)/N;
noise_power = sum(noise.^2)/N;
SQNR(i) = 10*log10(signal_power/noise_power);
end
figure;
plot(b, SQNR, 'b-o', 'LineWidth', 2);
xlabel('Number of Bits');
ylabel('Signal to Quantization Noise Ratio (dB)');
grid on;
```

<u>Output</u> :-



Exp-7 :- Study of passband digital communication technique BPSK. Calculate the BER of the BPSK modulated signal.

Theory :- Bit error rate (BER) of a communication system is defined as the ratio of number of error bits and total number of bits transmitted during a specific period. It is the likelihood that a single error bit will occur within received bits, independent of rate of transmission.

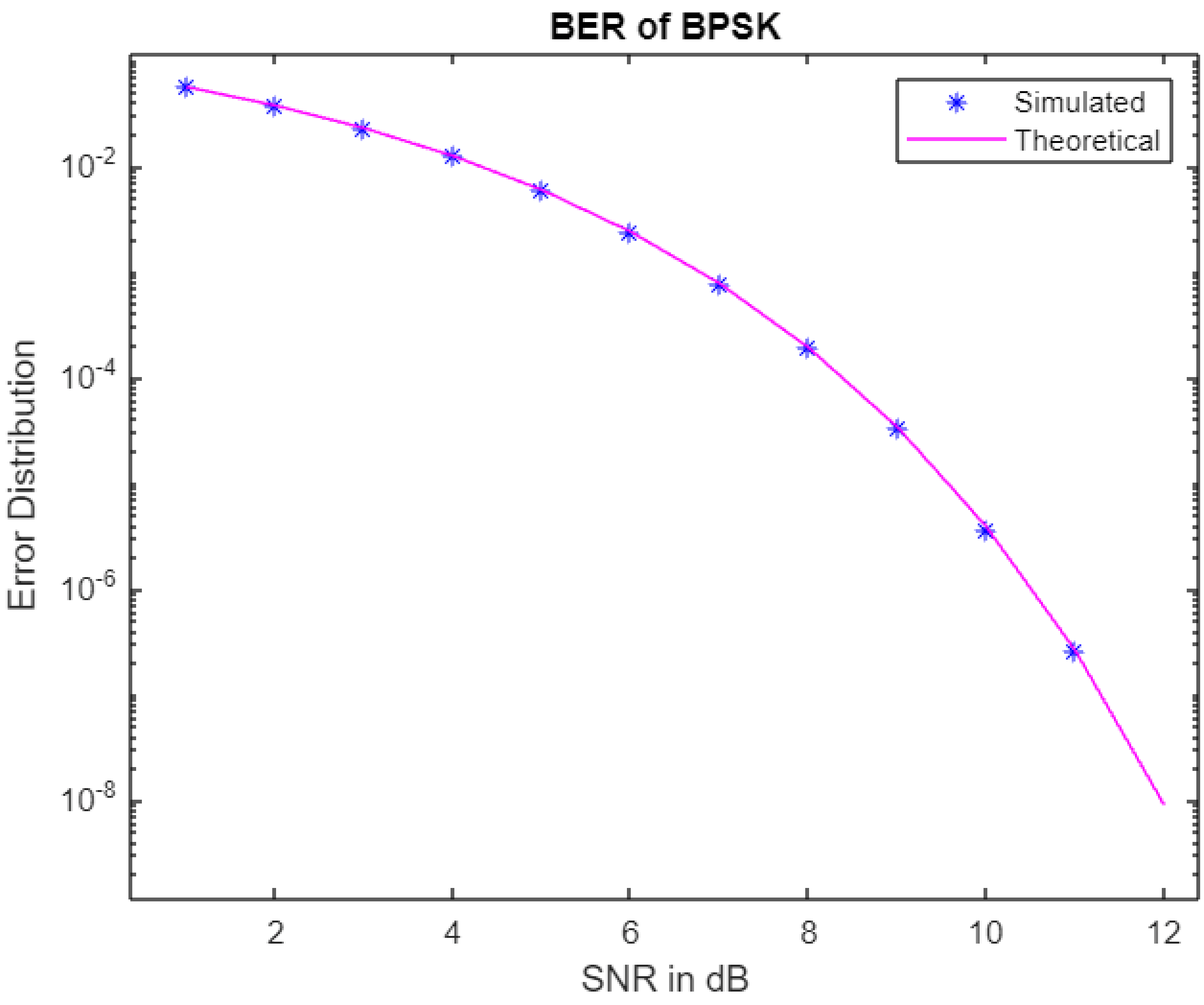
$$P_{BPSK}(e) = \frac{1}{2} erfc\left[\sqrt{\frac{E}{N_0}}\right]$$

Code :-

```
clc; clear
all;close
all;

N = 10^8;
a =rand(1,N) > 0.5;s
= 2.*a-1;
n = (1/sqrt(2))*(randn(1,N) + 1i*rand(1,N));snr_dB
= 1:1:12;
snr_ratio = 10.^(0.1.*snr_dB);

for i=1:length(snr_dB)
 y = 10.^(0.05.*snr_dB(i)).*s + n;adec
 = real(y)>0;
 err(i) = size(find(a-adec),2);end

sim_avg_err = err/N;
th_avg_err = 0.5 * erfc(sqrt(snr_ratio));
figure(1); semilogy(snr_dB,sim_avg_err,'b*');
hold onsemilogy(snr_dB,th_avg_err,'m-');
```

```
title("BER of BPSK");
legend('Simulated','Theoretical');
xlabel('SNR in dB');
ylabel('Error Distribution');
```

<u>Output</u> :-



Exp-8 :- Given is a linear block code with the generator matrix G :

G = 1 1 0 0 1 0 1
0 1 1 1 1 0 0
1 1 1 0 0 1 1

a. Calculate the number of valid code words N and the code rate RC. Specify thecomplete Code set C.

b. Determine the generator matrix G′ of the appropriate systematic (separable)code C’.

Theory :- The generator matrix is a matrix of basis vectors. The rows of the generator matrix G are used to derive the actual transmitted codewords. The generator matrix G for an (n,k) block code can be used to generate the appropriate n-digit code word from any given k -digit data sequence. For every [k × n] generator matrix G of the code, there exists a matrix H of dimension [n − k × n] such that the row space of G is orthogonal to the column space of H^T, i.e., $GH^T = 0$.
Therefore, for matrix G, matrix H is such that the inner product of a vector in the row space of G and the corresponding rows of H is zero.

$$G = [\,P|I_k\,]$$

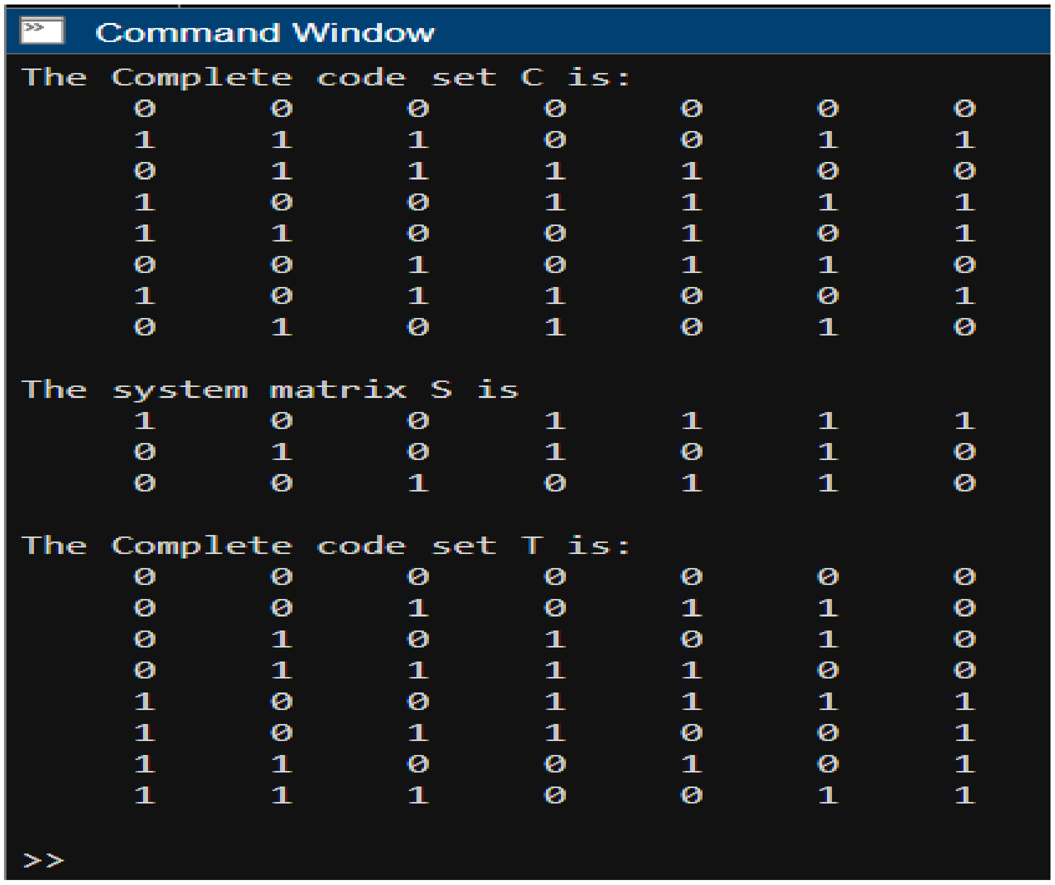
$$H = [\,-P^\top|I_{n-k}\,]$$

Code :-

```
% Given is a linear block code with the generator matrix
% G G = 1 1 0 0 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 1 a.
% Calculate the number of valid code words N and the code rate RC.
% Specify the complete Code set C. b. Determine the generator matrix G′
% of the appropriate systematic (separable) code C’.
clc;
clear all;
close
all;
G = [1,1,0,0,1,0,1; 0,1,1,1,1,0,0;1,1,1,0,0,1,1];
m = [0,0,0;0,0,1;0,1,0;0,1,1;1,0,0;1,0,1;1,1,0;1,1,1];
C = mod(m*G,2);
```

```
disp('The Complete code set C is:');
disp(C);

G(3,:)=mod(G(3,:)+G(1,:),2);
G(2,:)=mod(G(2,:)+G(3,:),2);
G(1,:)=mod(G(1,:)+G(2,:),2);

disp('The system matrix S is');
disp(G);
m = [0,0,0;0,0,1;0,1,0;0,1,1;1,0,0;1,0,1;1,1,0;1,1,1];
T= mod(m*G,2);

disp('The Complete code set T is:');
disp(T);
```

Output :-



```
Command Window
The Complete code set C is:
     0     0     0     0     0     0     0
     1     1     1     0     0     1     1
     0     1     1     1     1     0     0
     1     0     0     1     1     1     1
     1     1     0     0     1     0     1
     0     0     1     0     1     1     0
     1     0     1     1     0     0     1
     0     1     0     1     0     1     0

The system matrix S is
     1     0     0     1     1     1     1
     0     1     0     1     0     1     0
     0     0     1     0     1     1     0

The Complete code set T is:
     0     0     0     0     0     0     0
     0     0     1     0     1     1     0
     0     1     0     1     0     1     0
     0     1     1     1     1     0     0
     1     0     0     1     1     1     1
     1     0     1     1     0     0     1
     1     1     0     0     1     0     1
     1     1     1     0     0     1     1

>>
```

Exp-9 :- To generate a M/M/1 Queue having infinite buffer space with parameters (x,y) and plot the average delay per packet vs x/y.

Theory :- Markovian; M/M/1 means that the system has a Poisson arrival process, an exponential service time distribution, and one server. Queuing theory provides exact theoretical results for some performance measures of an M/M/1 queuing system and this model makes it easy to compare empirical results with the corresponding theoretical results.

$$Q = \begin{pmatrix} -\lambda & \lambda & & & \\ \mu & -(\mu+\lambda) & \lambda & & \\ & \mu & -(\mu+\lambda) & \lambda & \\ & & \mu & -(\mu+\lambda) & \lambda \\ & & & & \ddots \end{pmatrix}$$

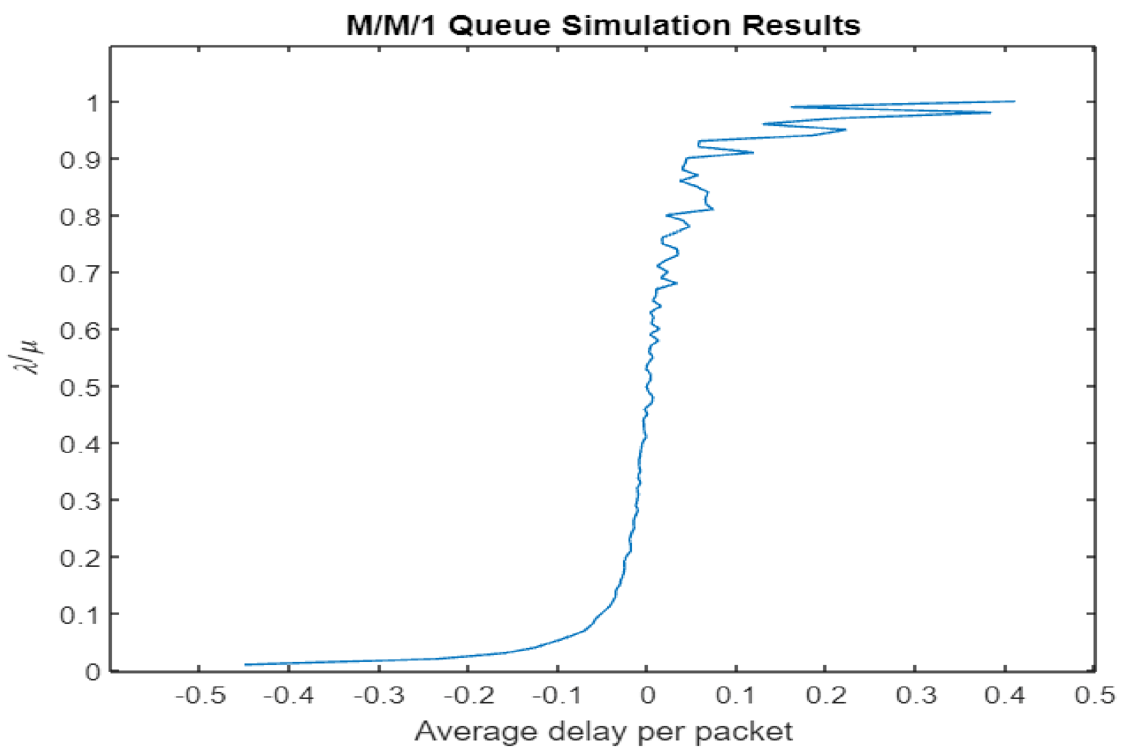
Code :-

```
clc;clear all;close all;

queue_lim = 200000;                    % system limit
sim_packets = 750;                     % number of clients to be simulatedmu
= 100;                                % service rate

% Define range of arrival rates to simulate
lambda_min = 1;lambda_max = 100; lambda_step =
1;

% Initialize results arrays
lambda_vals = lambda_min:lambda_step:lambda_max;
avg_delay = zeros(size(lambda_vals));

% Simulate M/M/1 queue for each value of lambdafor i
= 1:length(lambda_vals)

   % Define arrival rate for this simulationlambda =
   lambda_vals(i);

   % Initialize simulation variables
```

```
server_busy = false;      queue_size = 0;
total_delay = 0;     last_event_time = 0;

% Initialize event calendar
next_arrival_time = exprnd(1/lambda);
next_departure_time = Inf;

% Run simulation until target number of packets are serviced
packets_serviced = 0;
while packets_serviced < sim_packets

    % Determine next event time and type
    [event_time,  event_type]  =  min([next_arrival_time,  next_departure_time]);

    % Update queue statistics based on time since last event
    queue_size = queue_size + server_busy*(event_time-last_event_time);total_delay =
    total_delay + queue_size*(event_time-last_event_time); last_event_time = event_time;

    % Handle next event
    if event_type == 1 % arrival event

        % Schedule next arrival event
        next_arrival_time = event_time + exprnd(1/lambda);

        % If server is busy, add packet to queueif
        server_busy
            queue_size = queue_size + 1;if
            queue_size > queue_lim
                error('Queue overflow!');end
        else % Otherwise, start service immediately
            server_busy = true;
            next_departure_time = event_time + exprnd(1/mu);end

    else % departure event

        % Update statistics
        packets_serviced = packets_serviced + 1; server_busy = false;
        total_delay = total_delay + (event_time - next_departure_time);
```

```
      % Check if there are packets in the queueif
      queue_size > 0
         queue_size = queue_size - 1;server_busy =
         true;
         next_departure_time = event_time + exprnd(1/mu);else
         next_departure_time = Inf;end
    end
  end
  % Compute average delay for this simulation
  avg_delay(i) = total_delay/sim_packets;
end

% Plot results
figure;
plot(avg_delay,lambda_vals/mu); ylabel('\lambda/\mu');xlabel('Average
delay per packet'); title('M/M/1 Queue Simulation Results');
```

<u>Output</u> :-



Exp-10 :- To generate a M/M/1 Queue having finite buffer space with parameters (x,y) and plot blocking probability with respect to variation with buffer space.

Theory :- Finite-buffer M/G/1 priority queueing model where nonpreemptive time priority is given to delay-sensitive traffic and push-out space priority is given to loss-sensitive traffic. Compared to the previous study on finite-buffer M/M/1 priority queues with time and space priority, where service times are identical and exponentially distributed for both types of traffic.

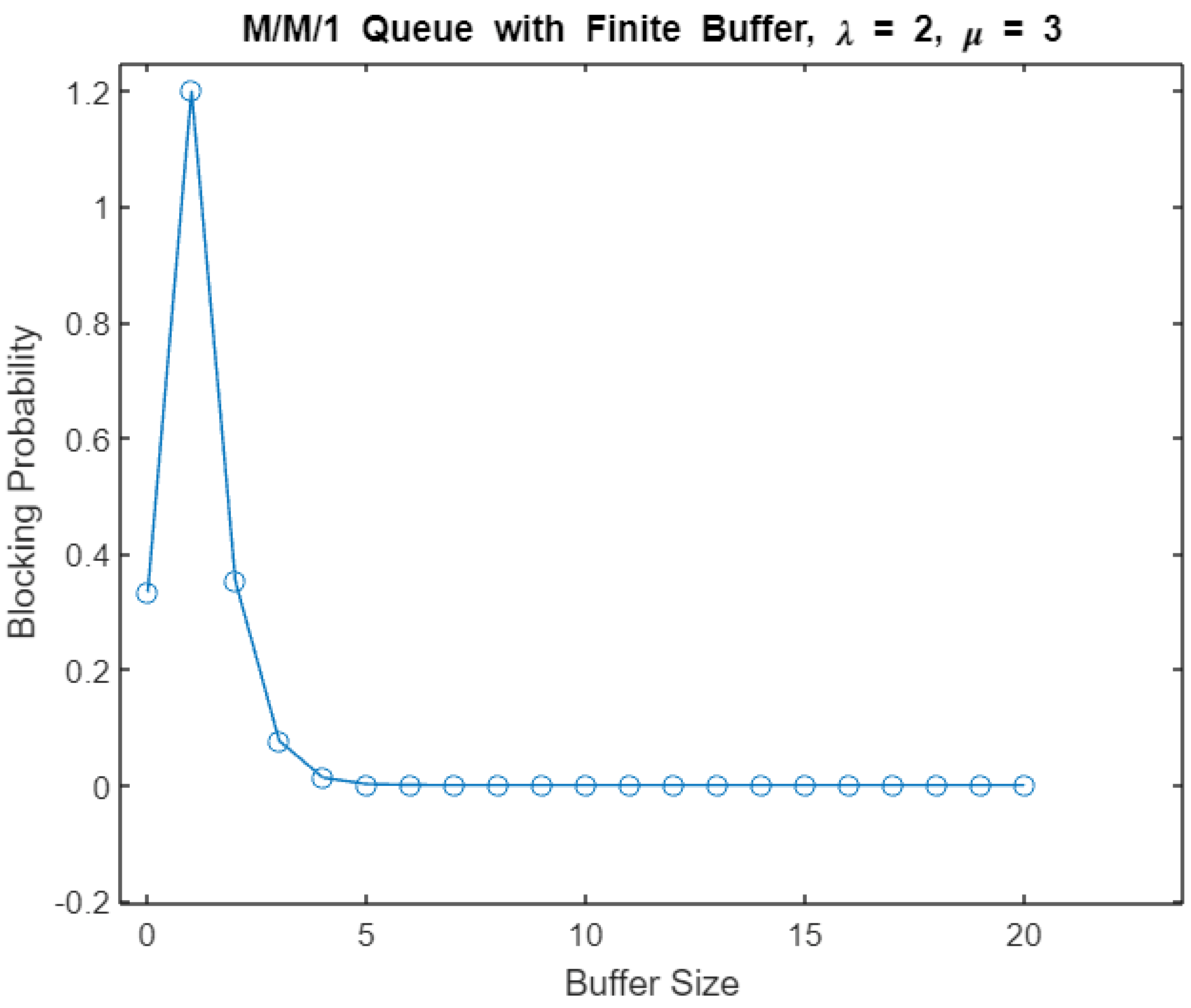
$$Q = \begin{pmatrix} -\lambda & \lambda & & & \\ \mu & -(\mu+\lambda) & \lambda & & \\ & \mu & -(\mu+\lambda) & \lambda & \\ & & \mu & -(\mu+\lambda) & \lambda \\ & & & & \ddots \end{pmatrix}$$

Code :-

```
lambda = 2; % arrival rate
mu = 3; % service rate
buffer_sizes = 0:20; % vary buffer size from 0 to 20

blocking_probabilities = zeros(size(buffer_sizes)); % preallocate for efficiencyfor i =

1:length(buffer_sizes)
    buffer_size = buffer_sizes(i);if
    buffer_size == 0
        blocking_probabilities(i) = 1 - lambda/mu; % no bufferelse
        rho = lambda/mu;p0 =
        1 - rho; summation =
        0;
        for j = 0:buffer_size
            summation = summation + (rho^j)/factorial(j);end
        blocking_probabilities(i) = (rho^buffer_size)/(factorial(buffer_size)*p0*summation); % compute blockingprobability
    end
end
```

```
plot(buffer_sizes, blocking_probabilities, 'o-'); % plot blocking probability vs. buffer size xlabel('Buffer Size');
ylabel('Blocking Probability');
title(['M/M/1 Queue with Finite Buffer, \lambda = ', num2str(lambda), ', \mu = ', num2str(mu)]);
```

<u>Output</u> :-


M/M/1 Queue with Finite Buffer, λ = 2, μ = 3
Blocking Probability
Buffer Size