NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

DATA COMMUNICATION (CAECC12)

NAME- TANISHA MAJUMDAR

ROLL NO-2021UCA1874

SEMESTER- IV

BRANCH-CSAI

SECTION-1 (GROUP-2)

INDEX

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 number and plot its density function. Find its mean and variance.

5. Compute the Signal to quantization Noise ratio of Uniform Quantization. Plot SNQR versus Quantization levels.

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

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

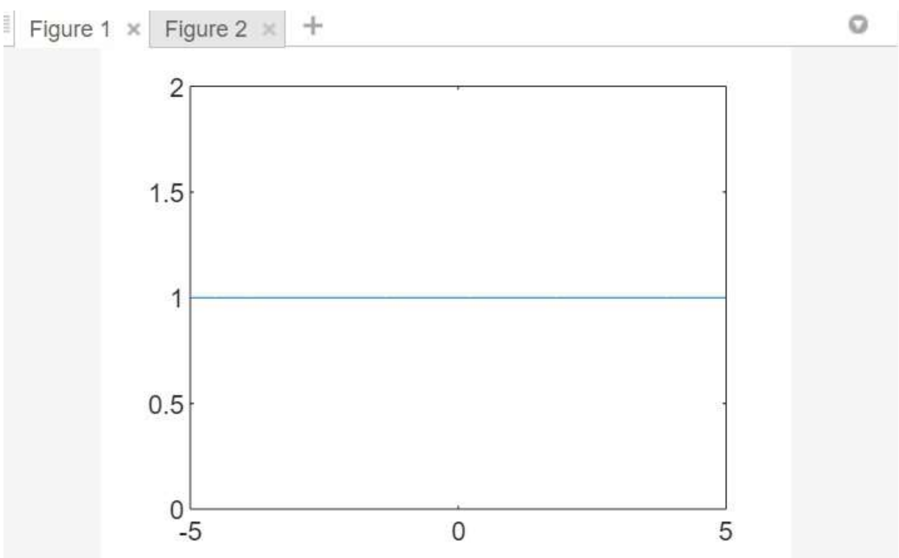
9.To generate a M/M/1 Queue having infinite buffer space with parameters (·, ·) and plot the average delay per packet vs ·/·.

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

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

/MATLAB Drive/exp1.m

```
clc;
clear all;
close all;
syms t w
x=1;
expw=exp(-1*1i*w*t);
z=int(x*expw,t,-5,5);
xlabel('t')
ylabel('x(t)')
figure(1);
fplot('1',[-5 5])
figure(2);
fplot(z)
```



2. **To verify following properties of Fourier Transform:**

   i. **Time Shifting**

/MATLAB Drive/exp2.m

```
clc;
clear all;
close all;

syms t w;
x = cos(t);
t0=2;
xt0=cos(t-t0);

Left = fourier(xt0,w)
X= fourier(x,w)
Right= exp(-j*w*t0)*X
```

Command Window

```
Left =

pi*(dirac(w - 1)*exp(-2i) + dirac(w + 1)*exp(2i))


X =

pi*(dirac(w - 1) + dirac(w + 1))


Right =

pi*exp(-w*2i)*(dirac(w - 1) + dirac(w + 1))

>>
```

ii. Frequency shifting

/MATLAB Drive/exp21.m

```
clc;
clear all;
close all;

syms t w
x=t^3;

w0=3;
to_output=exp(j*w0*t)*x;
Left=fourier(to_output,w)
```

Command Window

```
Left =

-pi*dirac(3, w - 3)*2i

>>
```

iii. Convolutional

```
% Define two signals t =
linspace(-2, 2, 1000); x = exp(-
t.^2 / 2); y = exp(-2*t.^2);

% Zero-pad one of the signals x_padded = [x,
zeros(1, length(y) - 1)]; y_padded = [y, zeros(1,
length(x) - 1)];

% Compute the convolution of the signals z =
conv(x_padded, y_padded);

% Compute the Fourier transforms of the signals
X  = fft(x_padded);
Y  = fft(y_padded);
Z  = fft(z);

% Plot the results
figure; subplot(3, 1,
1); plot(t, x);
title('Signal x');
 subplot(3, 1, 2);
plot(t, y); title('Signal
y');
 subplot(3, 1, 3);
plot(t, abs(Z));
title('Fourier Transform of the
Convolution of x and y');
```

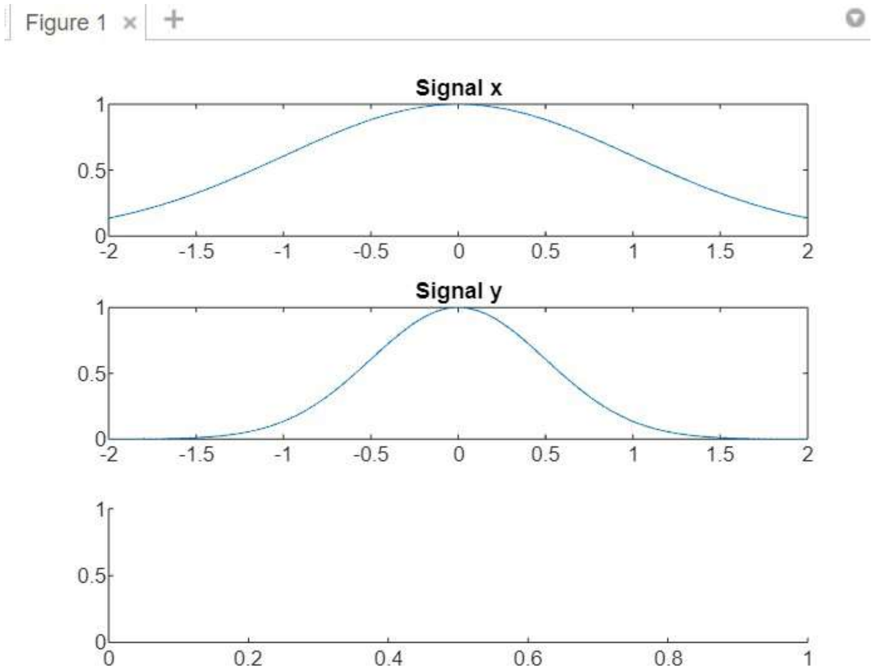
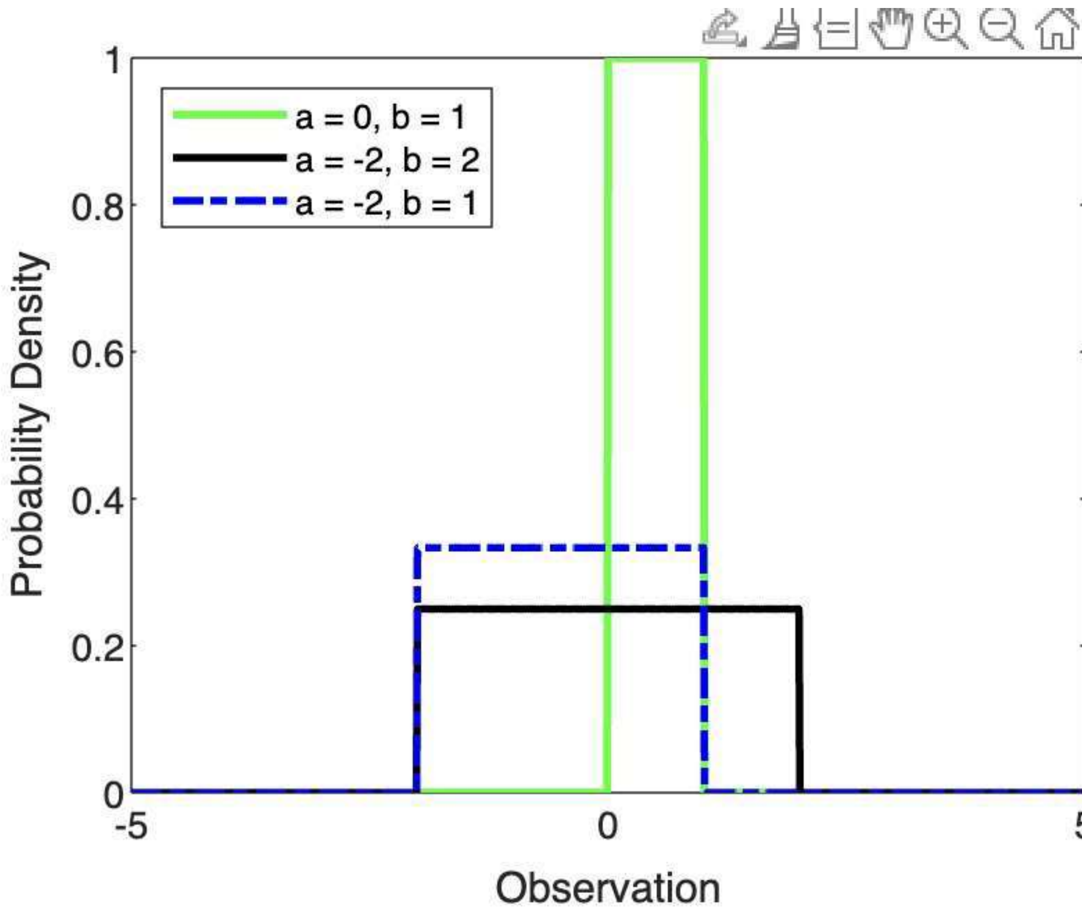


Figure 1
Signal x
Signal y

```
Time Domain Convolution:
     1     3     6    10     9     7     4

Frequency Domain Convolution:
    10    10    10    10

The Convolutional Property does not hold
>>
```

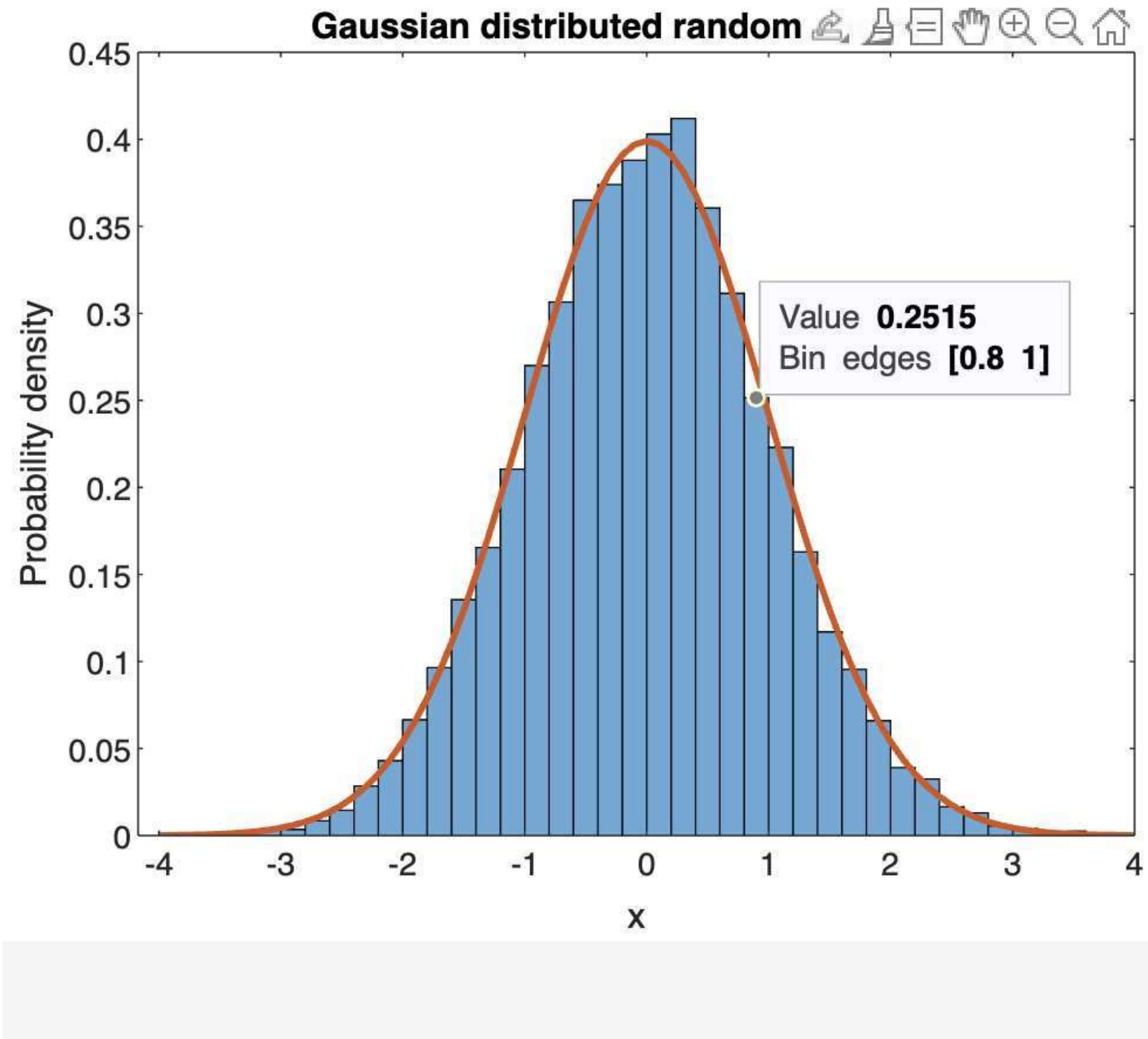
3. To generate Uniform random number and plot its density function. Find its mean and variance.

```
%generating uniform random number & find mean, variance
clear all;
 close all;
pd1 = makedist('Uniform');
pd2 = makedist('Uniform','lower',-2,'upper',2);
pd3 = makedist('Uniform','lower',-2,'upper',1);
 x = -5:.01:5;
pdf1 = pdf(pd1,x);
pdf2 = pdf(pd2,x);
pdf3 = pdf(pd3,x);
 figure;
plot(x,pdf1,'g','LineWidth',2);
hold on;
plot(x,pdf2,'k-','LineWidth',2);
plot(x,pdf3,'b-.','LineWidth',2);
legend({'a = 0, b = 1','a = -2, b = 2','a = -2, b = 1'},'Location','northwest');
xlabel('Observation')
ylabel('Probability Density')

hold off;
```



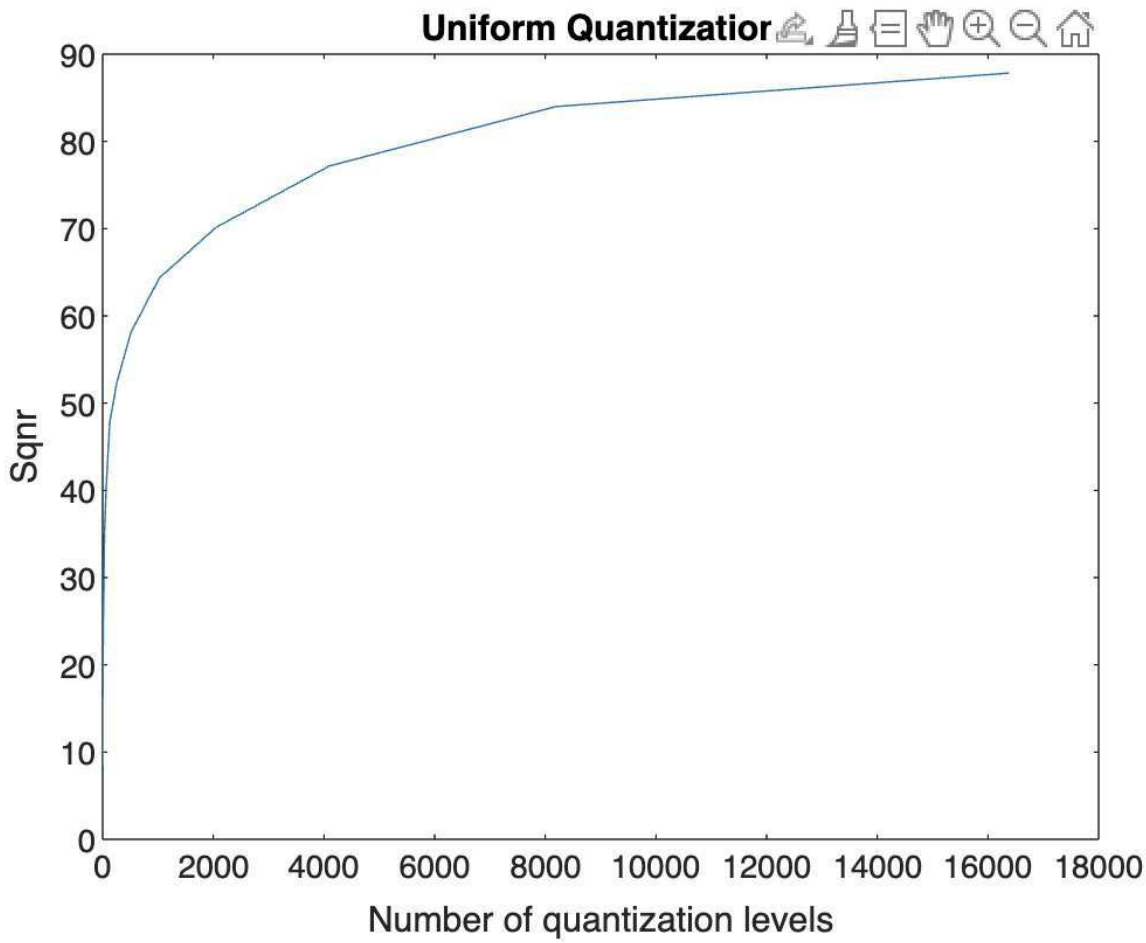
4. To generate Gaussian distributed random number and plot its density function. Find its mean and variance

```matlab
% Set the parameters for the Gaussian distribution
mu = 0;     % Mean of the distribution
sigma = 1;  % Standard deviation of the distribution

% Generate 10000 random numbers from the Gaussian distribution using randn
N = 10000;
X = mu + sigma * randn(N, 1);

% Plot the histogram of the generated random numbers
histogram(X, 'Normalization', 'pdf')

% Overlay the Gaussian density function on the histogram
hold on
x_range = -4:0.1:4;
y = normpdf(x_range, mu, sigma);
plot(x_range, y, 'LineWidth', 2)
hold off

% Add axis labels and title
xlabel('x')
ylabel('Probability density')
title('Gaussian distributed random numbers')
```



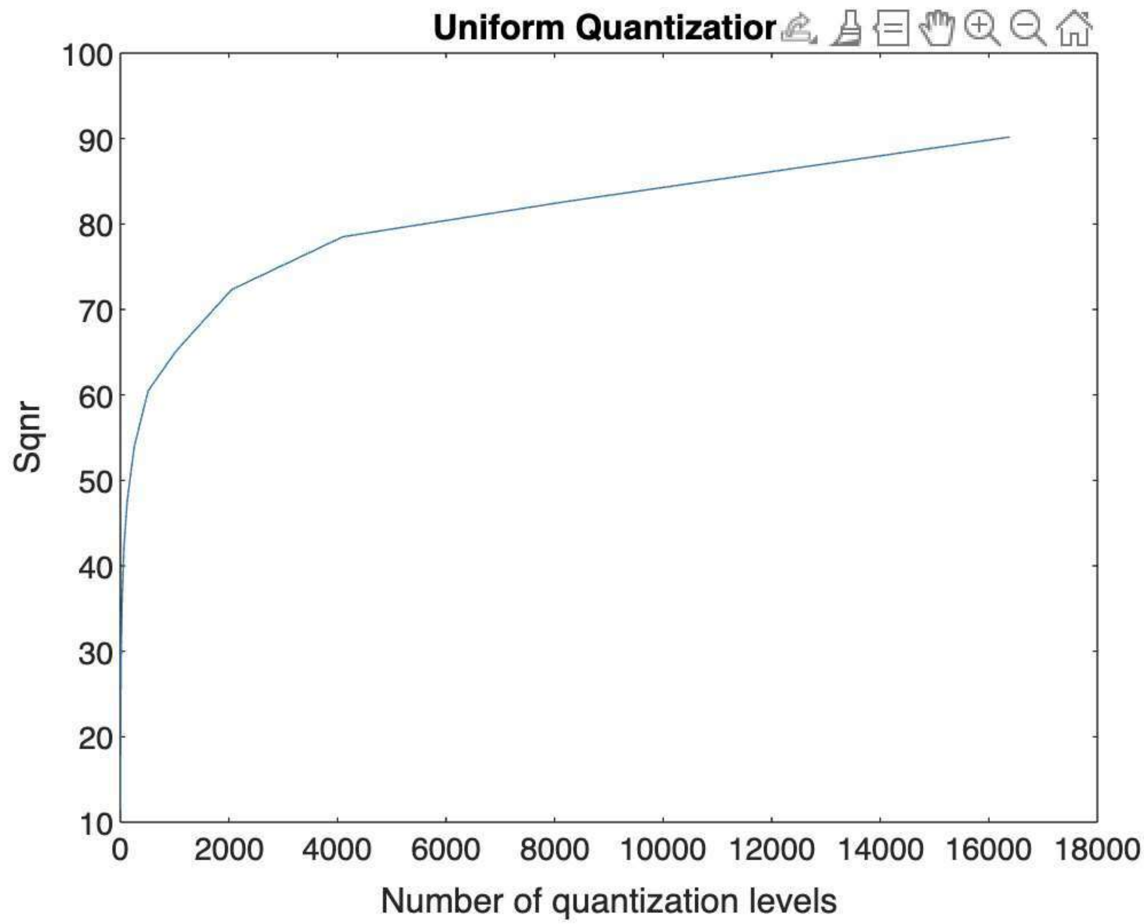
5. Compute the Signal to quantization Noise ratio of Uniform Quantization.Plot SNQR versus Quantization levels

```matlab
clc;clear all;close all;
n = 10; x = rand(1,n);vmin = min(x);vmax = max(x);xpow = sum(x.^2)/n;
for i=1:1:14
    L(i)=2^i;
    d=(vmax-vmin)/L(i);
    for j=1:length(x)
        start = min(x);
        while(start < x(j))
            start=start+d;
        end
        xq(j)=start-d;
        if(start == x(j))
            xq(j)=start;
        end
    end
    err=x-xq;
    noisepow(i)=sum(err.^2)/n;
end
sqnr=xpow./noisepow;
sqnrdb=10.*log10(sqnr);
plot(L,sqnrdb)
xlabel('Number of quantization levels');
ylabel('Sqnr');
title('Uniform Quantization')
```



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

```
clc;clear all;close all;
n = 10; x = rand(1,n);
u = 255;
xcomp = ((log(1+(abs(x)./max(x)).*u))./log(1+u)); %compressed sample
vmin = min(xcomp);vmax = max(xcomp);xpow = sum(xcomp.^2)/n;
for i=1:1:14
    L(i)=2^i;
    d=(vmax-vmin)/L(i);
    for j=1:length(xcomp)
        start = min(xcomp);
        while(start < xcomp(j))
            start=start+d;
        end
        xq(j)=start-d;
        if(start == x(j))
            xq(j)=start;
        end
    end
    err=xcomp-xq;
    noisepow(i)=sum(err.^2)/n;
end
sqnr=xpow./noisepow;
sqnrdb=10.*log10(sqnr);
plot(L,sqnrdb)
xlabel('Number of quantization levels');
ylabel('Sqnr');
title('Uniform Quantization')
```



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

```matlab
% Parameters
nBits = 1000000; % Number of bits in the binary sequence
EbNo_dB = 10; % Eb/No in dB
M = 2; % Number of modulation symbols (2 for BPSK)

% Generate random binary sequence
bitsTx = randi([0 1], 1, nBits);

% Modulate the binary sequence using BPSK
symbTx = 2*bitsTx - 1;

% Compute the energy per bit
Eb = sum(symbTx.^2)/nBits;

% Compute the noise power spectral density
No = Eb/10^(EbNo_dB/10);

% Generate AWGN
noise = sqrt(No/2)*randn(1, length(symbTx));

% Add noise to the modulated signal
symbRx = symbTx + noise;

% Demodulate the received signal using BPSK
bitsRx = (symbRx < 0);

% Calculate the bit error rate
ber = sum(bitsTx ~= bitsRx)/nBits;

% Display the results
fprintf('Eb/No = %f dB\n', EbNo_dB);
fprintf('Bit error rate = %e\n', ber);
```

```
Eb/No = 10.000000 dB
Bit error rate = 9.999970e-01
>> experiment7
Eb/No = 10.000000 dB
Bit error rate = 9.999950e-01
>> experiment7
Eb/No = 10.000000 dB
Bit error rate = 9.999930e-01
>>
```

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’.

```
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 sysytem matrix 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);
```

```
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 sysytem matrix 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

>>
```

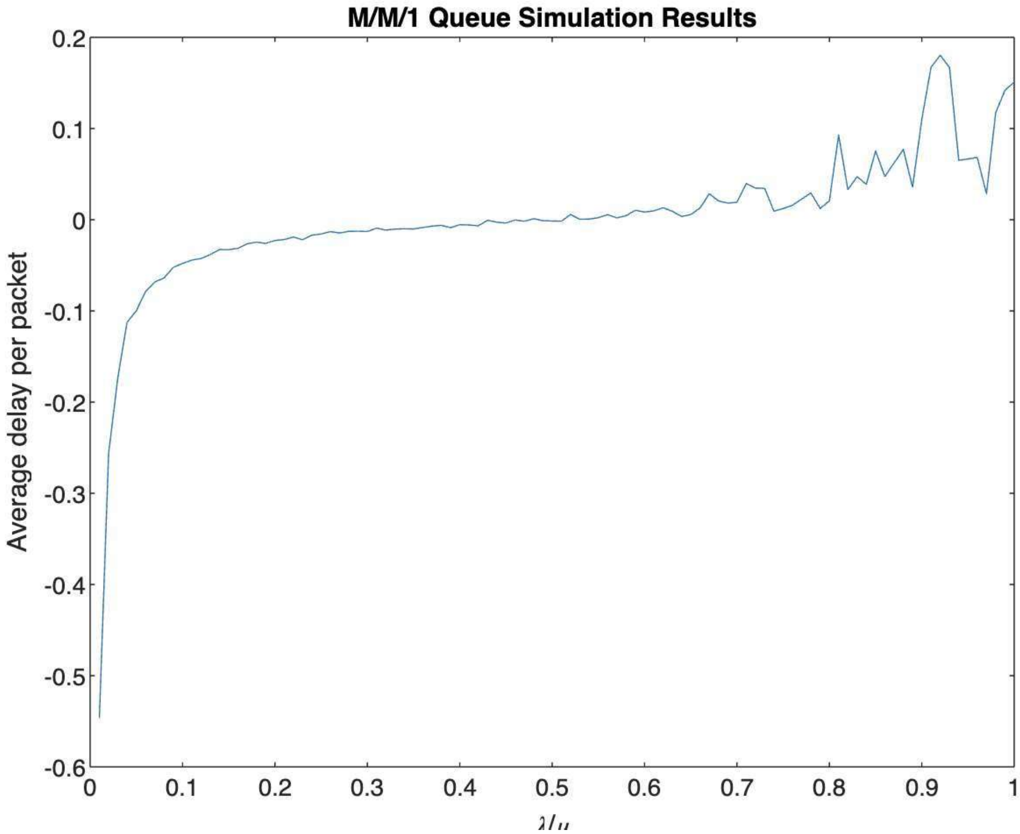
9. To generate a M/M/1 Queue having infinite buffer space with parameters (,) and plot the average delay per packet vs /.

AND

```
clc;
clear all;
close all;

% Define simulation parameters
queue_lim = 200000;          % system limit
sim_packets = 750;           % number of clients to be simulated
mu = 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 lambda
for i = 1:length(lambda_vals)

    % Define arrival rate for this simulation
    lambda = 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
```

```
% 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 queue
        if 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 queue
        if queue_size > 0
```

```
            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 queue
        if 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(lambda_vals/mu, avg_delay);
xlabel('\lambda/\mu');
ylabel('Average delay per packet');
title('M/M/1 Queue Simulation Results');
```



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

```
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 efficiency

for i = 1:length(buffer_sizes)
    buffer_size = buffer_sizes(i);
    if buffer_size == 0
        blocking_probabilities(i) = 1 - lambda/mu; % no buffer
    else
        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 blocking probability
    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)]);
```