INDEX

SI No.	Date	Name of the Experiment	Page No.	Marks Awarded	SIGNATURE

INDEX

SI No.	Date	Name of the Experiment	Page No.	Marks Awarded	SIGNATURE

Exp. No.: 1	QAM MODULATION SISO
Date:	

AIM:

To design and simulate QAM Modulation SISO using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Initialize Environment: Clear the command window and any previously stored variables.

2. Set Parameters: Define the number of data points (`N`), the size of the signal constellation (`mlevel`), and calculate the number of bits per symbol (`k`).

3. Generate Random Bits: Create a random binary bit stream of length `N`.

4. Form Symbols from Bits: Reshape the binary stream into `k`-bit groups and convert these groups to decimal symbols using binary-to-decimal conversion.

5. Modulate Using QAM: Apply Quadrature Amplitude Modulation (QAM) to the symbols to generate the modulated signal ready for transmission.

6. Add Noise: Introduce Additive White Gaussian Noise (AWGN) to the transmitted signal using a specified Signal-to-Noise Ratio (SNR).

7. Demodulate Signal: Demodulate the received noisy signal back into symbols using QAM demodulation.

8. Convert Symbols to Bits: Convert the demodulated symbols back into a binary bit stream.

9. Calculate Bit Error Rate (BER): Compare the transmitted and received bit streams to calculate the number of errors and the Bit Error Rate (BER).

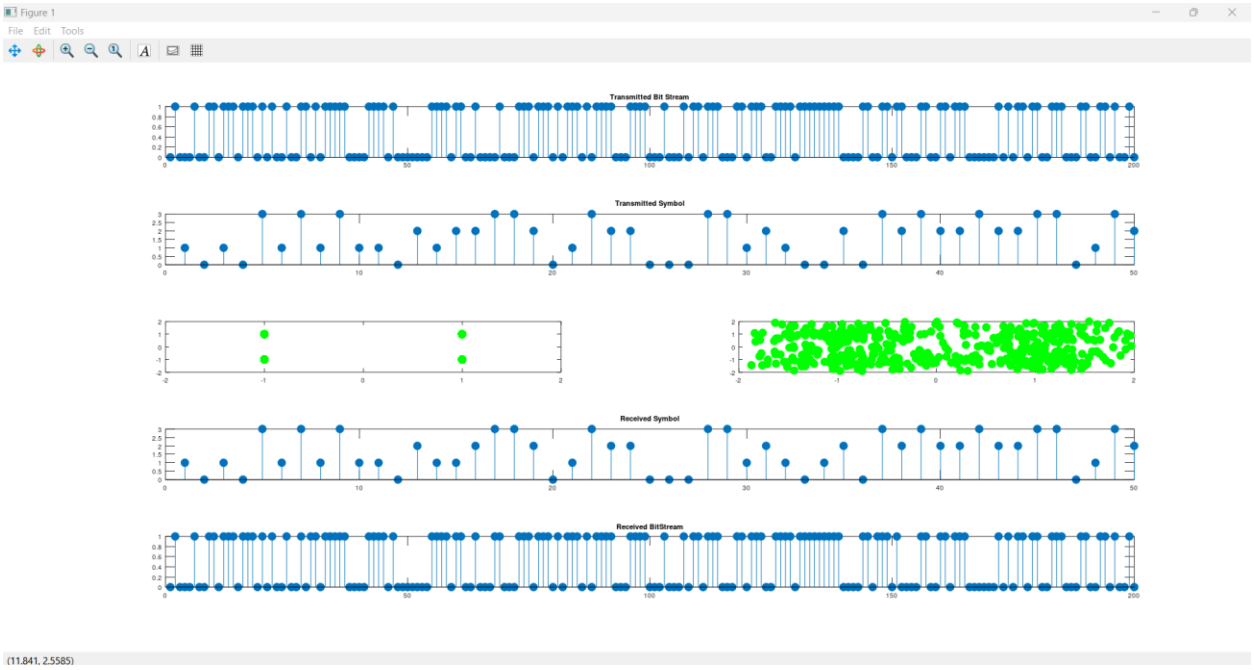
10. Visualization: Plot various stages of the modulation process including the transmitted bit stream, transmitted symbols, constellation diagram of the modulated and received signals, received symbols, and the received bit stream.

PROGRAM:

```
% Clear the command window and any variables in the workspace
clear
clc

% Define the number of data points
N = 1000; % number of data

% Define the size of the signal constellation
mlevel = 4; % size of signal constellation

% Calculate the number of bits per symbol
k = log2(mlevel); % number of bits per symbol

% Generate random binary bit stream of length N
x = randi([0 1], N, 1); % signal generation in bit stream

% Convert the bit stream into symbols using binary to decimal conversion
% Reshape the binary stream into a matrix with each column representing a symbol
% and each row representing a set of bits for each symbol
% 'left-msb' indicates that the most significant bit (MSB) is on the left side
xsym = bi2de(reshape(x, k, length(x)/k).', 'left-msb'); % convert the bit stream into symbol stream

% Modulate the symbols using Quadrature Amplitude Modulation (QAM)
xmod = qammod(xsym, mlevel); % modulation

% Store the modulated symbols in a variable for transmission
Tx_x = xmod;



% Define the Signal-to-Noise Ratio (SNR) in decibels
SNR = 5;

% Add Additive White Gaussian Noise (AWGN) to the transmitted signal
Tx_awgn = awgn(Tx_x, SNR, 'measured'); % adding AWGN

% Store the received signal after noise addition
Rx_x = Tx_awgn; % Received signal

% Demodulate the received signal to recover the symbols
```

```
Rx_x_demod = qamdemod(Rx_x, mlevel); % demodulation

% Convert the demodulated symbols back to binary bits
z = de2bi(Rx_x_demod, 'left-msb'); % Convert integers to bits.

% Convert the matrix of bits back to a vector
Rx_x_BitStream = reshape(z.', prod(size(z)), 1); % Convert z from a matrix to a vector.

% Calculate the Bit Error Rate (BER) by comparing the transmitted and received bits
[number_of_errors, bit_error_rate] = biterr(x, Rx_x_BitStream); % Calculate BER

% Display BER
disp(['Number of Errors: ', num2str(number_of_errors)]);
disp(['Bit Error Rate (BER): ', num2str(bit_error_rate)]);

% Plot each step of the process
subplot(5,2,[1 2]); stem(x(1:200),'filled'); title('Transmitted Bit Stream');
subplot(5,2,[3 4]); stem(xsym(1:50),'filled'); title('Transmitted Symbol');
subplot(5,2,5); plot(real(Tx_x), imag(Tx_x), 'go', 'MarkerFaceColor', [0, 1, 0]);
                axis([-mlevel/2 mlevel/2 -mlevel/2 mlevel/2]);
subplot(5,2,6); plot(real(Rx_x), imag(Rx_x), 'go', 'MarkerFaceColor', [0, 1, 0]);
                axis([-mlevel/2 mlevel/2 -mlevel/2 mlevel/2]);
subplot(5,2,[7 8]); stem(Rx_x_demod(1:50),'filled'); title('Received Symbol');
subplot(5,2,[9 10]); stem(Rx_x_BitStream(1:200),'filled'); title('Received BitStream');
```

OUTPUT:



RESULT:

Thus the design and simulation of QAM Modulation SISO using octave has been executed successfully and output was verified.

Exp. No.: 2	QAM MIMO BER
Date:	

AIM:

To design and simulate QAM MIMO BER using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Initialize Environment:
   - Clear workspace variables.
   - Define simulation parameters: the number of bits `N`, range of (E_b/N_0) values in dB, number of transmitters (`nTx`), and number of receivers (`nRx`).
2. Loop Over (E_b/N_0) Values
3. Generate Random Bit Sequence:
   - Create a random sequence of bits (0s and 1s) of length `N`.
4. BPSK Modulation:
   - Modulate the bit sequence using BPSK, where 0 maps to -1 and 1 maps to +1.
5. Prepare for MIMO Transmission:
   - Repeat the modulated signal to match the number of receivers, and reshape for MIMO transmission.
6. Simulate Rayleigh Fading Channel:
   - Generate a Rayleigh fading channel matrix `h` for each transmitter-receiver pair for each symbol.
7. Add White Gaussian Noise:
   - Generate complex Gaussian noise scaled according to the (E_b/N_0) value.
8. Channel and Noise Impact:
   - Apply the channel effects and noise to the transmitted signal.
9. MMSE Equalization at the Receiver:
   - Calculate the MMSE equalization matrix for each symbol using the channel matrix `h` and noise variance derived from \(E_b/N_0\).
   - Equalize the received signal using the calculated MMSE matrix to estimate the transmitted symbols.
10. Hard Decision Decoding:
    - Make decisions on the estimated symbols to recover the transmitted bits. A positive real part in the estimated complex symbol decodes to 1; otherwise, it decodes to 0.
11. Error Counting:
    - Compare the decoded bit sequence with the original transmitted bits to count the number of errors.

12. BER Calculation:
- Calculate the BER for each \(E_b/N_0\) value as the ratio of the number of errors to the total number of transmitted bits.

13. Theoretical BER Calculation:
- Compute theoretical BER for comparison:
  - For a single receiver (nRx=1).
  - For two receivers using Maximal Ratio Combining (MRC) in a diversity scenario.

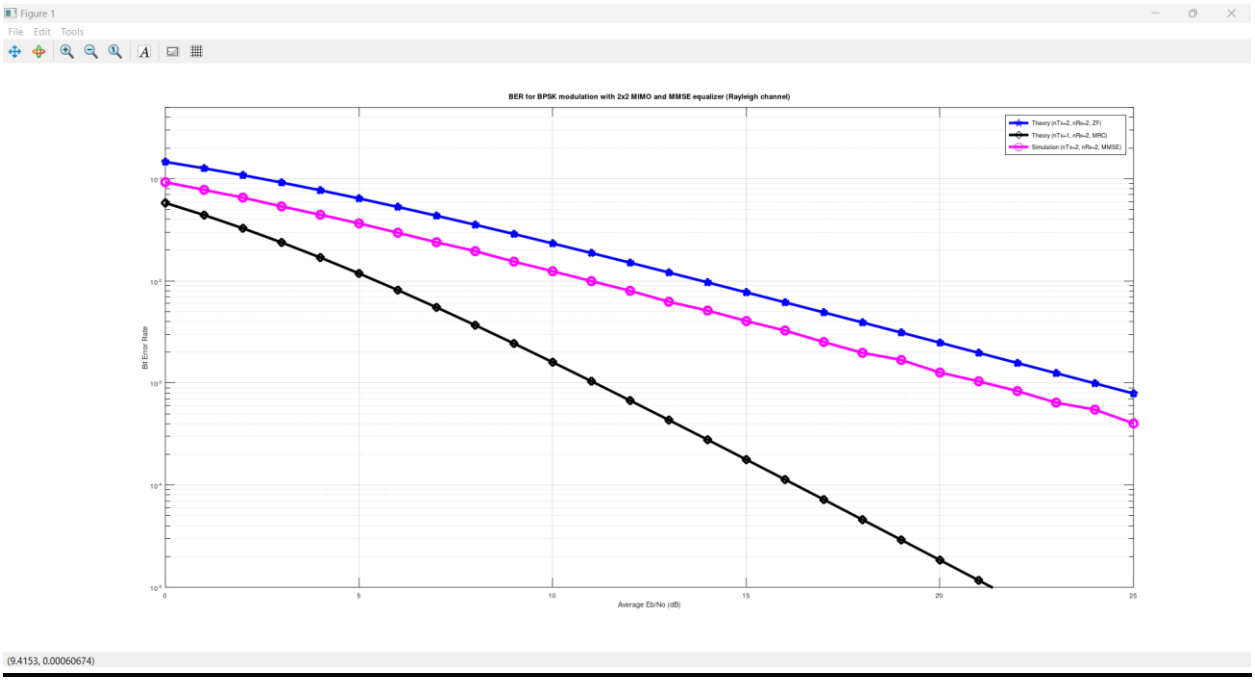
PROGRAM:

```
% Script for computing the Bit Error Rate (BER) for Binary Phase Shift Keying (BPSK) modulation in a
% Rayleigh fading channel with 2 Transmitters (Tx) and 2 Receivers (Rx) MIMO channel,
% using Minimum Mean Square Error (MMSE) equalization.

clear; % Clear workspace variables

N = 10^6; % Number of bits or symbols
Eb_N0_dB = [0:25]; % Multiple Eb/N0 values in dB
nTx = 2; % Number of transmitters
nRx = 2; % Number of receivers

for ii = 1:length(Eb_N0_dB)
    % Transmitter

    % Generate random bit sequence for transmission
    ip = rand(1, N) > 0.5; % Generating 0s and 1s with equal probability
    s = 2 * ip - 1; % BPSK modulation: 0 -> -1; 1 -> 0

    % Modulate the symbols for MIMO transmission
    sMod = kron(s, ones(nRx, 1));
    sMod = reshape(sMod, [nRx, nTx, N / nTx]);

    % Generate Rayleigh fading channel
    h = 1/sqrt(2) * [randn(nRx, nTx, N / nTx) + j * randn(nRx, nTx, N / nTx)];

    % Generate white Gaussian noise with 0dB variance
    n = 1/sqrt(2) * [randn(nRx, N / nTx) + j * randn(nRx, N / nTx)];

    % Channel and noise addition
    y = squeeze(sum(h .* sMod, 2)) + 10^(-Eb_N0_dB(ii) / 20) * n;

    % Receiver
```

```
 % Compute the MMSE equalization matrix: W = inv(H^H*H + sigma^2*I)*H^H
    hCof = zeros(2, 2, N / nTx);
    hCof(1, 1, :) = sum(h(:, 2, :) .* conj(h(:, 2, :)), 1) + 10^(-Eb_N0_dB(ii) / 10);
    hCof(2, 2, :) = sum(h(:, 1, :) .* conj(h(:, 1, :)), 1) + 10^(-Eb_N0_dB(ii) / 10);
    hCof(2, 1, :) = -sum(h(:, 2, :) .* conj(h(:, 1, :)), 1);
    hCof(1, 2, :) = -sum(h(:, 1, :) .* conj(h(:, 2, :)), 1);
    hDen = ((hCof(1, 1, :) .* hCof(2, 2, :)) - (hCof(1, 2, :) .* hCof(2, 1, :)));
    hDen = reshape(kron(reshape(hDen, 1, N / nTx), ones(2, 2)), 2, 2, N / nTx);
    hInv = hCof ./ hDen;

    hMod = reshape(conj(h), nRx, N);

    yMod = kron(y, ones(1, 2));
    yMod = sum(hMod .* yMod, 1);
    yMod = kron(reshape(yMod, 2, N / nTx), ones(1, 2));
    yHat = sum(reshape(hInv, 2, N) .* yMod, 1);

    % Receiver - hard decision decoding
    ipHat = real(yHat) > 0;

    % Counting the errors
    nErr(ii) = size(find([ip - ipHat]), 2);

end

% Compute simulated BER
simBer = nErr / N;

% Compute theoretical BER for nRx=1 and nRx=2
EbN0Lin = 10 .^ (Eb_N0_dB / 10);
theoryBer_nRx1 = 0.5 * (1 - (1 + 1 ./ EbN0Lin) .^ (-0.5));
p = 1 / 2 - 1 / 2 * (1 + 1 ./ EbN0Lin) .^ (-1 / 2);
theoryBerMRC_nRx2 = p .^ 2 .* (1 + 2 * (1 - p));

% Plot BER curves
close all;
figure;
semilogy(Eb_N0_dB, theoryBer_nRx1, 'bp-', 'LineWidth', 2);
hold on;
semilogy(Eb_N0_dB, theoryBerMRC_nRx2, 'kd-', 'LineWidth', 2);
semilogy(Eb_N0_dB, simBer, 'mo-', 'LineWidth', 2);
axis([0 25 10^-5 0.5]);
grid on;
legend('Theory (nTx=2, nRx=2, ZF)', 'Theory (nTx=1, nRx=2, MRC)', 'Simulation (nTx=2,
```

```
nRx=2, MMSE)');
xlabel('Average Eb/No (dB)');
ylabel('Bit Error Rate');
title('BER for BPSK modulation with 2x2 MIMO and MMSE equalizer (Rayleigh channel)');
```

OUTPUT:



RESULT:

Thus the design and simulation of QAM MIMO BER using octave has been executed successfully and output was verified.

Exp. No.: 3	OFDM WAVE FORM GENERATION
Date:	

AIM:

To design and simulate OFDM Wave Form Generation using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Environment Initialization:
   - Clear the MATLAB/Octave command window and workspace variables.
   - Load necessary communications packages for operations related to signal processing.
2. Parameter Definition:
   - Define the FFT size (e.g., 128) to set the number of points for the Fast Fourier Transform (FFT).
   - Specify the indices for the subcarriers used to transmit data (typically excluding the DC component).
   - Define the total number of bits to be transmitted and set the number of bits per symbol (e.g., 1 for BPSK).
   - Compute the total number of symbols required based on the number of bits and the number of bits per symbol.
3. Data Generation and Modulation:
   - Generate a random binary sequence of the defined bit length.
   - Modulate this binary sequence using BPSK (Binary Phase Shift Keying), where logical '0' maps to -1 and '1' maps to +1.
   - If necessary, pad the modulated sequence to match the total number of symbols times bits per symbol.
4. OFDM Symbol Preparation:
   - Reshape the padded, modulated bit sequence into symbols.
   - Initialize an empty vector to store the OFDM symbols.
5. IFFT Processing and Cyclic Prefix Addition:
   - For each symbol, initialize a zeros vector of length equal to the FFT size.
   - Place the modulated data onto the specified subcarrier positions.
   - Apply `fftshift` to align the subcarriers appropriately within the vector.
   - Perform the Inverse Fast Fourier Transform (IFFT) to convert the frequency domain data back to the time domain.
   - Add a cyclic prefix to each time-domain OFDM symbol to mitigate inter-symbol interference.
6. Concatenation and Signal Preparation:
   - Concatenate all OFDM symbols, each with its cyclic prefix, into a single time-domain signal vector.

7. Power Spectral Density Calculation:
- Close all previously opened figure windows.
- Set the sampling frequency and any other parameters necessary for spectral analysis (e.g., subcarrier spacing and total bandwidth).
- Use Welch's method (`pwelch`) to estimate the power spectral density of the OFDM signal, specifying the number of points for spectral analysis.

8. PSD Plotting:
- Plot the power spectral density against frequency, adjusting the frequency axis based on the sampling frequency and the number of FFT points used in the PSD calculation.
- Label the axes and title the plot to reflect the content (e.g., "Transmit Spectrum of OFDM").

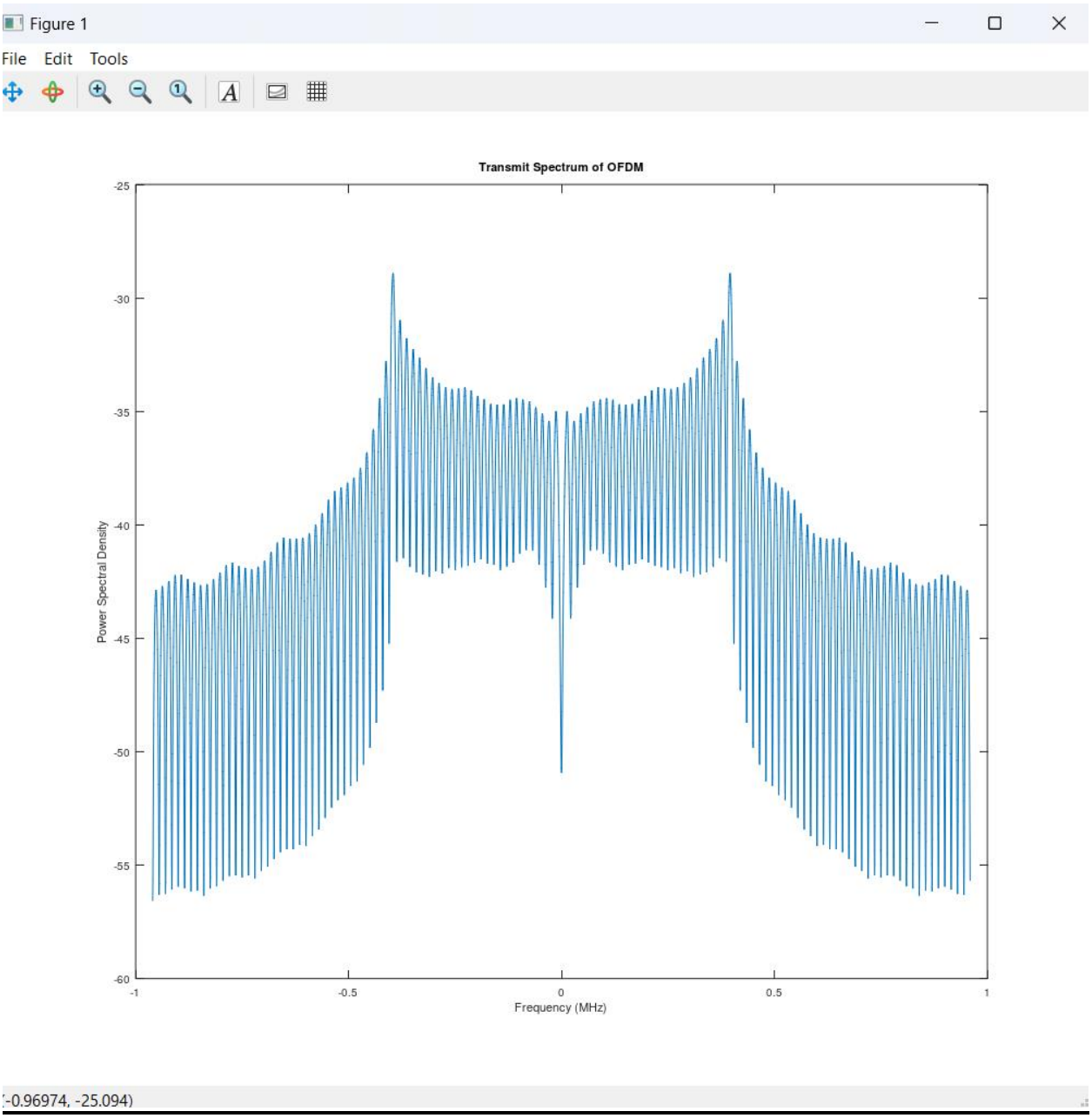
PROGRAM:

```
% Clear the command window and any variables in the workspace
clc
clear

% Load the communications package
pkg load communications

% Define the size of the FFT (Fast Fourier Transform)
nFFTSize = 128;

% Define the subcarrier index range for each symbol
% The indices range from -26 to -1 and 1 to 26
subcarrierIndex = [-26:-1 1:26];

% Define the number of bits for transmission
nBit = 2500;

% Generate a random binary sequence of length nBit
ip = rand(1, nBit) > 0.5; % generating 1's and 0's

% Define the number of bits per symbol
nBitPerSymbol = 1;

% Calculate the number of symbols required to transmit nBit
nSymbol = ceil(nBit / nBitPerSymbol);

% Modulate the binary sequence using Binary Phase Shift Keying (BPSK)
% Convert 0's to -1 and 1's to +1
ipMod = 2 * ip - 1;
```

```
% Pad the modulated symbols with zeros to match the required length
ipMod = [ipMod zeros(1, nBitPerSymbol * nSymbol - nBit)];

% Reshape the modulated symbols into a matrix with each row representing a symbol
% and each column representing bits for each symbol
ipMod = reshape(ipMod, nSymbol, nBitPerSymbol);

% Initialize an empty vector to store the OFDM symbols
st = [];

% Iterate over each symbol
for ii = 1:nSymbol

    % Initialize an array to store the input for the Inverse Fast Fourier Transform (IFFT)
    inputiFFT = zeros(1, nFFTSize);

    % Assign the bits from the modulated symbols to subcarriers
    inputiFFT(subcarrierIndex + nFFTSize / 2 + 1) = ipMod(ii, :);

    % Shift the subcarriers at indices [-26 to -1] to fft input indices [38 to 63]
    inputiFFT = fftshift(inputiFFT);

    % Perform IFFT to convert frequency domain symbols to time domain
    outputiFFT = ifft(inputiFFT, nFFTSize);

    % Add cyclic prefix of 16 samples to the output
    outputiFFT_with_CP = [outputiFFT(49:64) outputiFFT];

    % Concatenate the OFDM symbol with cyclic prefix to the vector st
    st = [st outputiFFT_with_CP];

end

% Close all open figures
close all

% Define the subcarrier spacing and bandwidth
% 64 subcarrier , with 15KHz subcarrier spacing equals 960KHZ Bandwidth
fsMHz = 1.92; % 960 KHz
nOverlap = 20; % Overlap for pwelch function
```

```
% Compute the power spectral density using the Welch method
[Pxx, W] = pwelch(st, [], [], 4096, nOverlap);

% Plot the power spectral density
plot([-2048:2047] * fsMHz / 4096, 10 * log10(fftshift(Pxx)));
xlabel('Frequency (MHz)')
ylabel('Power Spectral Density')
title('Transmit Spectrum of OFDM');
```

OUTPUT:



RESULT:

Thus the design and simulation of OFDM Wave Form Generation using octave has been executed successfully and output was verified.

Exp. No.: 4	OFDM BER
Date:	

AIM:

To design and simulate OFDM BER using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Clearing the workspace and importing necessary packages. This step ensures that all previous variables are erased and that the required functions for communication systems are available.
2. Function Definitions:
   - Guard Interval Functions: The code includes functions to add and remove guard intervals (either cyclic prefix or zero padding) to OFDM symbols. These intervals help in reducing ISI and enhancing system performance in multipath environments.
   - BER Calculation Functions: There are functions for computing the Bit Error Rate (BER) for AWGN and Rayleigh fading scenarios, essential for performance evaluation.
   - Q Function and Modulation It defines the Q function (used in computing error probabilities) and sets up functions for modulation and demodulation processes.
3. Parameter Configuration:
   - The script configures key parameters like the type of channel (AWGN or multipath), guard interval type, modulation scheme, FFT size, and the number of virtual carriers. These parameters dictate the structure of the OFDM signal and its resilience against channel impairments.
4. Simulation Setup:
   - The main loop of the simulation varies the SNR (Signal to Noise Ratio) to evaluate the BER under different noise conditions. It generates OFDM symbols, applies the modulation, and simulates channel effects followed by demodulation and BER computation.
5. File Output:
   - BER results are stored in a data file, enabling offline analysis or graphical representation.
6. Plotting:
   - Finally, the script plots BER against Eb/N0 (the energy per bit to noise power spectral density ratio), providing a visual representation of system performance under specified conditions.

PROGRAM:

```
% Simulate the effect of ISI as the length of a guard interval(CP, CS, or ZP)
% It considers the BER performance of an OFDM system with 64-point FFT and
% 16 virtual carriers for 16-QAM signaling in the AWGN or multipath
% Rayleigh fading channel(with the maximum delay of 15 samples).
clear,close,clc all

% Load the communications package
pkg load communications

% Function to add guard interval (GI) to OFDM symbols
function y = guard_interval(Ng,Nfft,NgType,ofdmSym)
if NgType == 1 % CP
  y = [ofdmSym(Nfft-Ng+1:Nfft) ; ofdmSym(1:Nfft)];

elseif NgType == 2 % ZP
   y = [zeros(Ng,1) ; ofdmSym(1:Nfft)];
endif
endfunction

% Function to remove guard interval (GI) from OFDM symbols
function y = remove_GI(Ng,Lsym,NgType,ofdmSym)
if Ng ~= 0
   if NgType == 1 % CP
      y = ofdmSym(Ng+1:Lsym);
   elseif NgType == 2 % CS
      y = ofdmSym(1:Lsym-Ng) + [ofdmSym(Lsym-Ng+1:Lsym) zeros(1,Lsym-2*Ng)];
   endif
else
   y = ofdmSym;
endif
endfunction

% Function to remove cyclic prefix (CP) from OFDM symbols
function y = remove_CP(x,Ncp,Noff)
if nargin < 3
   Noff = 0;
endif
y = x(:,Ncp+1-Noff:end-Noff);
endfunction

% Function to compute Q function
function y = Q(x)
y = erfc(x/sqrt(2))/2;
endfunction
```

```
% Function to plot Bit Error Rate (BER)
function plot_ber(file_name,Nbps)
EbN0dB = [0:1:10];
M = 2^Nbps;
ber_AWGN = ber_QAM(EbN0dB,M,'AWGN');
ber_Rayleigh = ber_QAM(EbN0dB,M,'Rayleigh');

semilogy(EbN0dB,ber_AWGN,'r:');
hold on;
semilogy(EbN0dB,ber_Rayleigh,'r-');
a = load(file_name);
semilogy(a(:,1),a(:,2),'b--s');
grid on
title("BER plot of Rayleigh fading channel with OFDM");
legend('AWGN analytic','Rayleigh fading analytic','Simulation');
xlabel('EbN0[dB]'); ylabel('BER with 16 QAM OFDM with Rayleigh fading'); axis([a(1,1)
a(end,1) 1e-5 1])
endfunction

% Function to compute Bit Error Rate (BER) for QAM modulation
function ber = ber_QAM(EbN0dB,M,AWGN_or_Rayleigh)
N = length(EbN0dB);
sqM = sqrt(M);
a = 2*(1-power(sqM,-1))/log2(sqM);
b = 6*log2(sqM)/(M-1);
if nargin < 3
   AWGN_or_Rayleigh = 'AWGN';
endif
if lower(AWGN_or_Rayleigh(1)) == 'a'
   ber = a*Q(sqrt(b*10.^(EbN0dB/10)));
else
   rn = b*10.^(EbN0dB/10)/2;
   ber = 0.5*a*(1-sqrt(rn./(rn+1)));
endif
endfunction


% Set guard interval type (CP or ZP)

NgType = 2; % NgType=1/2 for cyclic prefix/zero padding
if NgType == 1
   nt = 'CP';
elseif NgType == 2
   nt = 'ZP';
end
```

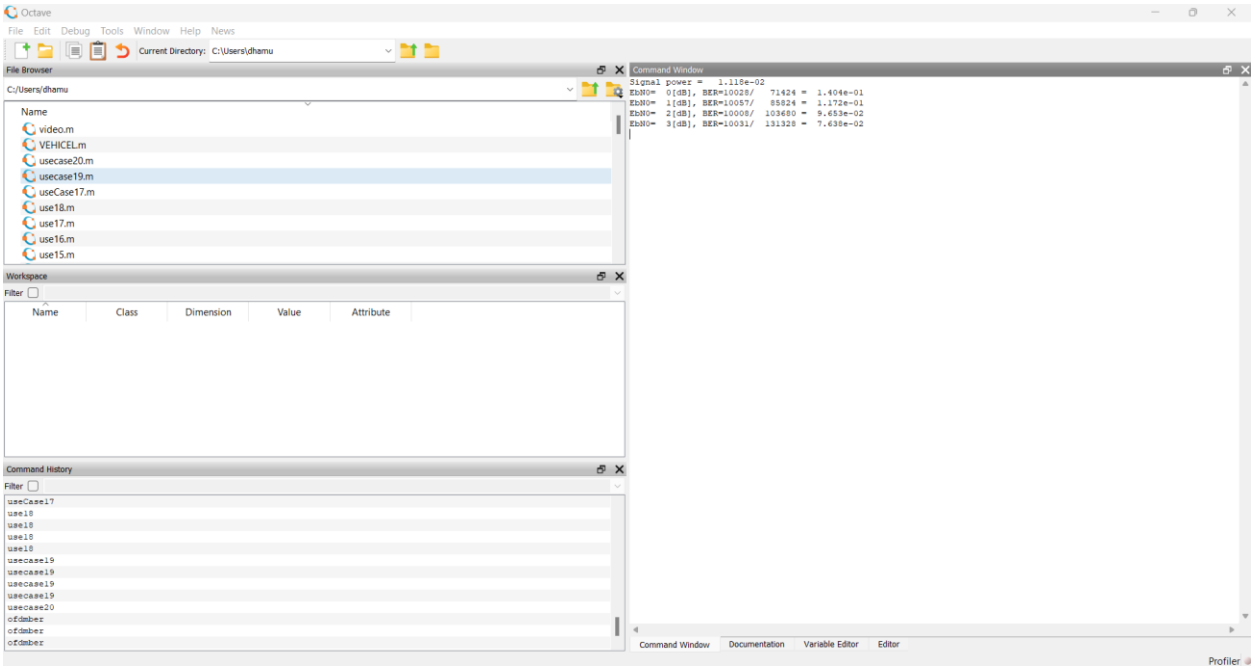
```
% Set channel type (AWGN or multipath)

Ch=0; % Ch=0/1 for AWGN/multipath channel
#Ch=1; % Ch=0/1 for AWGN/multipath channel
if Ch == 0
   chType = 'AWGN';
   Target_neb = 10000;
else
   chType = 'CH';
   Target_neb = 50000;
end
%figure(Ch+1), clf

% Define power and delay profiles for multipath channel
PowerdB = [0 -8 -17 -21 -25]; % Channel tap power profile 'dB'
Delay = [0 3 5 6 8];          % Channel delay 'sample'
Power = 10.^(PowerdB/10);     % Channel tap power profile 'linear scale'
Ntap = length(PowerdB);       % Chanel tap number
Lch = Delay(end)+1;           % Channel length

% Define modulation order (2 for QPSK, 4 for 16-QAM, 6 for 64-QAM)
Nbps = 4;
M = 2^Nbps; % Modulation order=2/4/6 for QPSK/16QAM/64QAM

% Define FFT size and number of virtual carriers
Nfft = 64;  % FFT size
Ng = 3;%Nfft/4;  % Ng=0: Guard interval length
% Ng=Nfft/4;
Nsym = Nfft + Ng;      % Symbol duration
Nvc = Nfft/4;
#Nvc=0;      % Nvc=0: no virtual carrier 16
Nused = Nfft-Nvc;    % 48 = 64 - 16

% Other parameters
No_of_frames=10;  % Number of frames
EbN0 = [0:1:10];    % EbN0 in dB
N_iter = 1e3;       % Number of iterations for each EbN0
Nframe = 3;         % Number of symbols per frame
sigPow = 0;         % Signal power initialization

file_name = ['OFDM_BER_' chType '_' nt '_' 'GL' num2str(Ng) '.dat']; %
OFDM_BER_AWGN_CP_GL16
fid = fopen(file_name, 'w+');
norms = [1 sqrt(2) 0 sqrt(10) 0 sqrt(42)];     % BPSK 4-QAM 16-QAM

for i = 0:length(EbN0)
```

```
randn('state',0); rand('state',0); %rand
Ber = 0; % BER initialization
Neb = 0; Ntb = 0; % Initialize the number of error/total bits
for m = 1:N_iter
   % Tx
   % Generate random bit sequence for transmission
   X = randint(1,Nused*Nframe,M); % bit: integer vector 1 x 48*3, M=16
   Xmod = qammod(X,M)/norms(Nbps); % qammod(X,M,0,'gray')/norms(Nbps);

   % Add guard interval (GI) to OFDM symbols
   if NgType ~= 2
      x_GI = zeros(1,Nframe*Nsym); % 3*80
   elseif NgType == 2
      x_GI = zeros(1,Nframe*Nsym+Ng);
   % Extend an OFDM symbol by Ng zeros
  end

% Generate OFDM symbols
% Extend each OFDM symbol with a guard interval (GI)
% Apply modulation and normalization

   kk1 = [1:Nused/2]; % Nused = 48
   kk2 = [Nused/2+1:Nused];
   kk3 = 1:Nfft; % 64
   kk4 = 1:Nsym; % 80
   for k = 1:Nframe
      if Nvc ~= 0 % 16
         X_shift = [0;Xmod(kk2) ; zeros(Nvc-1,1);  Xmod(kk1)];
      else
         X_shift = [Xmod(kk2) Xmod(kk1)];
      end
      x = ifft(X_shift);
      x_GI(kk4) = guard_interval(Ng,Nfft,NgType,x);
      kk1 = kk1 + Nused;
      kk2 = kk2 + Nused;
      kk3 = kk3 + Nfft;
      kk4 = kk4 + Nsym;
   end

   %Channel
   if Ch == 0 % AWGN
      y=x_GI; % No channel
   else
      % Generate Multipath fading channel
      channel = (randn(1,Ntap)+j*randn(1,Ntap)).*sqrt(Power/2);
```

```
    h = zeros(1,Lch);
    h(Delay+1) = channel; % Channel Impulse Response
    y = conv(x_GI,h);
endif
if i == 0 % Only to measure the signal power for adding AWGN noise
    y1 = y(1:Nframe*Nsym);
    sigPow = sigPow + y1*y1';
    continue;
endif


% Add AWGN noise
snr = EbN0(i) + 10*log10(Nbps*(Nused/Nfft)); % SNR vs. Eb/N0
noise_mag = sqrt((10.^(-snr/10))*sigPow/2); % N0=Eb/SNR
y_GI = y + noise_mag*(randn(size(y)) + j*randn(size(y)));

% Rx
kk1 = (NgType == 2)*Ng + [1:Nsym];
kk2 = 1:Nfft;
kk3 = 1:Nused;
kk4 = Nused/2 + Nvc + 1:Nfft;
kk5 = (Nvc~=0) + [1:Nused/2];
if Ch == 1
    H = fft([h zeros(1,Nfft-Lch)]); % Channel frequency response
    H_shift(kk3) = [H(kk4) H(kk5)];
end
for k = 1:Nframe
    Y(kk2) = fft(remove_GI(Ng,Nsym,NgType,y_GI(kk1)));
    Y_shift = [Y(kk4) Y(kk5)];
    if Ch == 0
        Xmod_r(kk3) = Y_shift;
    else
        Xmod_r(kk3) = Y_shift./H_shift;  % Equalizer - channel compensation
    end
    kk1 = kk1 + Nsym;
    kk2 = kk2 + Nfft;
    kk3 = kk3 + Nused;
    kk4 = kk4 + Nfft;
    kk5 = kk5 + Nfft;
end

% Demodulate received symbols
X_r = qamdemod(Xmod_r*norms(Nbps),M);

% Compute number of bit errors and total bits
```

```
      Neb = Neb + sum(sum(de2bi(X_r,Nbps)~=de2bi(X,Nbps)));
       Ntb = Ntb + Nused*Nframe*Nbps;  %[Ber,Neb,Ntb]=ber(bit_Rx,bit,Nbps);
       if Neb > Target_neb
          break;
       end
    end


    % Write BER data to file
    if i == 0
       sigPow = sigPow/Nsym/Nframe/N_iter;
       fprintf('Signal power = %11.3e\n', sigPow);
       fprintf(fid,'%%Signal power= %11.3e\n%%EbN0[dB]       BER\n', sigPow);
    else
       Ber = Neb/Ntb;
       fprintf('EbN0=%3d[dB], BER=%4d/%8d =%11.3e\n', EbN0(i), Neb, Ntb, Ber)
       fprintf(fid, '%d\t%11.3e\n', EbN0(i), Ber);
       if Ber < 1e-6
          break;
       end
    end
end

% Close file
if (fid ~= 0)
   fclose(fid);
endif

% Display completion message
disp('Simulation is finished');

% Plot BER
plot_ber(file_name,Nbps);
```

OUTPUT:



RESULT:

Thus the design and simulation of OFDM BER using octave has been executed successfully and output was verified.

Exp. No.: 5	eMBB OPTIMIZATION WORKSHOP
Date:	

AIM:

To design and simulate eMBB Optimization Workshop using octave.

APPARATUS REQUIRED:

Octave Simulation

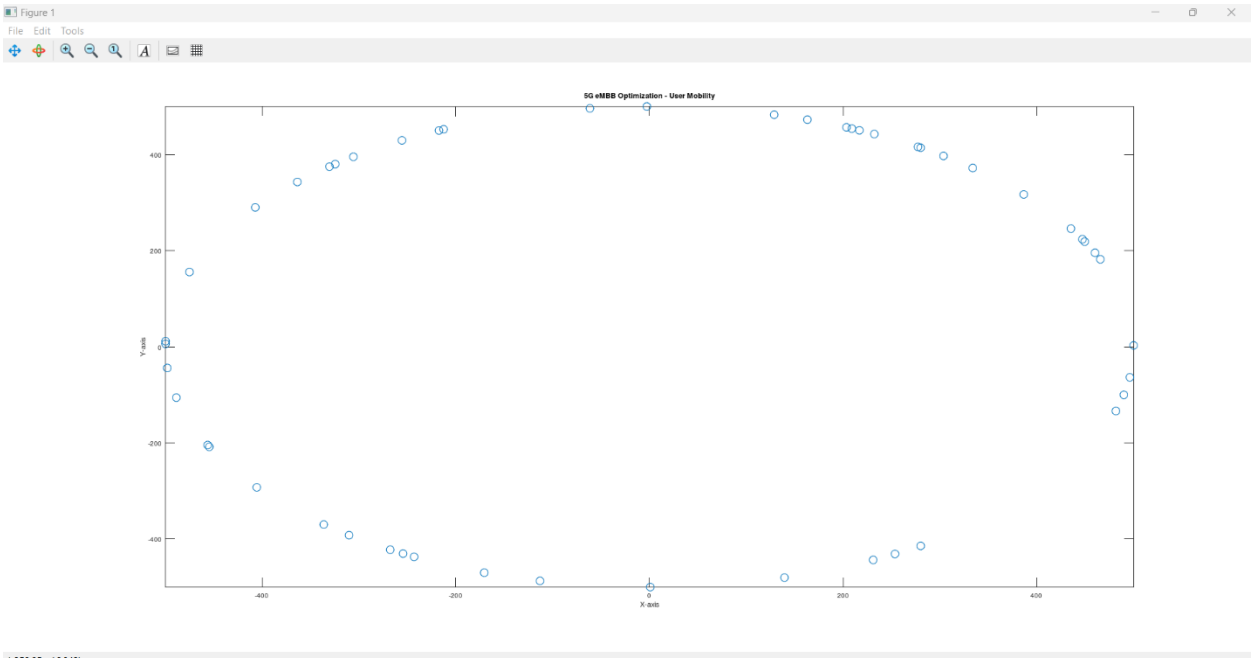
ALGORITHM:

1. Parameter Initialization:
   - Initialize parameters such as the number of mobile users, cell radius, user speed, simulation duration, time step, and antenna configuration parameters.
2. Random User Placement:
   - Randomly place users within the circular cell coverage by generating random positions using the exponential function and the `rand` function.
3. User Mobility Simulation:
   - Simulate the mobility of users over time by updating their positions based on their speed and direction. Use a loop to iterate over the simulation duration, updating user positions at each time step.
4. Plotting:
   - Plot the positions of users on a 2D graph to visualize their mobility within the cell. Set appropriate axis labels, limits, and a title for clarity.
5. Antenna Configuration and Bandwidth Optimization:
   - Placeholder section for adjusting antenna configurations and optimizing bandwidth allocation.
   - Perform adjustments such as modifying antenna tilt angle, beamwidth, or implementing dynamic bandwidth allocation algorithms based on current network conditions.
   - Further development and testing can be conducted in this section to optimize the network's performance for eMBB applications.

PROGRAM:

```
% Parameters
num_users = 50; % Number of mobile users
cell_radius = 500; % Radius of the cell (assuming circular cell coverage)
user_speed = 30; % Average user speed in km/h
simulation_duration = 60; % Simulation duration in seconds
time_step = 1; % Time step for simulation in seconds

% Antenna configuration parameters
antenna_height = 25; % Height of the antenna in meters
antenna_tilt = 5; % Antenna tilt angle in degrees
beamwidth = 120; % Antenna beamwidth in degrees

% Bandwidth optimization parameters
initial_bandwidth = 100e6; % Initial bandwidth in Hz
% Additional parameters for dynamic bandwidth allocation algorithm can be added here

% Randomly place users within the cell
user_positions = cell_radius * exp(2i * pi * rand(1, num_users));

% Simulate user mobility over time
for t = 0:time_step:simulation_duration
    % Update user positions based on their speed and direction
    user_positions = user_positions + user_speed * (time_step / 3600) * exp(2i * pi * rand(1, num_users));

    % Plot user positions
    plot(real(user_positions), imag(user_positions), 'o');
    title('5G eMBB Optimization - User Mobility');
    xlabel('X-axis');
    ylabel('Y-axis');
    xlim([-cell_radius, cell_radius]);
    ylim([-cell_radius, cell_radius]);
    drawnow;
```

OUTPUT:



RESULT:

Thus the design and simulation eMBB Optimization Workshop of using octave has been executed successfully and output was verified.

Exp. No.: 6	URLLC IMPLEMENTATION CHALLENGE
Date:	

AIM:

To design and simulate URLLC Implementation Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1.Parameter Definition:
- Define parameters such as the number of IoT devices, latency threshold for URLLC, reliability threshold, number of network slices, slice names, and QoS values for each slice.

2.Latency and Reliability Simulation:
- Simulate latency and reliability for each device:
  - Generate random latency values between 1 and 10 milliseconds for each device.
  - Generate random reliability values between 0 and 1 for each device.

3.Slice Assignment:
- Determine slice assignments for each device based on QoS requirements:
  - Check if device latency meets URLLC requirements.
  - If latency meets URLLC requirements, assign the device to the URLLC slice.
  - If not, assign the device to the slice with the highest QoS value among eMBB and mMTC.

4. Display Slice Assignments:
- Display the slice assignments for each device, showing which slice each device is assigned to.

5. QoS Impact Analysis:
- Analyze the impact of QoS settings on latency and reliability:
  - Calculate the average latency and reliability across all devices.

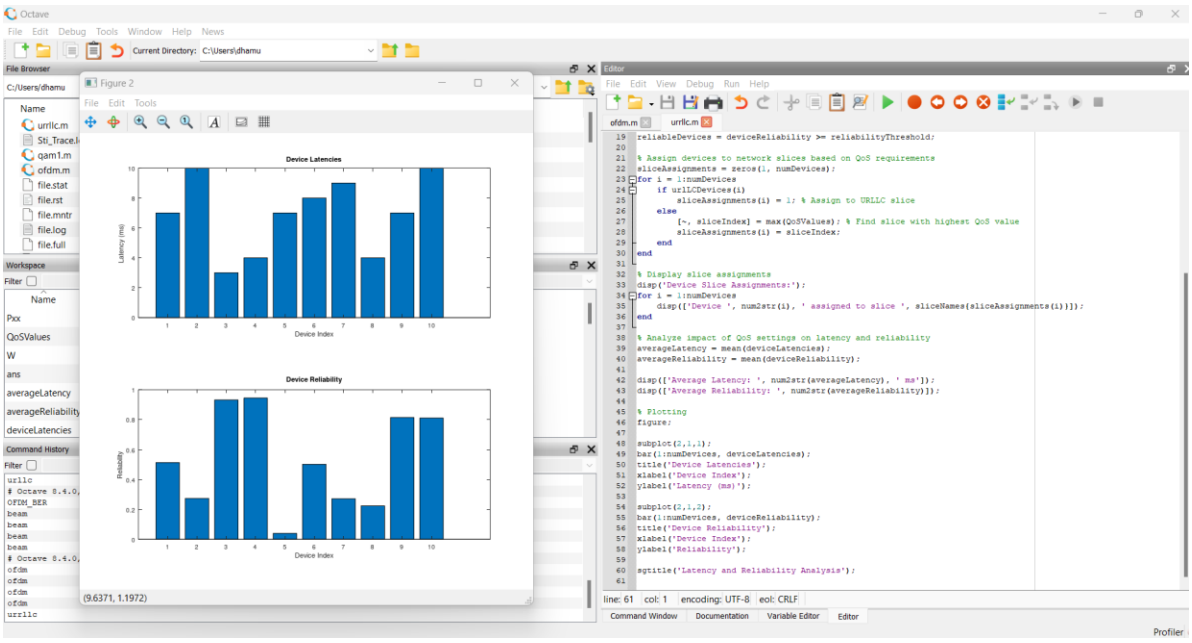
6. Plotting:
- Visualize latency and reliability distributions:
  - Plot device latencies and reliabilities using bar charts.
  - Display separate plots for latency and reliability.
  - Include appropriate axis labels, titles, and a title for the entire figure.

PROGRAM:

```
% Define parameters
numDevices = 10; % Number of IoT devices
latencyThreshold = 5; % Latency threshold in milliseconds for URLLC
reliabilityThreshold = 0.95; % Required reliability
numSlices = 3; % Number of network slices
sliceNames = {'URLLC', 'eMBB', 'mMTC'}; % Slice names
QoSValues = [10, 5, 3]; % QoS values for each slice

% Simulate latency for each device
deviceLatencies = randi([1, 10], 1, numDevices); % Random latency between 1 and 10 ms

% Check if latency meets URLLC requirements
urlLCDevices = deviceLatencies <= latencyThreshold;

% Simulate reliability for each device
deviceReliability = rand(1, numDevices); % Random reliability between 0 and 1

% Check if reliability meets requirements
reliableDevices = deviceReliability >= reliabilityThreshold;

% Assign devices to network slices based on QoS requirements
sliceAssignments = zeros(1, numDevices);
for i = 1:numDevices
   if urlLCDevices(i)
      sliceAssignments(i) = 1; % Assign to URLLC slice
   else
      [~, sliceIndex] = max(QoSValues); % Find slice with highest QoS value
      sliceAssignments(i) = sliceIndex;
   end
end

% Display slice assignments
disp('Device Slice Assignments:');
for i = 1:numDevices
   disp(['Device ', num2str(i), ' assigned to slice ', sliceNames{sliceAssignments(i)}]);
end

% Analyze impact of QoS settings on latency and reliability
averageLatency = mean(deviceLatencies);
averageReliability = mean(deviceReliability);

disp(['Average Latency: ', num2str(averageLatency), ' ms']);
disp(['Average Reliability: ', num2str(averageReliability)]);
```

```
% Plotting
figure;

subplot(2,1,1);
bar(1:numDevices, deviceLatencies);
title('Device Latencies');
xlabel('Device Index');
ylabel('Latency (ms)');

subplot(2,1,2);
bar(1:numDevices, deviceReliability);
title('Device Reliability');
xlabel('Device Index');
ylabel('Reliability');

sgtitle('Latency and Reliability Analysis');
```

OUTPUT:



RESULT:

Thus the design and simulation URLLC Implementation Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 7	mMTC DEPLOYMENT CHALLENGE FOR SMART CITIES
Date:	

AIM:

To design and simulate mMTC Deployment Challenge for Smart Cities using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

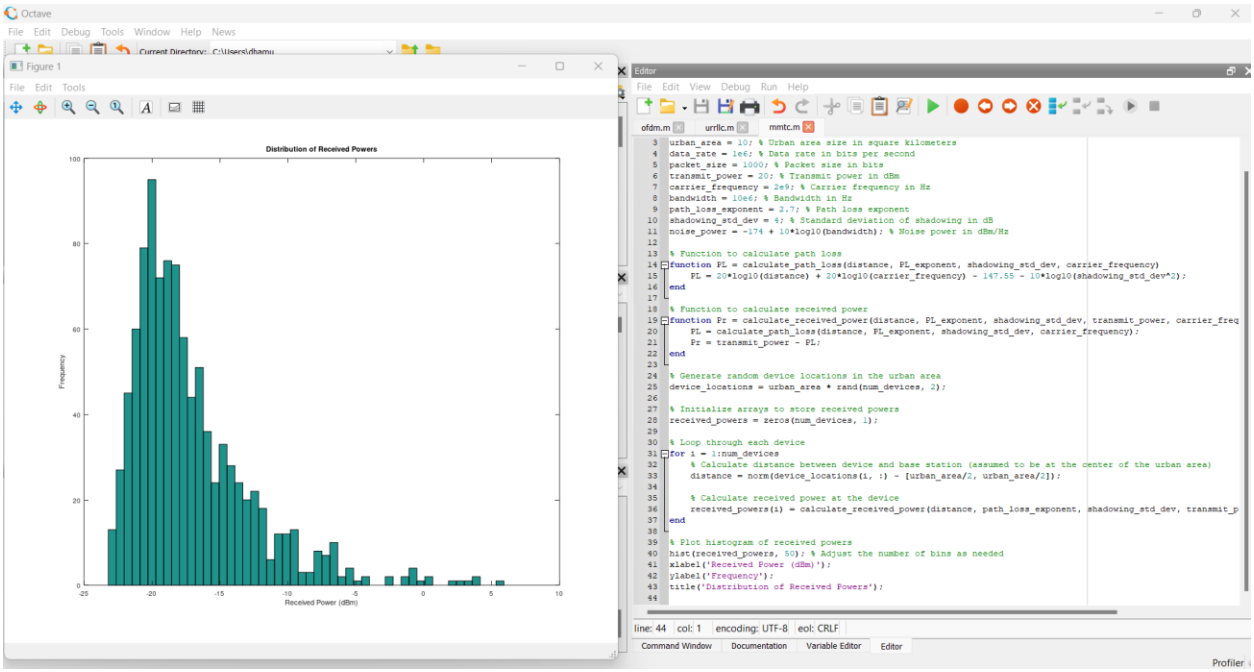
1. Parameter Definition:
   - Define parameters such as the number of IoT devices, urban area size, data rate, packet size, transmit power, carrier frequency, bandwidth, path loss exponent, shadowing standard deviation, noise power, scalability factor, maximum number of devices, sleep duration, required data rate for QoS, encryption algorithm, key length, and security protocol.
2. Function Definitions:
   - Define functions to calculate path loss, received power, and achievable data rate:
     - `calculate_path_loss`: Calculates the path loss based on distance, path loss exponent, shadowing standard deviation, and carrier frequency.
     -`calculate_received_power`: Calculates the received power based on path loss, transmit power, and carrier frequency.
     - `calculate_data_rate`: Calculates the achievable data rate based on received power, noise power, and bandwidth.
   - Define a function `simulate_communication` to simulate communication in an urban area:
     - Generate random device locations within the urban area.
     - Loop through each device:
       - Calculate the distance between the device and the base station.
       - Calculate the received power at the device.
       - Calculate the achievable data rate.
     - Calculate the average achievable data rate across all devices.
     - Display the average achievable data rate.
3. Simulation:
   - Call the `simulate_communication` function with the defined parameters to simulate communication in the urban area and analyze the average achievable data rate.

PROGRAM:

```
% Define parameters
num_devices = 1000; % Number of IoT devices
urban_area = 10; % Urban area size in square kilometers
data_rate = 1e6; % Data rate in bits per second
packet_size = 1000; % Packet size in bits
transmit_power = 20; % Transmit power in dBm
carrier_frequency = 2e9; % Carrier frequency in Hz
bandwidth = 10e6; % Bandwidth in Hz
path_loss_exponent = 2.7; % Path loss exponent
shadowing_std_dev = 4; % Standard deviation of shadowing in dB
noise_power = -174 + 10*log10(bandwidth); % Noise power in dBm/Hz

% Parameters for scalability testing
scalability_factor = 1; % Factor to scale the number of devices
max_devices = 5000; % Maximum number of devices to test scalability

% Parameters for energy efficiency optimization
sleep_duration = 0.1; % Duration of sleep mode in seconds

% Parameters for QoS configuration
required_data_rate = 1e6; % Required data rate for QoS in bits per second

% Parameters for security implementation
encryption_algorithm = 'AES'; % Encryption algorithm
key_length = 128; % Key length in bits
security_protocol = 'TLS'; % Security protocol

% Function to calculate path loss
function PL = calculate_path_loss(distance, PL_exponent, shadowing_std_dev, carrier_frequency)
    PL = 20*log10(distance) + 20*log10(carrier_frequency) - 147.55 - 10*log10(shadowing_std_dev^2);
end

% Function to calculate received power
function Pr = calculate_received_power(distance, PL_exponent, shadowing_std_dev, transmit_power, carrier_frequency)
    PL = calculate_path_loss(distance, PL_exponent, shadowing_std_dev, carrier_frequency);
    Pr = transmit_power - PL;
end

% Function to calculate achievable data rate
```

```
function data_rate = calculate_data_rate(received_power, noise_power, bandwidth)
    data_rate = bandwidth * log2(1 + 10^(received_power/10)/10^(noise_power/10));
end

% Function to simulate communication in urban area
function simulate_communication(num_devices, urban_area, path_loss_exponent, shadowing_std_dev, transmit_power, noise_power, carrier_frequency, bandwidth)
    % Generate random device locations in the urban area
    device_locations = urban_area * rand(num_devices, 2);

    % Initialize arrays to store results
    received_powers = zeros(num_devices, 1);
    data_rates = zeros(num_devices, 1);

    % Loop through each device
    for i = 1:num_devices
        % Calculate distance between device and base station (assumed to be at the center of the urban area)
        distance = norm(device_locations(i, :) - [urban_area/2, urban_area/2]);

        % Calculate received power at the device
        received_powers(i) = calculate_received_power(distance, path_loss_exponent, shadowing_std_dev, transmit_power, carrier_frequency);

        % Calculate achievable data rate
        data_rates(i) = calculate_data_rate(received_powers(i), noise_power, bandwidth);
    end

    % Calculate average data rate
    avg_data_rate = mean(data_rates);

    % Display results
    disp(['Average achievable data rate: ', num2str(avg_data_rate/1e6), ' Mbps']);
end

% Call the function to simulate communication
simulate_communication(num_devices, urban_area, path_loss_exponent, shadowing_std_dev, transmit_power, noise_power, carrier_frequency, bandwidth);
```

OUTPUT:



RESULT:

Thus the design and simulation mMTC Deployment Challenge for Smart Cities of using octave has been executed successfully and output was verified.

Exp. No.: 8	5G NETWORK SLICING WORKSHOP
Date:	

AIM:

To design and simulate NETWORK SLICING Workshop using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

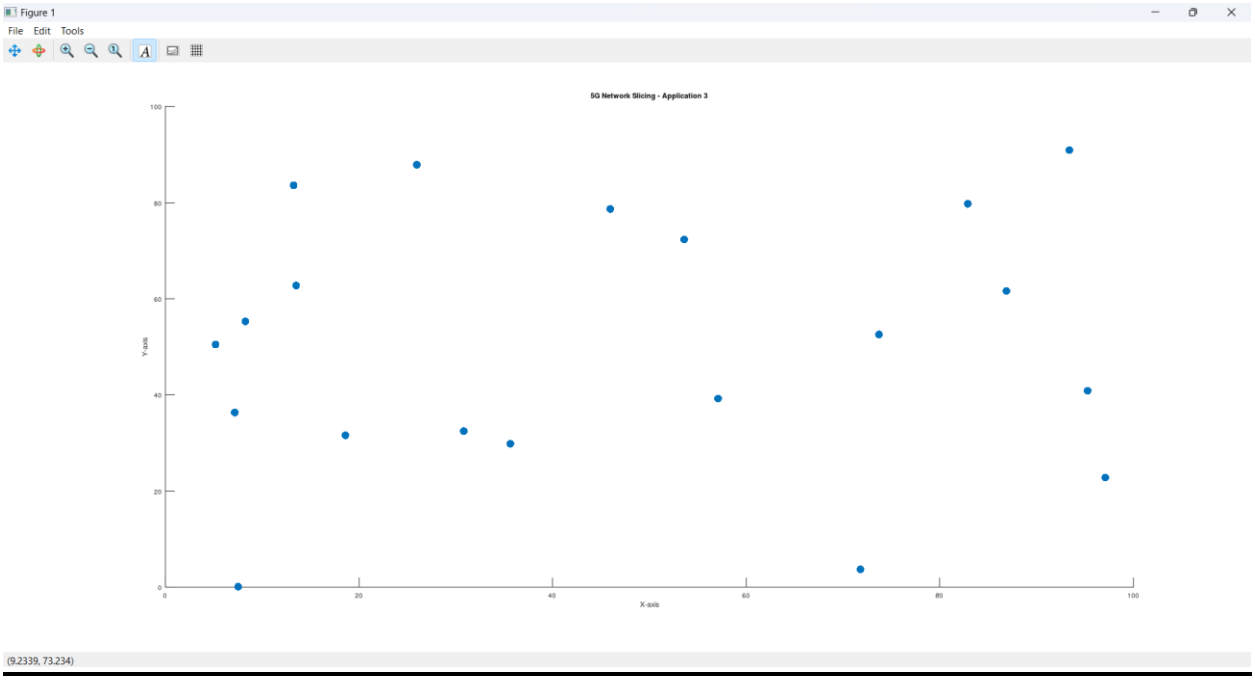
1. Parameter Definition:
   - Define parameters such as the number of diverse applications (`num_applications`) and the number of users per network slice (`num_users_per_slice`).
2. Network Slicing Simulation:
   - Generate random characteristics for each network slice associated with diverse applications:
     - `slice_bandwidths`: Generate random bandwidth values ranging from 50 Mbps to 200 Mbps for each application.
     - `slice_latency_targets`: Generate random latency target values ranging from 5 ms to 20 ms for each application.
   - Iterate over each application to simulate network slices:
     - Generate random user positions within a predefined area for each slice.
     - Display slice information including bandwidth and latency targets.
     - Visualize user positions within the slice using a scatter plot.
     - Provide appropriate titles, labels, and axis limits for visualization.

PROGRAM:

```
% Parameters
num_applications = 3; % Number of diverse applications
num_users_per_slice = 20; % Number of users per network slice

% Generate random network slice characteristics for each application
slice_bandwidths = randi([50, 200], 1, num_applications); % Bandwidth in Mbps
slice_latency_targets = randi([5, 20], 1, num_applications); % Latency targets in milliseconds

% Simulate network slices for each application
for app = 1:num_applications
    % Generate random user positions within a predefined area for each slice
    user_positions = 100 * rand(num_users_per_slice, 2);

    % Display slice information
```

```
    disp(['Application ', num2str(app)]);
    disp(['Slice Bandwidth: ', num2str(slice_bandwidths(app)), ' Mbps']);
    disp(['Slice Latency Target: ', num2str(slice_latency_targets(app)), ' ms']);

    % Visualize user positions within the slice
    scatter(user_positions(:, 1), user_positions(:, 2), 'filled');
    title(['5G Network Slicing - Application ', num2str(app)]);
    xlabel('X-axis');
    ylabel('Y-axis');
    xlim([0, 100]);
    ylim([0, 100]);
    drawnow;
end
```

OUTPUT:



RESULT:

Thus the design and simulation NETWORK SLICING Workshop of using octave has been executed successfully and output was verified.

Exp. No.: 9	MEC-AR INTEGRATION CHALLENGE
Date:	

AIM:

To design and simulate MEC-AR Integration Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

Step 1: Function Definitions

- Define functions for initializing connection with the MEC server, offloading AR computation to the server, receiving processed data from the server, rendering the AR scene, and the main function to orchestrate the process.
- Each function performs a specific step in the process, including connecting to the MEC server, offloading computation, receiving processed data, and rendering the AR scene.
- Additionally, include a helper function to generate AR data for demonstration purposes.

Initialize Connection with MEC Server:

- Call the `mec_init` function with the MEC server address to initialize the connection.

Step 2: Offload AR Computation to MEC Server:

- Call the `offload_computation` function with the AR data and the MEC server address to offload computation.

Step 3: Receive Processed Data from MEC Server:

- Call the `receive_processed_data` function with the MEC server address to receive processed data.

Step 4: Render AR Scene:

- Call the `render_ar_scene` function with processed data, temperature, date & time, and location information to render the AR scene.

Step 5: Main Function Execution:

- Call the `main` function with the MEC server address to orchestrate the entire process.
- This function internally calls the initialization, offloading, receiving, and rendering functions in sequence.

PROGRAM:

```
% Step 1: Initialize connection with MEC server
function mec_init(mec_server_address)
    % Code to initialize connection with MEC server
    disp(['Initializing connection with MEC server at ' mec_server_address]);
end

% Step 2: Offload AR computation to MEC server
function offload_computation(data, mec_server_address)
    % Code to offload computation to MEC server
    disp('Offloading computation to MEC server...');
    % Simulating offloading process
    pause(2); % Placeholder for actual offloading process
    disp('Computation offloaded successfully.');
end

% Step 3: Receive processed data from MEC server
function processed_data = receive_processed_data(mec_server_address)
    % Code to receive processed data from MEC server
    disp('Receiving processed data from MEC server...');
    % Simulating receiving processed data
    processed_data = rand(640, 480); % Placeholder data, replace with actual processed data
    disp('Processed data received successfully.');
end

% Step 4: Render AR scene
function render_ar_scene(data, temperature, datetime_str, location_str)
    % Code to render AR scene using processed data, temperature, date, time, and location
information
    disp('Rendering AR scene...');

    % Create a white background image
    white_background = ones(size(data));

    % Display the AR scene with white background
    figure;
    imshow(white_background); % Set the background color to white

    % Display information
    text(20, 20, ['Temperature: ' num2str(temperature) '°C'], 'Color', 'red', 'FontSize', 14);
    text(20, 40, ['Date & Time: ' datetime_str], 'Color', 'red', 'FontSize', 14);
    text(20, 60, ['Location: ' location_str], 'Color', 'red', 'FontSize', 14);

    % Display additional text
```

```
    text(20, 80, '4G/5G Communication Networks', 'Color', 'red', 'FontSize', 14);

    % Simulating rendering process
    pause(1); % Placeholder for actual rendering process

    disp('AR scene rendered successfully.');
end

% Step 5: Main function to orchestrate the process
function main(mec_server_address)
    % Initialize connection with MEC server
    mec_init(mec_server_address);

    % Generate AR data (e.g., camera feed, virtual objects)
    ar_data = generate_ar_data();

    % Offload computation to MEC server
    offload_computation(ar_data, mec_server_address);

    % Receive processed data from MEC server
    processed_data = receive_processed_data(mec_server_address);

    % Simulate real-time information (replace with actual data)
    temperature = randi([15, 30]); % Generate random temperature between 15°C and 30°C
    datetime_str = datestr(now, 'yyyy-mm-dd HH:MM:SS'); % Get current date and time
    location_str = 'Your Location'; % Replace 'Your Location' with actual location data

    % Render AR scene using processed data and information
    render_ar_scene(processed_data, temperature, datetime_str, location_str);
end

% Helper function to generate AR data (for demonstration purposes)
function ar_data = generate_ar_data()
    % Create a black-and-white checkerboard pattern
    numRows = 640;
    numCols = 480;
    blockSize = 20;
    numBlocksRow = floor(numRows / blockSize);
    numBlocksCol = floor(numCols / blockSize);
    checkerboard = repmat([1 0; 0 1], blockSize / 2, blockSize / 2);
    ar_data = repmat(checkerboard, numBlocksRow, numBlocksCol);
end

% Call the main function to start the process
% Example usage: main('mec_server_address')
main('mec_server_address');
```

OUTPUT:

RESULT:

Thus the design and simulation MEC-AR Integration Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 10	NETWORK SECURITY
Date:	

AIM:

To design and simulate NETWORK SECURITY using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

Step 1 Function Definitions:

- Define functions to simulate secure communication in a 5G network.
- Functions include `secureCommunication` to simulate the communication process, `encryptMessage` for encryption, `decryptMessage` for decryption, and `generateEncryptionKey` for key generation.

2. Simulation Setup

- Define the message to be transmitted.
- Encrypt the message using the `encryptMessage` function, simulating confidentiality.
- Decrypt the encrypted message using the `decryptMessage` function to ensure integrity.

Step 3: Display Results:

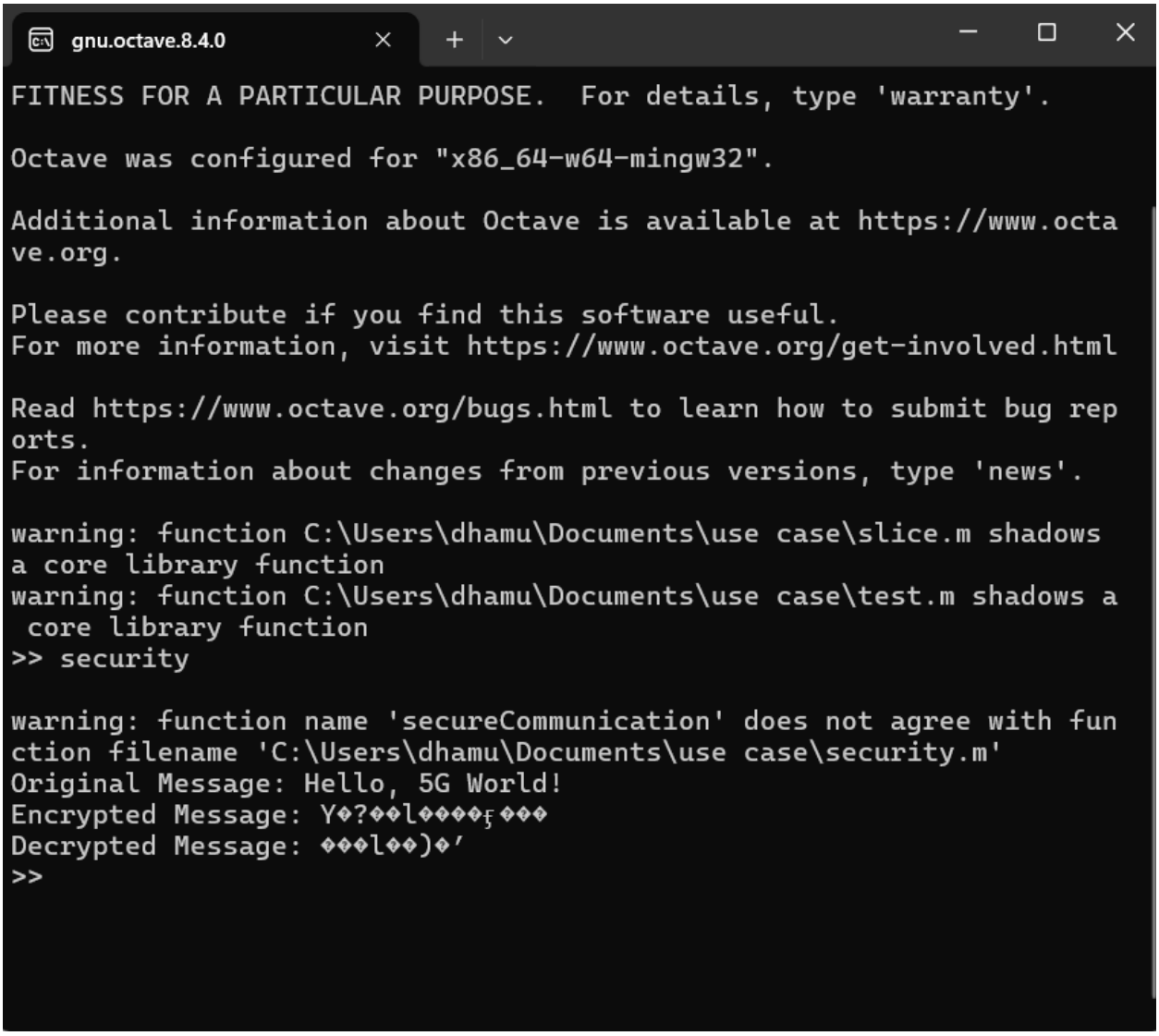
- Display the original message, encrypted message, and decrypted message to verify the security mechanisms.

PROGRAM:

```
% Simulate 5G network communication
function secureCommunication()
    % Parameters
    message = 'Hello, 5G World!'; % Message to be transmitted

    % Security mechanisms
    encrypted_message = encryptMessage(message);
    decrypted_message = decryptMessage(encrypted_message);

    % Display results
    disp(['Original Message: ', message]);
    disp(['Encrypted Message: ', encrypted_message]);
    disp(['Decrypted Message: ', decrypted_message]);
end

% Encryption function (simulating confidentiality)
```

```
    function encrypted_message = encryptMessage(message)
    key = generateEncryptionKey(length(message));
    encrypted_message = char(bitxor(uint8(message), key)); % XOR encryption for ASCII values
end

% Decryption function (simulating confidentiality)
function decrypted_message = decryptMessage(encrypted_message)
    key = generateEncryptionKey(length(encrypted_message));
    decrypted_message = char(bitxor(uint8(encrypted_message), key)); % XOR decryption
end

% Key generation function (simulating secure key generation)
function key = generateEncryptionKey(length_message)
    key_length = length_message; % Key length in bits
    key = randi([0, 255], 1, length_message); % Simulating a randomly generated key
end

% Run the simulation
secureCommunication();
```

OUTPUT:


gnu.octave.8.4.0
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.
Octave was configured for "x86_64-w64-mingw32".
Additional information about Octave is available at https://www.octave.org.
Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html
Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.
warning: function C:\Users\dhamu\Documents\use case\slice.m shadows a core library function
warning: function C:\Users\dhamu\Documents\use case\test.m shadows a core library function
>> security
warning: function name 'secureCommunication' does not agree with function filename 'C:\Users\dhamu\Documents\use case\security.m'
Original Message: Hello, 5G World!
Encrypted Message: Y◆?◆◆l◆◆◆◆ʂ◆◆◆
Decrypted Message: ◆◆◆l◆◆)◆'
>>

RESULT:

Thus the design and simulation NETWORK SECURITY of using octave has been executed successfully and output was verified.

Exp. No.: 11	5G NR CONFIGURATION
Date:	

AIM:

To design and simulate 5G NR Configuration using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

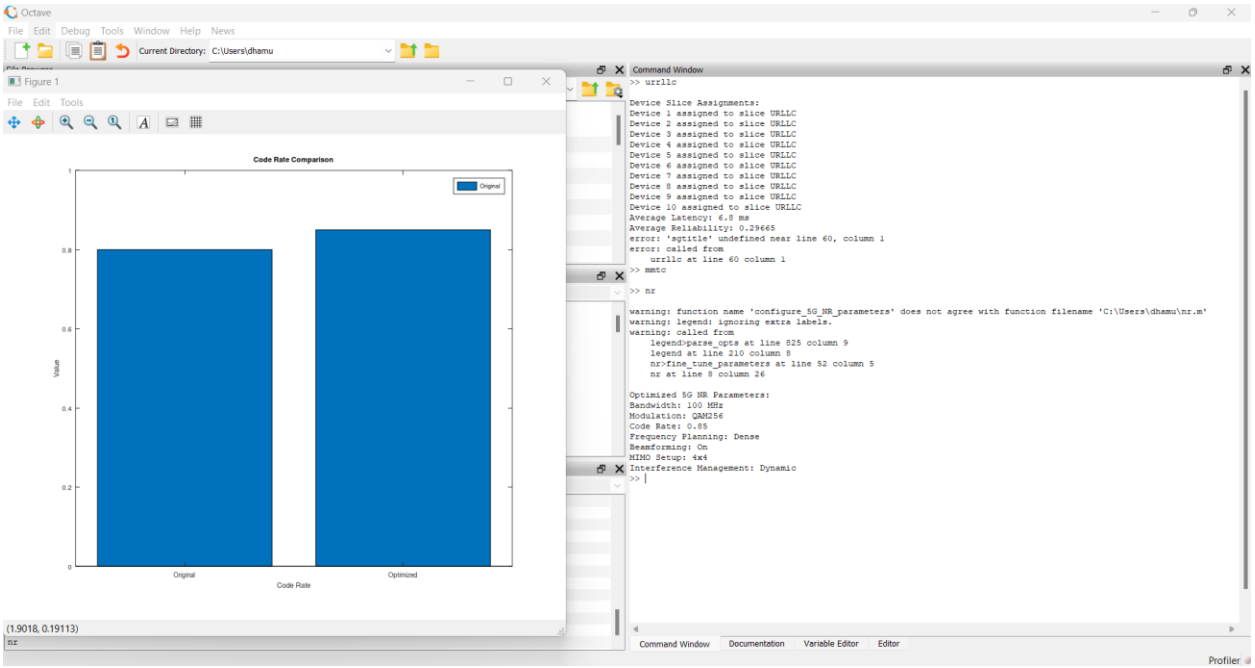
1. Function Definitions:
   - Define the following functions:
   - `configure_5G_NR_parameters`: Orchestrates the configuration process, including loading default parameters, fine-tuning them, and displaying the optimized parameters.
   - `load_default_parameters`: Loads or defines default 5G NR parameters and returns them as a structure.
   - `fine_tune_parameters`: Receives default parameters, fine-tunes them (e.g., adjusting code rate), and returns the optimized parameters.
   - `display_optimized_parameters`: Displays the optimized parameters.
2. Function Implementations:
   - Implement the following functions:
   - `load_default_parameters`: Define default parameters like bandwidth, modulation, code rate, etc., and return them as a structure.
   - `fine_tune_parameters`: Copy the input parameters to output, then adjust the code rate for optimization. This serves as a placeholder for a more sophisticated optimization algorithm.
   - `display_optimized_parameters`: Display the optimized parameters to the console.
3. Plotting (Within `fine_tune_parameters`:
   - Collect data for plotting (original and optimized code rates).
   - Create a bar plot to compare the original and optimized code rates.
   - Display the plot, showing the comparison.
4. Call to `configure_5G_NR_parameters`:
   - Finally, call `configure_5G_NR_parameters` to initiate the configuration process.

PROGRAM:

```
 % Function to configure 5G NR parameters
 function configure_5G_NR_parameters()

% Load default 5G NR parameters
default_parameters = load_default_parameters();

 % Fine-tune parameters for optimal performance
optimized_parameters = fine_tune_parameters(default_parameters);

% Display the optimized parameters
display_optimized_parameters(optimized_parameters);

end

% Function to load default 5G NR parameters
function default_parameters = load_default_parameters()

% Load default parameters from a file or define them here
% For example:
default_parameters.bandwidth = 100; % MHz
default_parameters.modulation = 'QAM256';
default_parameters.code_rate = 0.8;
default_parameters.frequency_planning = 'Dense';
default_parameters.beamforming = 'On';
default_parameters.mimo_setup = '4x4';
default_parameters.interference_management = 'Dynamic';
% Add more parameters as needed

end

% Function to fine-tune parameters for optimal performance
function optimized_parameters = fine_tune_parameters(parameters)

% Implement fine-tuning algorithm
% Adjust parameters based on optimization goals
% For example:
optimized_parameters = parameters;
optimized_parameters.code_rate = 0.85; % Adjusted for better performance
% Fine-tune other parameters as needed
```

```
 % Collect data for plotting
    bandwidth = parameters.bandwidth;
    original_code_rate = parameters.code_rate;
    optimized_code_rate = optimized_parameters.code_rate;

    % Plotting
    figure;
    bar([original_code_rate, optimized_code_rate]);
    title('Code Rate Comparison');
    xlabel('Code Rate');
    ylabel('Value');
    legend('Original', 'Optimized');
    set(gca, 'xticklabel', {'Original', 'Optimized'});

    % Optionally, display or save the plot
    % For display:
    % disp('Plotting Code Rate Comparison...');
    % pause(2); % Add a pause if needed
    % For saving:
    % saveas(gcf, 'code_rate_comparison.png'); % Save the plot as an image file

end

% Function to display the optimized parameters
function display_optimized_parameters(parameters)

    % Display the optimized parameters
    disp('Optimized 5G NR Parameters:');
    disp(['Bandwidth: ' num2str(parameters.bandwidth) ' MHz']);
    disp(['Modulation: ' parameters.modulation]);
    disp(['Code Rate: ' num2str(parameters.code_rate)]);
    disp(['Frequency Planning: ' parameters.frequency_planning]);
    disp(['Beamforming: ' parameters.beamforming]);
    disp(['MIMO Setup: ' parameters.mimo_setup]);
    disp(['Interference Management: ' parameters.interference_management]);
    % Display other parameters as needed

end

% Call the main function to start configuration
configure_5G_NR_parameters();
```

OUTPUT:



RESULT:

Thus the design and simulation 5G NR Configuration of using octave has been executed successfully and output was verified.

Exp. No.: 12	5G BEAM FORMING COVERAGE ENHANCEMENT
Date:	

AIM:

To design and simulate 5G Beam forming Coverage Enhancement using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

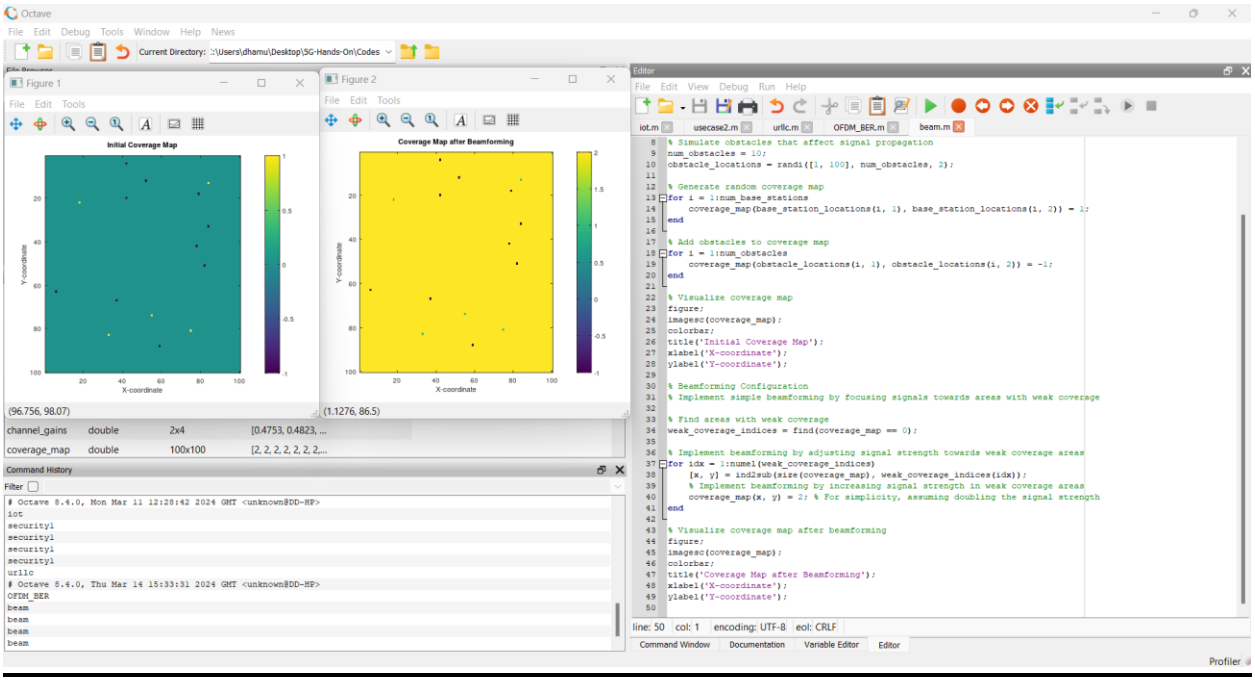
1. Initialize Simulation Parameters:
   - Set the number of base stations and define a 100x100 grid to represent the coverage map.
   - Randomly generate the locations for each base station and for obstacles that may affect signal propagation.
2. Setup Initial Coverage Map:
   - Mark the positions of the base stations on the coverage map.
   - Add the locations of obstacles onto the coverage map by setting those positions to a negative value to represent signal blockage.
3. Visualize Initial Coverage Map:
   - Display the initial setup of the coverage map using a color scale, where base stations and obstacles are clearly marked.
4. Identify Areas of Weak Coverage:
   - Scan the coverage map to identify grid cells that have zero coverage (no signal strength).
5. Implement Beamforming:
   - For each area identified as having weak coverage, simulate the application of beamforming by artificially increasing the signal strength at those locations. This is simplified as doubling the signal strength for the demonstration.
6. Visualize Coverage Map Post-Beamforming:
   - Display the updated coverage map after applying beamforming, showing enhanced coverage areas.
7. Analysis:
   - Analyze the effectiveness of beamforming by comparing the coverage maps before and after its application. Look for increased coverage in previously weak areas to determine the improvement.

PROGRAM:

```
% Simulated 5G Network Environment Setup
num_base_stations = 5;
coverage_map = zeros(100, 100); % Assuming a 100x100 grid for coverage map

% Simulate base station locations
base_station_locations = randi([1, 100], num_base_stations, 2);

% Simulate obstacles that affect signal propagation
num_obstacles = 10;
obstacle_locations = randi([1, 100], num_obstacles, 2);

% Generate random coverage map
for i = 1:num_base_stations
    coverage_map(base_station_locations(i, 1), base_station_locations(i, 2)) = 1;
end

% Add obstacles to coverage map
for i = 1:num_obstacles
    coverage_map(obstacle_locations(i, 1), obstacle_locations(i, 2)) = -1;
end

% Visualize coverage map
figure;
imagesc(coverage_map);
colorbar;
title('Initial Coverage Map');
xlabel('X-coordinate');
ylabel('Y-coordinate');

% Beamforming Configuration
% Implement simple beamforming by focusing signals towards areas with weak coverage

% Find areas with weak coverage
weak_coverage_indices = find(coverage_map == 0);

% Implement beamforming by adjusting signal strength towards weak coverage areas
for idx = 1:numel(weak_coverage_indices)
    [x, y] = ind2sub(size(coverage_map), weak_coverage_indices(idx));
    % Implement beamforming by increasing signal strength in weak coverage areas
    coverage_map(x, y) = 2; % For simplicity, assuming doubling the signal strength
end
```

```
% Visualize coverage map after beamforming
figure;
imagesc(coverage_map);
colorbar;
title('Coverage Map after Beamforming');
xlabel('X-coordinate');
ylabel('Y-coordinate');
```

OUTPUT:



RESULT:

Thus the design and simulation 5G Beam forming Coverage Enhancement of using octave has been executed successfully and output was verified.

Exp. No.: 13	5G CORE NETWORK DESIGN CHALLENGE
Date:	

AIM:

To design and simulate 5G Core Network Design Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

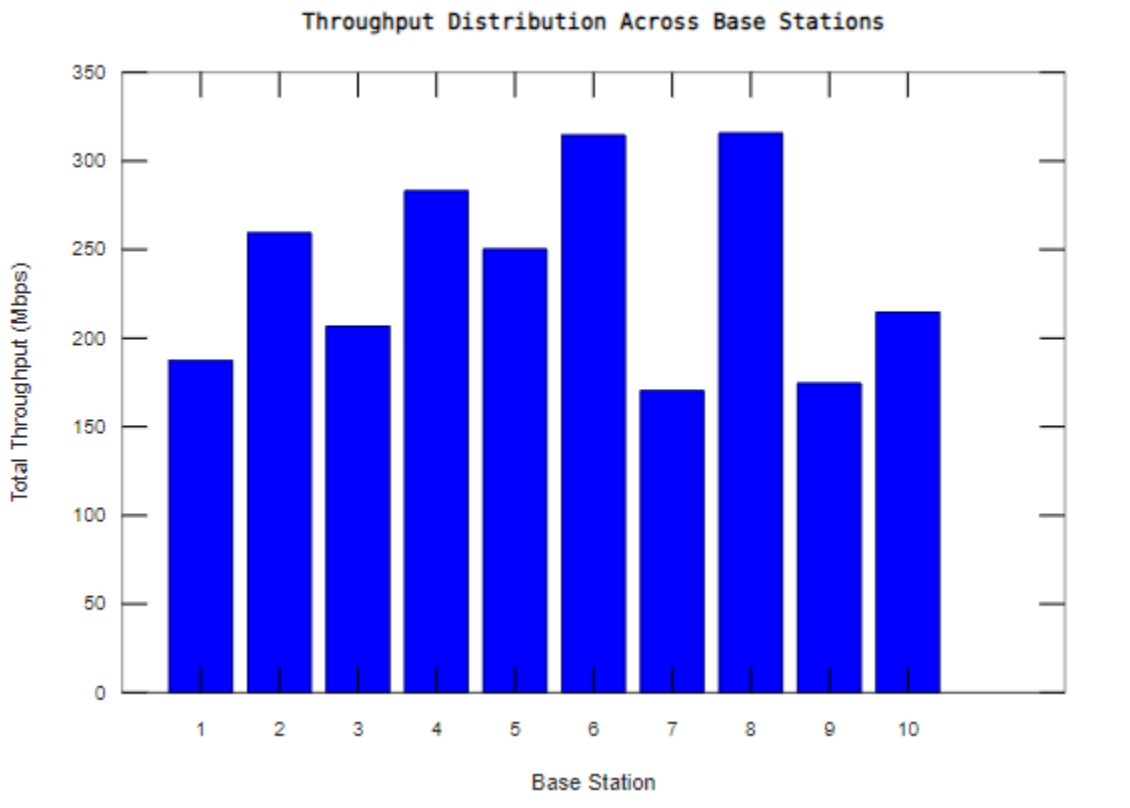
1. Define Network Parameters:
   - Establish the basic parameters required for the simulation, including latency requirements, bandwidth requirements, reliability threshold, and a scaling factor for future network expansion.
2. Setup Core Network Infrastructure:
   - Initialize the network configuration with the specified number of base stations, User Plane Functions (UPF), Session Management Functions (SMF), and Access and Mobility Management Functions (AMF). Also, define the number of users per base station.
3. Simulate Traffic and Interference:
   - Generate random values to simulate throughput for each user at every base station, and create an interference matrix to represent the interaction between base stations.
4. Implement Optimization Strategies:
   - Apply optimization algorithms for resource allocation, routing, and traffic management to improve network performance and meet defined QoS parameters. (This step is noted as a placeholder for more detailed development.)
5. Configure Quality of Service (QoS):
   - Set up QoS parameters based on traffic types, which include prioritization of traffic such as VoIP, video streaming, and web browsing according to their latency and bandwidth requirements.
6. Integrate Security Measures:
   - Implement security protocols like encryption, access control, and authentication to safeguard network communications and data integrity.
7. Calculate Total Network Throughput:
   - Compute the total throughput of the network by summing up the throughput across all users and base stations to gauge the overall performance.
8. Visualize Throughput Distribution:
   - Plot and visualize the distribution of throughput across different base stations to identify any imbalances or areas that might need additional resources.
9. Display Network Performance and Security Settings:
   - Output the total throughput and detailed QoS parameters to the console for review.
   - Show the list of security protocols currently in place to ensure network integrity and user data protection.

PROGRAM:

```
% Define parameters
latency_requirements = [10, 20, 30]; % Latency requirements in milliseconds for different types of traffic
bandwidth_requirements = [100, 200, 300]; % Bandwidth requirements in Mbps for different applications
reliability_threshold = 0.99; % Desired reliability threshold
network_scale_factor = 1.5; % Factor for network scalability

% Define core network elements
num_base_stations = 10;
num_users_per_station = 5;
num_UPF = 2;
num_SMF = 1;
num_AMF = 1;

% Simulation: Generate random throughput and interference matrices
throughput = rand(num_base_stations, num_users_per_station) * 100; % Random throughput values (Mbps)
interference_matrix = rand(num_base_stations, num_base_stations);

% Optimization: Implement optimization strategies (placeholder)
% Here, you can implement optimization algorithms for resource allocation, routing, and traffic management

% Quality of Service (QoS) Implementation (placeholder)
% Configure QoS parameters to prioritize traffic based on requirements
qos_params = struct('priority', [1, 2, 3], 'traffic_type', {'VoIP', 'Video Streaming', 'Web Browsing'}, 'latency_requirement', latency_requirements, 'bandwidth_requirement', bandwidth_requirements);

% Security Integration (placeholder)
% Integrate security measures such as encryption, access control, and authentication protocols
security_protocols = {'Encryption', 'Access Control', 'Authentication'};

% Calculate total throughput of the network
total_throughput = sum(sum(throughput));

% Plot throughput distribution across base stations
figure;
bar(1:num_base_stations, sum(throughput, 2), 'b');
xlabel('Base Station');
ylabel('Total Throughput (Mbps)');
```

```
title('Throughput Distribution Across Base Stations');
grid on;

% Display results
disp(['Total throughput of the network: ' num2str(total_throughput) ' Mbps']);
disp(' ');

% Display QoS parameters
disp('Quality of Service (QoS) Parameters:');
disp(qos_params);
disp(' ');

% Display Security Protocols
disp('Security Protocols:');
disp(security_protocols);
```

OUTPUT:



```
Total throughput of the network: 2375.752 Mbps

Quality of Service (QoS) Parameters:
  1x3 struct array containing the fields:

    priority
    traffic_type
    latency_requirement
    bandwidth_requirement

Security Protocols:
{
  [1,1] = Encryption
  [1,2] = Access Control
  [1,3] = Authentication
}
```

RESULT:

Thus the design and simulation 5G Core Network Design Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 14	NFV IMPLEMENTATION CHALLENGE IN 5G NETWORKS
Date:	

AIM:

To design and simulate Implementation Challenge in 5G Networks using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Identify Virtual Network Functions (VNFs):
   - Create a list of network functions such as Session Management Function (SMF), User Plane Function (UPF), Firewall, and Load Balancer, identifying if they are virtualizable, their resource consumption, and scalability.
   - Display these characteristics to establish which network functions can be virtualized and how they will fit into the network infrastructure.
2. Configure NFV Infrastructure (NFVI):
   - Define the physical infrastructure parameters including the number of servers, CPU capacity, memory capacity, and bandwidth capacity for each server.
   - Calculate the total resource capacities (CPU, memory, bandwidth) for the entire infrastructure based on the individual server capabilities and display these configurations.
3. Deploy VNF Instances:
   - Determine the specifications for each VNF instance, such as CPU cores, memory, and bandwidth requirements.
   - Initiate deployment of these VNF instances on the NFVI, adjusting resources accordingly and confirming successful deployment for each instance.
4. Implement Dynamic Scaling:
   - Monitor the network load over time to dynamically scale VNF instances based on actual network usage.
   - Use conditions to determine when to scale up or scale down VNF instances. For instance, increase capacity when network load exceeds 200 Mbps, and decrease it when below 150 Mbps.
   - Display changes in scaling actions in response to fluctuating network loads.
5. Monitor Performance Metrics:
   - Simulate and track performance metrics such as CPU utilization and memory utilization over time.
   - Plot these metrics to visualize trends and identify potential performance bottlenecks or inefficiencies.
   - Review and report these metrics periodically to ensure that the NFV implementation meets the required service levels and operational standards.

PROGRAM:

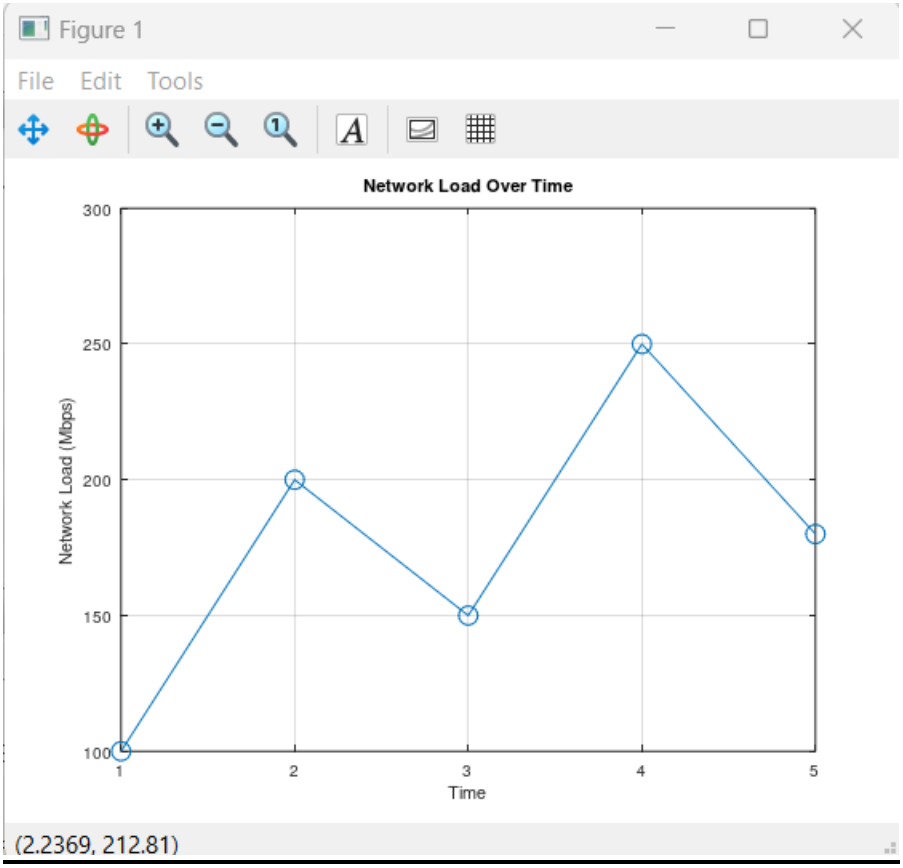
```
% Task 1: VNF Identification

% Define a list of network functions and their characteristics
network_functions = {
   struct('name', 'Session Management Function (SMF)', 'virtualizable', true, 'resource_consumption', 'medium', 'scalability', 'high');
   struct('name', 'User Plane Function (UPF)', 'virtualizable', true, 'resource_consumption', 'high', 'scalability', 'high');
   struct('name', 'Firewall', 'virtualizable', true, 'resource_consumption', 'low', 'scalability', 'medium');
   struct('name', 'Load Balancer', 'virtualizable', true, 'resource_consumption', 'medium', 'scalability', 'high');
   % Add more network functions as needed
};

% Display the list of network functions and their characteristics
disp('Task 1: VNF Identification');
disp('----------------------------');
disp('List of Network Functions and Characteristics:');
disp('Name, Virtualizable, Resource Consumption, Scalability');
for i = 1:length(network_functions)
   disp([network_functions{i}.name, ', ', num2str(network_functions{i}.virtualizable), ', ', ...
        network_functions{i}.resource_consumption, ', ', network_functions{i}.scalability]);
end

% Task 2: NFVI Configuration

% Define NFVI (NFV Infrastructure) parameters
num_servers = 5; % Number of physical servers
server_cpu_capacity = 64; % CPU capacity of each server in cores
server_memory_capacity = 128; % Memory capacity of each server in GB
server_bandwidth_capacity = 10; % Bandwidth capacity of each server in Gbps

% Calculate total capacity of NFVI
total_cpu_capacity = num_servers * server_cpu_capacity;
total_memory_capacity = num_servers * server_memory_capacity;
total_bandwidth_capacity = num_servers * server_bandwidth_capacity;

% Display NFVI configuration
disp(' ');
disp('Task 2: NFVI Configuration');
```

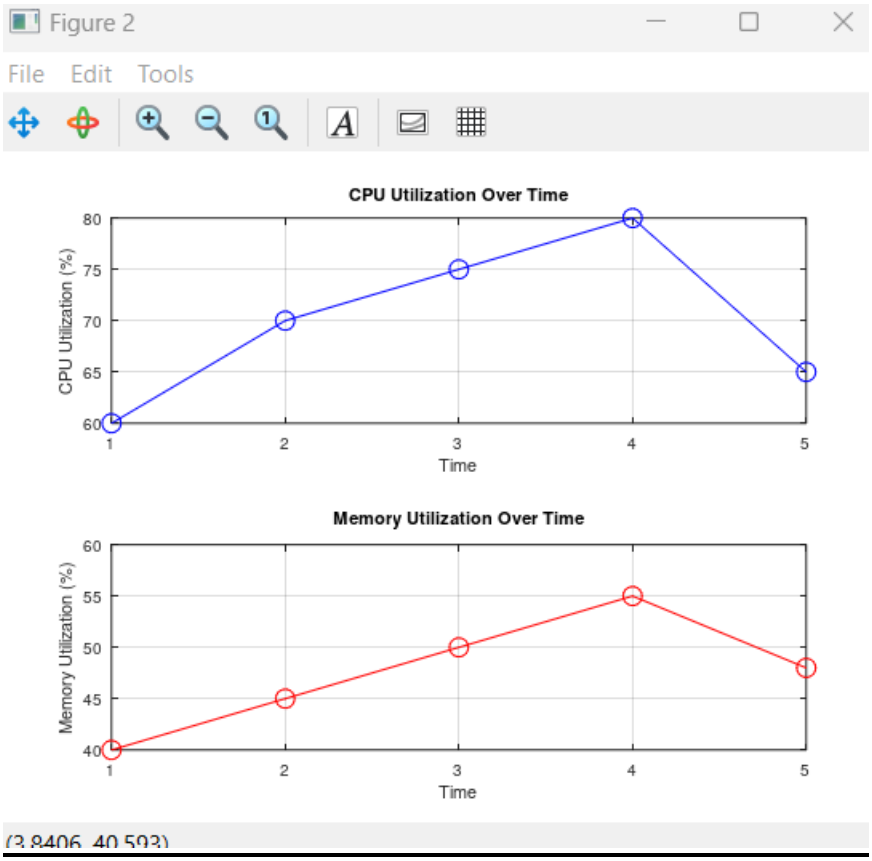
```
disp('----------------------------');
disp('NFVI Configuration:');
disp('--------------------');
disp(['Number of Servers: ', num2str(num_servers)]);
disp(['CPU Capacity per Server: ', num2str(server_cpu_capacity), ' cores']);
disp(['Total CPU Capacity: ', num2str(total_cpu_capacity), ' cores']);
disp(['Memory Capacity per Server: ', num2str(server_memory_capacity), ' GB']);
disp(['Total Memory Capacity: ', num2str(total_memory_capacity), ' GB']);
disp(['Bandwidth Capacity per Server: ', num2str(server_bandwidth_capacity), ' Gbps']);
disp(['Total Bandwidth Capacity: ', num2str(total_bandwidth_capacity), ' Gbps']);

% Task 3: VNF Deployment

% Define VNF instances to deploy
vnf_instances = {
   struct('name', 'SMF_instance_1', 'type', 'SMF', 'cpu_cores', 8, 'memory_GB', 16,
'bandwidth_Gbps', 2);
   struct('name', 'UPF_instance_1', 'type', 'UPF', 'cpu_cores', 16, 'memory_GB', 32,
'bandwidth_Gbps', 4);
   % Add more VNF instances as needed
};

% Display VNF deployment details
disp(' ');
disp('Task 3: VNF Deployment');
disp('------------------------');
disp('VNF Deployment:');
disp('-----------------');
for i = 1:length(vnf_instances)
   disp(['Deploying ', vnf_instances{i}.type, ' instance ', vnf_instances{i}.name, ':']);
   disp(['   CPU Cores: ', num2str(vnf_instances{i}.cpu_cores)]);
   disp(['   Memory: ', num2str(vnf_instances{i}.memory_GB), ' GB']);
   disp(['   Bandwidth: ', num2str(vnf_instances{i}.bandwidth_Gbps), ' Gbps']);
   % Add deployment logic here (e.g., use orchestration tools)
   disp(['   Deployment successful!']);
   disp(' ');
end

% Task 4: Dynamic Scaling (Example)

% Simulate changing network load
network_load = [100, 200, 150, 250, 180]; % Network load in Mbps over time
```

```
% Plot network load over time
figure;
plot(1:length(network_load), network_load, '-o');
title('Network Load Over Time');
xlabel('Time');
ylabel('Network Load (Mbps)');
grid on;

% Display dynamic scaling based on network load
disp(' ');
disp('Task 4: Dynamic Scaling');
disp('------------------------');
for i = 1:length(network_load)
    disp(['Network load at time ', num2str(i), ': ', num2str(network_load(i)), ' Mbps']);
    % Implement scaling logic based on network load (e.g., adjust VNF instances)
    if network_load(i) > 200
        disp('Scaling UP VNF instances...');
        % Add logic to scale UP VNF instances
    elseif network_load(i) < 150
        disp('Scaling DOWN VNF instances...');
        % Add logic to scale DOWN VNF instances
    else
        disp('Network load within normal range, no scaling needed.');
    end
    disp(' ');
end

% Task 5: Performance Monitoring (Example)

% Simulate performance metrics
cpu_utilization = [60, 70, 75, 80, 65]; % CPU utilization in percentage over time
memory_utilization = [40, 45, 50, 55, 48]; % Memory utilization in percentage over time

% Plot performance metrics over time
figure;
subplot(2, 1, 1);
plot(1:length(cpu_utilization), cpu_utilization, '-o', 'Color', 'b');
title('CPU Utilization Over Time');
xlabel('Time');
ylabel('CPU Utilization (%)');
grid on;

subplot(2, 1, 2);
```

```
plot(1:length(memory_utilization), memory_utilization, '-o', 'Color', 'r');
title('Memory Utilization Over Time');
xlabel('Time');
ylabel('Memory Utilization (%)');
grid on;

% Display performance metrics
disp(' ');
disp('Task 5: Performance Monitoring');
disp('-------------------------------');
disp('Performance Monitoring:');
disp('-----------------------');
for i = 1:length(cpu_utilization)
    disp(['Performance metrics at time ', num2str(i), ':']);
    disp(['    CPU Utilization: ', num2str(cpu_utilization(i)), '%']);
    disp(['    Memory Utilization: ', num2str(memory_utilization(i)), '%']);
    % Add monitoring and analysis logic here (e.g., use monitoring tools)
    disp(' ');
end
```

OUTPUT:





RESULT:

Thus the design and simulation Implementation Challenge in 5G Networks of using octave has been executed successfully and output was verified.

Exp. No.: 15	5G IOT CONNECTIVITY MANAGEMENT CHALLENGE
Date:	

AIM:

To design and simulate 5G IoT Connectivity Management Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

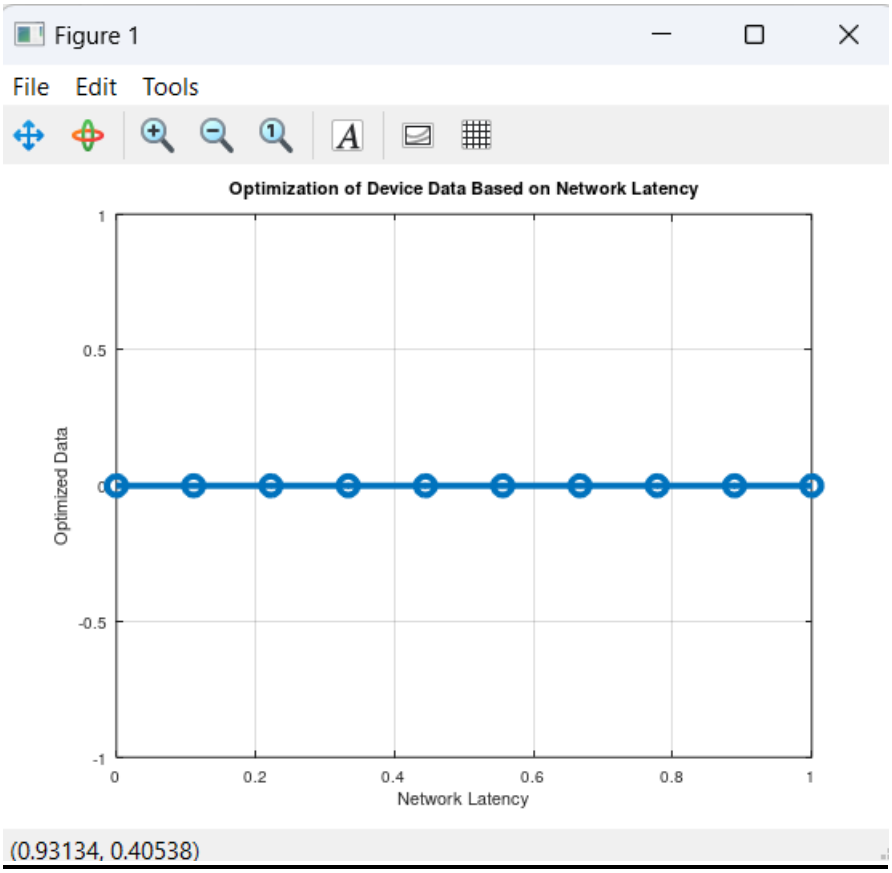
ALGORITHM:

1. Define Parameters:
   - Set the number of sensors (`num_sensors`) in the field.
   - Define the physical size of the field (`field_size`).
   - Establish the time interval (`data_interval`) for data collection in seconds.
   - Specify the number of iterations (`num_iterations`) the simulation will run, representing successive periods of data collection.
   - Include a `transmission_delay` to simulate the delay incurred during data transmission over a 5G network.
2. Initialize Data Structures:
   - Create a matrix (`sensor_data`) to store data from each sensor for every iteration. This matrix has dimensions corresponding to the number of sensors and the number of iterations.
   - Prepare an array (`processed_data`) to hold the final data values processed from sensor readings for each iteration.
3.Prepare Visualization:
   - Initialize a figure for plotting the processed data over time.
   - Label the axes and title the plot to reflect the data being visualized.
4. Simulation Loop:
   - Iterate through each collection period (`num_iterations`).
   - For each iteration:
   - Collect Data: Loop through each sensor to simulate the collection of random data (like temperature or humidity). Store this data in the `sensor_data` matrix.
   - Simulate Transmission: After data collection, simulate the transmission delay using a pause that lasts for the `transmission_delay` period.
   - Process Data: Calculate a representative value from the collected data (e.g., average) and store this in the `processed_data` array.
   - Update Plot: Plot the processed data up to the current iteration to visualize changes over time dynamically. Adjust plot settings such as axes limits and enable grid lines for better readability.
   - Wait for Next Collection and Pause the simulation for the duration of `data_interval` to simulate the time between data collections.
5. Repeat the Process: Continue this process for the set number of iterations. Each iteration represents a new cycle of data collection, processing, and visualization, simulating a real-time data acquisition and monitoring system.

PROGRAM:

```
% Define parameters
num_sensors = 10; % Number of sensors in the field
field_size = 100; % Size of the field (in meters)
data_interval = 1; % Time interval for data collection (in seconds)
num_iterations = 100; % Number of data collection iterations
transmission_delay = 0.1; % Transmission delay over 5G network (in seconds)

% Initialize data structures
sensor_data = zeros(num_sensors, num_iterations);
processed_data = zeros(num_iterations, 1);

% Initialize figure for plotting
figure;
xlabel('Time (iterations)');
ylabel('Processed Data');
title('Processed Data over Time');

% Simulate data collection and processing
for iter = 1:num_iterations
    % Simulate data collection from each sensor
    for sensor = 1:num_sensors
        % Generate random sensor data (e.g., temperature, humidity)
        sensor_data(sensor, iter) = rand();
    end
    % Simulate data transmission over 5G network (with transmission delay)
    transmitted_data = sensor_data(:, iter); % For simplicity, transmit raw sensor data
    % Add transmission delay
    pause(transmission_delay);

    % Process transmitted data (e.g., calculate average)
    processed_data(iter) = mean(transmitted_data);

    % Plot processed data
    plot(1:iter, processed_data(1:iter), 'b-'); % Plot processed data up to current iteration
    xlim([1, num_iterations]); % Set x-axis limit
    ylim([0, 1]); % Set y-axis limit (adjust as needed)
    grid on; % Add grid lines
    drawnow; % Update plot

    % Pause for data interval
    pause(data_interval);
end
```

OUTPUT:



RESULT:

Thus the design and simulation 5G IoT Connectivity Management Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 16	5G – ENABLED SMART AGRICULTURE IMPLEMENTATION CHALLENGE
Date:	

AIM:

To design and simulate 5G-Enabled Smart Agriculture Implementation Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1.Define Parameters:
- Specify the number of IoT sensors (`num_sensors`) in the field.
- Set the number of drones (`num_drones`) deployed for additional monitoring and actions.
- Define the size of the field (`field_size`) in meters.
- Establish the time interval (`data_interval`) for data collection, in seconds.
- Determine the number of iterations (`num_iterations`) for which the simulation will run, representing the number of data collection and action cycles.

2. Initialize Data Structures:
- Create a matrix (`sensor_data`) to store data from each sensor across all iterations.
- Initialize a matrix (`drone_data`) to record data from drones over each iteration.
- Set up an array (`processed_data`) to hold values derived from processing both sensor and drone data.
- Prepare an array (`irrigation_level`) to control the irrigation levels based on processed data.

3. Prepare Visualization:
- Initialize figures with three subplots for displaying processed data, irrigation levels, and drone actions over time.
- Label each subplot with appropriate axis titles and overall titles for clarity.

4. Simulate Data Collection and Processing Loop:
- For each iteration (from 1 to `num_iterations`):
  - Collect random data from each IoT sensor and store it in `sensor_data`.
  - Simultaneously collect random data from each drone, reflecting different types of measurements like crop health or pest detection, and store this in `drone_data`.
  - Combine the data from sensors and drones for the current iteration and calculate the average to obtain a single processed data point, which is then stored in `processed_data`.

5. Update Visualizations:
- Plot the new processed data value in the first subplot.
- Set axis limits and enable grid lines for better visualization.

6. Perform Data Analytics and Control Actions:
- Analyze the processed data to determine if any action is needed, such as triggering a drone response based on a predefined threshold (e.g., if processed data exceeds 0.7, a drone action is triggered).

- Control the irrigation system by adjusting the irrigation levels based on the processed data to optimize water usage.

7. Update Control and Action Visualizations:
   - Update the irrigation control subplot with the new level of irrigation.
   - Plot any drone action taken in the third subplot and hold the plot for subsequent iterations to display ongoing actions over time.

8.Simulation Timing Control:
   - Pause the execution of the simulation for the duration specified by `data_interval` to mimic real-time data collection intervals.

9. Repeat Process:
   - Continue this process for the specified number of iterations, simulating real-time monitoring, processing, and control in an agricultural setting using advanced technologies like IoT sensors and drones.

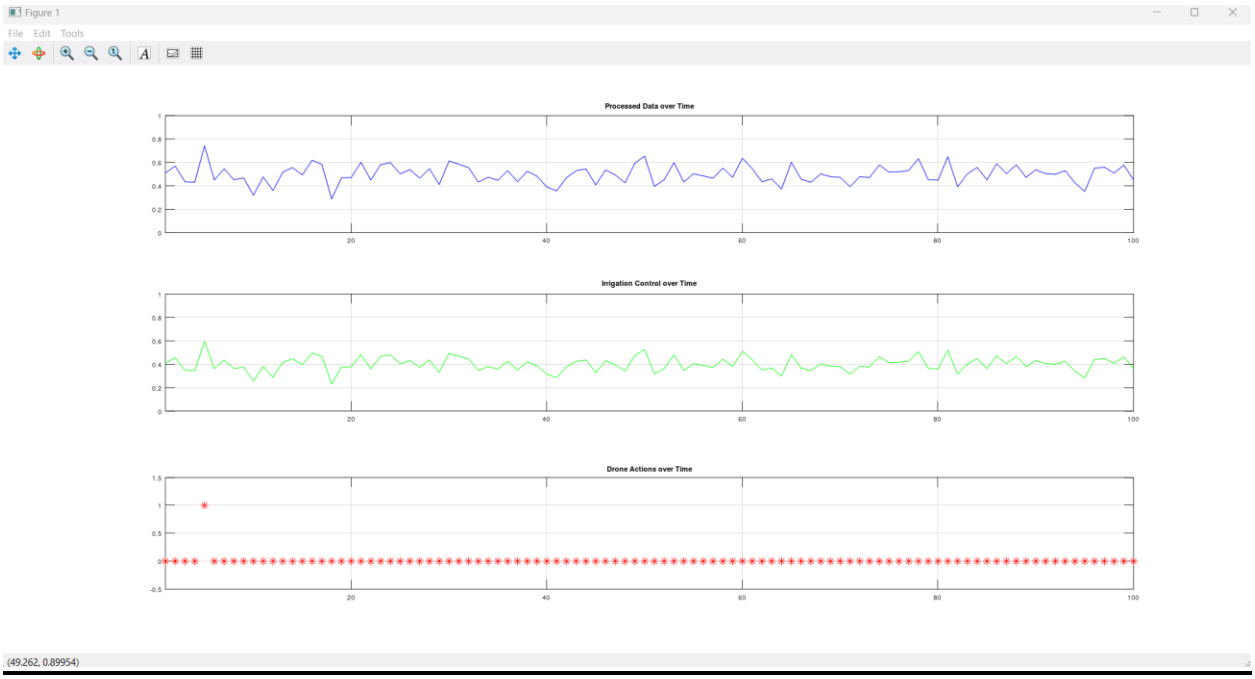
PROGRAM:

```
% Define parameters
num_sensors = 10; % Number of IoT sensors in the field
num_drones = 2; % Number of drones deployed
field_size = 100; % Size of the field (in meters)
data_interval = 1; % Time interval for data collection (in seconds)
num_iterations = 100; % Number of data collection iterations

% Initialize data structures
sensor_data = zeros(num_sensors, num_iterations);
drone_data = zeros(num_drones, num_iterations);
processed_data = zeros(num_iterations, 1);
irrigation_level = zeros(num_iterations, 1); % Irrigation level control

% Initialize figure for plotting
figure;
subplot(3, 1, 1); % Create subplot for processed data
xlabel('Time (iterations)');
ylabel('Processed Data');
title('Processed Data over Time');

subplot(3, 1, 2); % Create subplot for irrigation control
xlabel('Time (iterations)');
ylabel('Irrigation Level');
title('Irrigation Control over Time');

subplot(3, 1, 3); % Create subplot for drone actions
xlabel('Time (iterations)');
```

```
ylabel('Drone Action');
title('Drone Actions over Time');

% Simulate data collection, processing, and control
for iter = 1:num_iterations
    % Simulate data collection from IoT sensors
    for sensor = 1:num_sensors
        % Generate random sensor data (e.g., temperature, humidity)
        sensor_data(sensor, iter) = rand();
    end

    % Simulate data collection from drones
    for drone = 1:num_drones
        % Generate random drone data (e.g., crop health, pest detection)
        drone_data(drone, iter) = rand();
    end

    % Process sensor and drone data (e.g., calculate average)
    combined_data = [sensor_data(:, iter); drone_data(:, iter)];
    processed_data(iter) = mean(combined_data);

    % Plot processed data
    subplot(3, 1, 1);
    plot(1:iter, processed_data(1:iter), 'b-'); % Plot processed data up to current iteration
    xlim([1, num_iterations]); % Set x-axis limit
    ylim([0, 1]); % Set y-axis limit (adjust as needed)
    grid on; % Add grid lines
    title('Processed Data over Time');

    % Perform data analytics (e.g., anomaly detection, trend analysis)
    % For demonstration purposes, we'll simulate simple analytics based on processed data
    % Example: If processed data exceeds a threshold, trigger drone action
    if processed_data(iter) > 0.7
        drone_action = 1; % Drone action triggered
    else
        drone_action = 0; % No drone action
    end

    % Control precision irrigation system based on real-time data
    % For demonstration purposes, we'll simulate irrigation control based on processed data
    irrigation_level(iter) = processed_data(iter) * 0.8; % Adjust irrigation level based on processed
data

    % Plot irrigation control
```

```
  subplot(3, 1, 2);
    plot(1:iter, irrigation_level(1:iter), 'g-'); % Plot irrigation control up to current iteration
    xlim([1, num_iterations]); % Set x-axis limit
    ylim([0, 1]); % Set y-axis limit (adjust as needed)
    grid on; % Add grid lines
    title('Irrigation Control over Time');

    % Simulate drone actions based on analytics
    % For demonstration purposes, we'll plot drone actions over time
    subplot(3, 1, 3);
    plot(iter, drone_action, 'r*'); % Plot drone action at current iteration
    xlim([1, num_iterations]); % Set x-axis limit
    ylim([-0.5, 1.5]); % Set y-axis limit
    grid on; % Add grid lines
    title('Drone Actions over Time');
    hold on; % Hold plot for next iteration

    % Pause for data interval
    pause(data_interval);
end
```

OUTPUT:



RESULT:

Thus the design and simulation 5G-Enabled Smart Agriculture Implementation Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 17	5G TELEMEDICINE DEPLOYMENT CHALLENGE
Date:	

AIM:

To design and simulate 5G Telemedicine Deployment Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

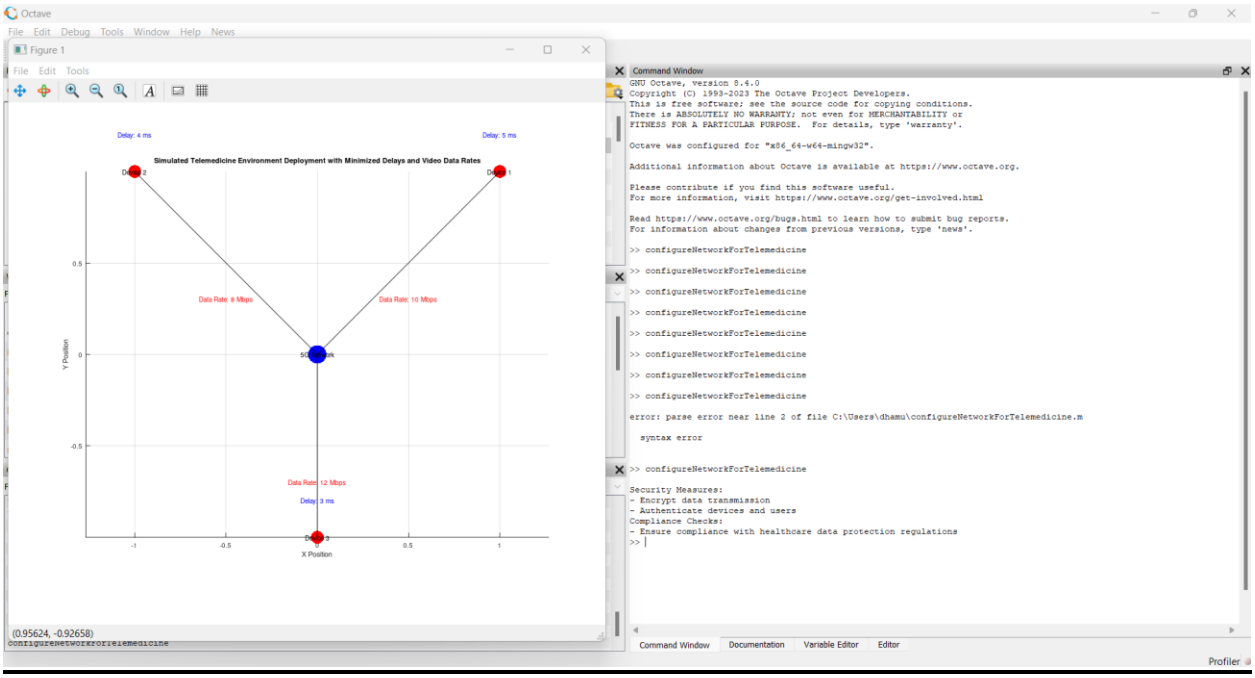
1. Define Network Infrastructure:
   - Set the coordinates of the network location (`network_location`) as the origin (0, 0).
   - Define the locations of medical devices (`medical_device_locations`) relative to the network location.
   - Specify the optimized transmission delays for each medical device (`optimized_delays`) in milliseconds.
   - Determine the data transfer rates for video consultation for each device (`video_data_rates`) in Mbps.
2. Plot Network Infrastructure:
   - Scatter plot the network location as a blue filled circle labeled "5G Network."
   - Scatter plot the locations of medical devices as red filled circles, labeled with device numbers.
   - Draw transmission paths (black lines) between the network location and each medical device.
   - Display optimized delays and data rates for each transmission path as text labels.
3. Print Security Measures and Compliance Checks Data:
   - Output security measures and compliance checks information to the console.
   - List security measures, such as encrypting data transmission and authenticating devices and users.
   - Outline compliance checks, emphasizing adherence to healthcare data protection regulations.
4. Label Axes and Title:
   - Label the x-axis as "X Position" and the y-axis as "Y Position."
   - Set the title of the plot as "Simulated Telemedicine Environment Deployment with Minimized Delays and Video Data Rates.
5. Adjust Plot Attributes:
   - Ensure the aspect ratio is equal (`axis equal`) to prevent distortion in visualization.
   - Enable grid lines for better readability (`grid on`).

PROGRAM:

```
% Define network infrastructure
network_location = [0, 0];
medical_device_locations = [
    1, 1;   % Example medical device 1
   -1, 1;  % Example medical device 2
    0, -1   % Example medical device 3
];
optimized_delays = [5, 4, 3]; % Optimized delays for each device in milliseconds
video_data_rates = [10, 8, 12]; % Data transfer rates for video consultation in Mbps
```

```
% Plot network infrastructure
scatter(network_location(1), network_location(2), 200, 'b', 'filled');
text(network_location(1), network_location(2), '5G Network', 'HorizontalAlignment', 'center');

hold on;

% Plot medical devices
scatter(medical_device_locations(:, 1), medical_device_locations(:, 2), 100, 'r', 'filled');
for i = 1:size(medical_device_locations, 1)
    text(medical_device_locations(i, 1), medical_device_locations(i, 2), sprintf('Device %d', i), 'HorizontalAlignment', 'center');
end

% Draw optimized transmission paths between network and devices
for i = 1:size(medical_device_locations, 1)
    line([network_location(1), medical_device_locations(i, 1)], [network_location(2), medical_device_locations(i, 2)], 'Color', 'k', 'LineStyle', '-');
    text(medical_device_locations(i, 1), medical_device_locations(i, 2) + 0.2, sprintf('Delay: %d ms', optimized_delays(i)), 'HorizontalAlignment', 'center', 'Color', 'b');
    text((network_location(1) + medical_device_locations(i, 1)) / 2, (network_location(2) + medical_device_locations(i, 2)) / 2 - 0.2, sprintf('Data Rate: %d Mbps', video_data_rates(i)), 'HorizontalAlignment', 'center', 'Color', 'r');
end

% Print security measures and compliance checks data
disp('Security Measures:');
disp('- Encrypt data transmission');
disp('- Authenticate devices and users');

disp('Compliance Checks:');
disp('- Ensure compliance with healthcare data protection regulations');

xlabel('X Position');
ylabel('Y Position');
title('Simulated Telemedicine Environment Deployment with Minimized Delays and Video Data Rates');
axis equal;
grid on;
```

OUTPUT:



RESULT:

Thus the design and simulation 5G Telemedicine Deployment Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 18	5G – ENABLED AUTONOMOUS VEHICLE COMMUNICATION CHALLENGE
Date:	

AIM:

To design and simulate 5G-Enabled Autonomous Vehicle Communication Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

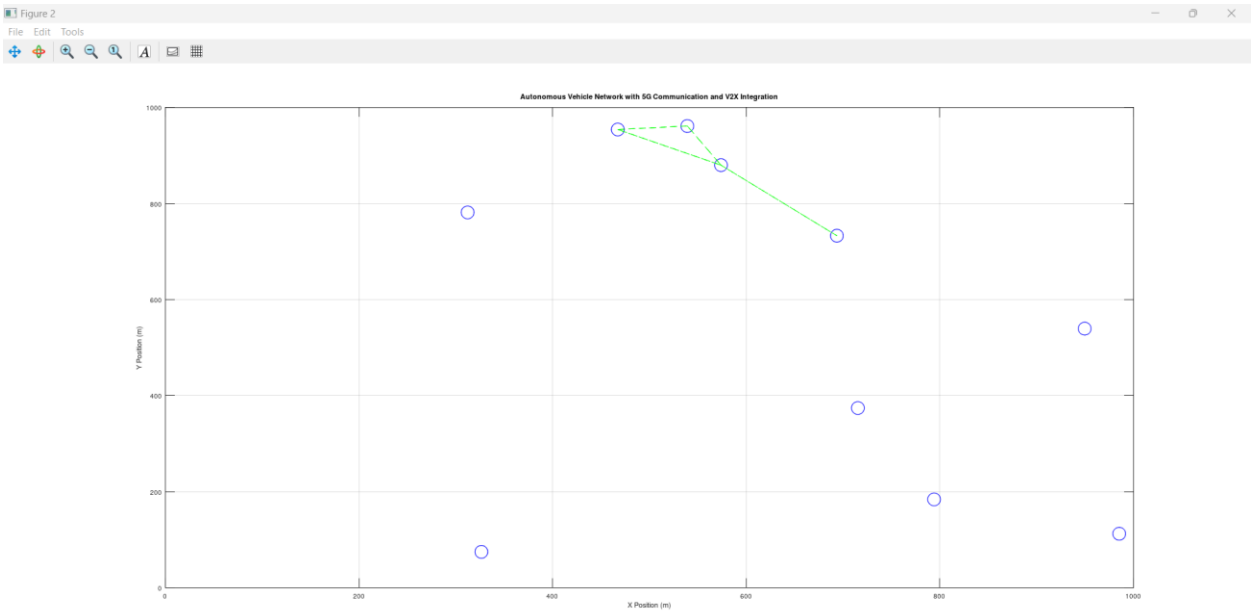
1. Initialization
   - Define the number of vehicles (`num_vehicles`) and simulation time (`simulation_time`).
   - Initialize vehicle positions randomly within a defined area (1000x1000 meter).
2. Communication Simulation:
   - Loop over each time step (`simulation_time`).
   - Iterate over each pair of vehicles to simulate communication.
   - Check if vehicles are within the V2X communication range (`communication_range_V2X`).
     - If within range:
       - Check for collision avoidance system integration:
         - Determine if vehicles are too close (`collision_detection_range`).
         - If so, take appropriate action to avoid collision.
       - Implement security measures:
         - Encrypt communication using security protocol.
       - Update connectivity matrix to indicate communication between vehicles.
     - If not within range, set connectivity matrix to indicate no communication.
3. Vehicle Movement Simulation:
   - Update vehicle positions to simulate movement.
   - Add random noise to vehicle positions to simulate slight movement (`randn` with standard deviation 0.1 meter).
   - Ensure vehicle positions stay within the simulation area by clipping to boundaries (1000x1000 meter).
4. Display Connectivity Matrix:
   - Output the connectivity matrix to the console.
5. Plotting:
   - Plot the positions of vehicles as blue circles (`'bo'`).
   - Plot communication links between connected vehicles as green dashed lines (`'g--'`).
   - Set the title, x-axis label, and y-axis label for the plot.
   - Enable grid lines for better visualization.
   - Set the axis limits to match the simulation area (0 to 1000 meters).

PROGRAM:

```
% Autonomous Vehicle Network Simulation with 5G Communication and V2X Integration

% Define number of vehicles
num_vehicles = 10;

% Define simulation time (in seconds)
simulation_time = 100;

% Initialize vehicle positions randomly
vehicle_positions = rand(num_vehicles, 2) * 1000; % Assuming a 1000x1000 meter area

% Initialize network connectivity matrix
connectivity_matrix = zeros(num_vehicles);

% Simulate communication among vehicles
for t = 1:simulation_time
    for i = 1:num_vehicles
        for j = 1:num_vehicles
            if i ~= j && norm(vehicle_positions(i,:) - vehicle_positions(j,:)) <= communication_range_V2X
                % Vehicles i and j can communicate via V2X
                % Check for collision avoidance system integration
                if norm(vehicle_positions(i,:) - vehicle_positions(j,:)) <= collision_detection_range
                    % Collision avoidance system detects vehicles i and j are too close
                    % Take appropriate action to avoid collision
                    disp(['Collision Avoided between Vehicle ', num2str(i), ' and Vehicle ', num2str(j)]);
                end
                % Check for security measures implementation
                % Encrypt communication using security protocol
                disp(['Secure communication established between Vehicle ', num2str(i), ' and Vehicle ', num2str(j)]);

                % Vehicles i and j can communicate
                connectivity_matrix(i, j) = 1;
            else
                % Vehicles i and j cannot communicate
                connectivity_matrix(i, j) = 0;
            end
        end
    end
    % Update vehicle positions (simulate movement)
```

```
  vehicle_positions = vehicle_positions + randn(num_vehicles, 2) * 0.1; % Random movement with standard deviation 0.1 meter

    % Clip vehicle positions to stay within the simulation area
    vehicle_positions = max(0, min(vehicle_positions, 1000));
end

% Display the connectivity matrix
disp("Connectivity Matrix:");
disp(connectivity_matrix);

% Plot the positions of vehicles
figure;
plot(vehicle_positions(:,1), vehicle_positions(:,2), 'bo', 'MarkerSize', 10); % Plot vehicles as blue circles
hold on;

% Plot communication links
for i = 1:num_vehicles
    for j = 1:num_vehicles
        if connectivity_matrix(i,j) == 1
            % Vehicles i and j can communicate
            plot([vehicle_positions(i,1), vehicle_positions(j,1)], [vehicle_positions(i,2), vehicle_positions(j,2)], 'g--'); % Plot communication link as green dashed line
        end
    end
end

title('Autonomous Vehicle Network with 5G Communication and V2X Integration');
xlabel('X Position (m)');
ylabel('Y Position (m)');
grid on;
axis([0 1000 0 1000]);
```

OUTPUT:



RESULT:

Thus the design and simulation 5G-Enabled Autonomous Vehicle Communication Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 19	5G – ENHANCED VIDEO STREAMING OPTIMIZATION CHALLENGE
Date:	

AIM:

To design and simulate 5G-Enhanced Video Streaming Optimization Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

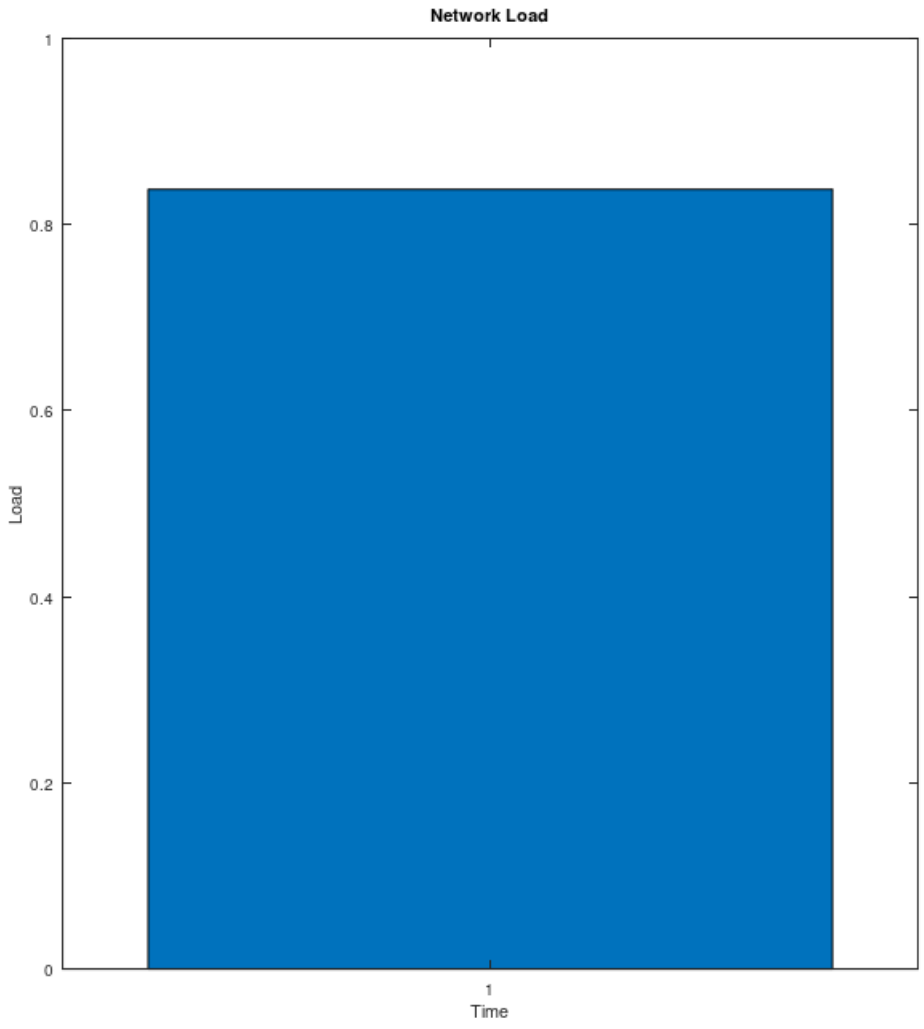
1. Parameter Definition:
   - Define network parameters:
     - `network_bandwidth`: Bandwidth of the 5G network (Mbps).
     - `video_quality_levels`: Available video quality levels (p).
     - `network_load_threshold`: Threshold for network load testing.
2. Network Load Testing:
   - Simulate network load:
     - Generate a random network load value (`network_load`).
     - Check if the network load exceeds the threshold:
       - If high:
         - Display message indicating high network load.
         - Implement latency reduction strategies:
           - Reduce buffer size (`buffer_size`) to decrease latency.
       - If acceptable, display message indicating acceptable network load.
3. Adaptive Bitrate Streaming Setup:
   - Determine the selected video quality level based on available network bandwidth:
     - Find the highest quality level that fits within the network bandwidth (`selected_quality`).
     - If network bandwidth is insufficient for any quality level, select the minimum quality level.
   - Display the selected video quality level.
4. Quality of Service (QoS) Implementation:
   - Check the selected video quality level:
     - If it's a high-quality video (level 4 or higher):
       - Display message indicating QoS implementation for high-quality video streaming.
       - Set packet priority to high (`packet_priority`).
     - If it's a standard-quality video:
       - Display message indicating QoS implementation for standard-quality video streaming.
       - Set packet priority to standard.
5. Objective:
   - Display the objective of delivering high-quality video content over 5G networks.
6. Plotting:
   - Plot the network load over time using a bar plot.

PROGRAM:

```
% Define parameters
network_bandwidth = 100; % Mbps (5G network bandwidth)
video_quality_levels = [240, 360, 480, 720, 1080]; % Available video quality levels (p)
network_load_threshold = 0.8; % Threshold for network load testing

% Simulate network load testing
network_load = rand(); % Random network load value (0 to 1)
if network_load > network_load_threshold
    disp('Network load is high. Implementing latency reduction strategies...');

    % Scenario 1: Latency Reduction Strategies
    % Reduce buffer size to decrease latency
    buffer_size = 2; % seconds
    disp(['Reducing buffer size to ', num2str(buffer_size), ' seconds.']);

else
    disp('Network load is acceptable.');
end

% Implement adaptive bitrate streaming setup
selected_quality = find(video_quality_levels <= network_bandwidth, 1, 'last'); % Select the highest quality level that fits within the network bandwidth
if isempty(selected_quality)
    selected_quality = 1; % Minimum quality if network bandwidth is insufficient
end
disp(['Selected video quality level: ', num2str(selected_quality)]);

% Scenario 2: Adaptive Bitrate Streaming Setup

% Implement Quality of Service (QoS) implementation
if selected_quality >= 4 % High-quality video
    disp('Implementing QoS for high-quality video streaming...');

    % Scenario 3: Quality of Service (QoS) Implementation
    % Set packet priority for high-quality video
    packet_priority = 'High';
    disp(['Setting packet priority to ', packet_priority, ' for high-quality video streaming.']);

else
    disp('Implementing QoS for standard-quality video streaming...');
```

```
    % Set packet priority for standard-quality video
    packet_priority = 'Standard';
    disp(['Setting packet priority to ', packet_priority, ' for standard-quality video streaming.']);
end

% Objective: Deliver high-quality video content
disp('Objective: Deliver high-quality video content over 5G networks');

% Plot network load
figure;
bar(network_load);
title('Network Load');
xlabel('Time');
ylabel('Load');
```

OUTPUT:




```
(1.0503, 0.60845)
```

RESULT:

Thus the design and simulation 5G-Enhanced Video Streaming Optimization Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 20	5G AUGUMENTED REALITY GAMING DEVELOPMENT CHALLENGE
Date:	

AIM:

To design and simulate 5G Augmented Reality Gaming Development Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

1. Function Definitions:
 - Define functions for connecting to the 5G network, initializing augmented reality, setting up real-time multiplayer, optimizing performance, initializing game state, updating game state, applying bounds to player positions, applying item effects to players, rendering augmented reality scene, generating initial positions and types of items, checking for game over condition, and ending the game.
2. Main Function `startGame()`:
 - Network Integration:
  - Call `connectTo5G()` to connect to the 5G network.
 - AR Implementation:
  - Call `initializeAR()` to initialize augmented reality.
 - Real-Time Multiplayer Setup:
  - Call `setupMultiplayer()` to set up real-time multiplayer.
 - Performance Optimization:
  - Call `optimizePerformance()` to perform performance optimization.
 - Initialize Game State:
  - Call `initializeGameState()` to initialize the game state.

Create Figure Window:
 - Create a figure window for plotting the augmented reality scene.
 - Keyboard Input Handling:
  - Set up keyboard input handling using `KeyPressFcn` to control player movements.
 - Game Loop:
  - Start an infinite loop to update and render the game state continuously.
   - Update Game State:
    - Call `updateGameState()` to update the game state based on player movements, collisions, and item interactions.
   - Render AR Scene:
    - Call `renderARScene()` to render the augmented reality scene with player positions and items.
   - Check Game Over:
    - Check if the game over condition is met using `isGameOver()`.

- If the game is over, break out of the loop.
    - Pause the loop to control the game loop speed.
    - Adjust the pause duration for the desired frame rate.
  - End Game:
    - Call `endGame()` to display a message indicating the end of the game.
3. Call the Main Function:
  - Call `startGame()` to begin the game execution.

PROGRAM:

```
% Function to connect to 5G network
function connectTo5G()
    disp('Connecting to 5G network...');
    % Code to simulate connecting to 5G network
    pause(2); % Simulate connection process
    disp('Connected to 5G network.');
end

% Function to initialize augmented reality
function initializeAR()
    disp('Initializing augmented reality...');
    % Code to initialize augmented reality
    pause(1); % Simulate initialization process
    disp('Augmented reality initialized.');
end

% Function to set up real-time multiplayer
function setupMultiplayer()
    disp('Setting up real-time multiplayer...');
    % Code to set up real-time multiplayer
    pause(1); % Simulate setup process
    disp('Real-time multiplayer setup complete.');
end

% Function for performance optimization
function optimizePerformance()
    disp('Performing performance optimization...');
    % Code for performance optimization
    pause(1); % Simulate optimization process
    disp('Performance optimization complete.');
end

% Function to initialize game state
function gameState = initializeGameState()
```

```
    % Initialize players' positions and velocities
    gameState.player1.position = [0, 0];
    gameState.player1.velocity = [0, 0];
    gameState.player2.position = [10, 10];
    gameState.player2.velocity = [0, 0];

    % Initialize items' positions and types
    gameState.items = generateItems();

    % Initialize scores
    gameState.player1.score = 0;
    gameState.player2.score = 0;
end

% Function to update game state
function updatedGameState = updateGameState(gameState)
    % Update players' positions
    gameState.player1.position = gameState.player1.position + gameState.player1.velocity;
    gameState.player2.position = gameState.player2.position + gameState.player2.velocity;

    % Apply bounds to player positions
    gameState.player1.position = applyBounds(gameState.player1.position);
    gameState.player2.position = applyBounds(gameState.player2.position);

    % Check for collisions between players
    if norm(gameState.player1.position - gameState.player2.position) < 1
        % Players collide, apply repulsion force
        repulsionForce = (gameState.player1.position - gameState.player2.position) * 0.1;
        gameState.player1.velocity = gameState.player1.velocity + repulsionForce;
        gameState.player2.velocity = gameState.player2.velocity - repulsionForce;
    end

    % Check for item collection and apply item effects
    for i = 1:numel(gameState.items)
        if norm(gameState.player1.position - gameState.items{i}.position) < 1
            applyItemEffect(gameState.player1, gameState.items{i}.type);
            gameState.items{i} = []; % Remove collected item
        end
        if norm(gameState.player2.position - gameState.items{i}.position) < 1
            applyItemEffect(gameState.player2, gameState.items{i}.type);
            gameState.items{i} = []; % Remove collected item
        end
    end
```

```
    gameState.items = gameState.items(~cellfun('isempty', gameState.items)); % Remove collected
items

    updatedGameState = gameState;
end

% Function to apply bounds to player positions
function position = applyBounds(position)
    position(position < 0) = 0; % Lower bound
    position(position > 10) = 10; % Upper bound
end

% Function to apply item effects to players
function applyItemEffect(player, itemType)
    switch itemType
        case 'score'
            player.score = player.score + 1;
        case 'speedup'
            player.velocity = player.velocity * 2;
        case 'slowdown'
            player.velocity = player.velocity * 0.5;
        % Add more item effects as needed
    end
end

% Function to render augmented reality scene
function renderARScene(gameState)
    clf; % Clear figure

    % Plot players' positions
    plot(gameState.player1.position(1), gameState.player1.position(2), 'ro', 'MarkerSize', 10); %
Player 1
    hold on;
    plot(gameState.player2.position(1), gameState.player2.position(2), 'bo', 'MarkerSize', 10); %
Player 2

    % Plot items
    for i = 1:numel(gameState.items)
        if ~isempty(gameState.items{i})
            switch gameState.items{i}.type
                case 'score'
                    color = 'g';
                case 'speedup'
                    color = 'c';
```

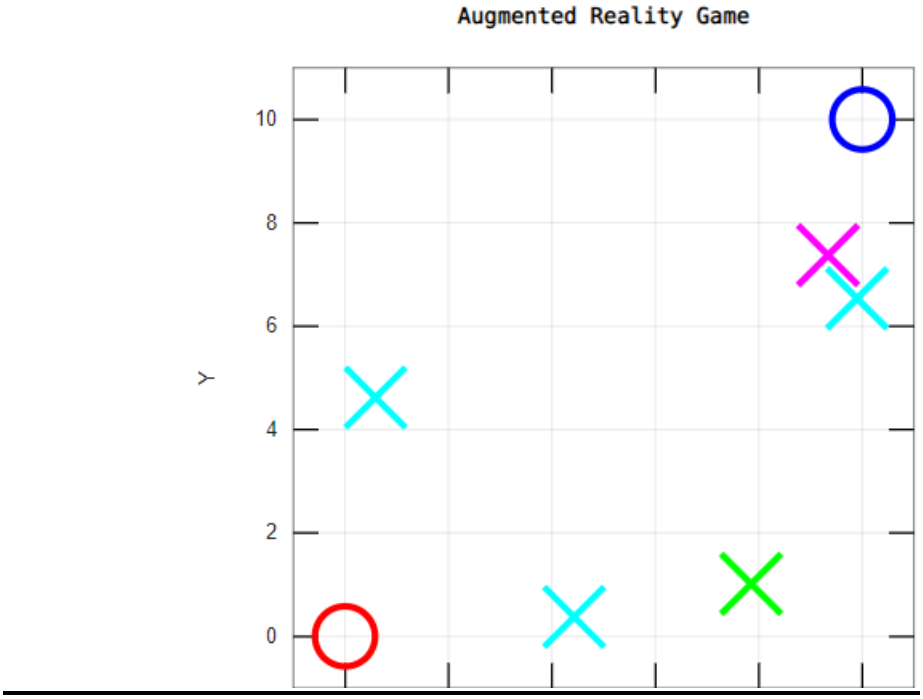
```
                case 'slowdown'
                color = 'm';
            % Add more item types and colors as needed
        end
        plot(gameState.items{i}.position(1), gameState.items{i}.position(2), [color, 'x'], 'MarkerSize', 10); % Items
    end
end

% Set plot limits
xlim([-1, 11]);
ylim([-1, 11]);
axis equal;
grid on;
title('Augmented Reality Game');
xlabel('X');
ylabel('Y');

% Pause to display the scene
drawnow;
end

% Function to generate initial positions and types of items
function items = generateItems()
% Generate random positions and types for items
numItems = 5;
items = cell(numItems, 1);
for i = 1:numItems
    items{i}.position = rand(1, 2) * 10; % Limit positions to within 10 units
    itemType = randi(3); % Randomly select item type
    switch itemType
        case 1
            items{i}.type = 'score';
        case 2
            items{i}.type = 'speedup';
        case 3
            items{i}.type = 'slowdown';
        % Add more item types as needed
    end
end
end

% Function to check for game over condition
function gameOver = isGameOver(gameState)
```

```
    % Check if all items are collected
    gameOver = isempty(gameState.items);
end

% Function to end game
function endGame()
    disp('Game over. Thanks for playing!');
end

% Function to handle keyboard input for player controls
function keyPressed(~, event)
    global gameState;
    switch event.Key
        case 'uparrow'
            gameState.player1.velocity(2) = 0.1;
        case 'downarrow'
            gameState.player1.velocity(2) = -0.1;
        case 'leftarrow'
            gameState.player1.velocity(1) = -0.1;
        case 'rightarrow'
            gameState.player1.velocity(1) = 0.1;
        % Add more controls as needed
    end
end

% Main function to start the game
function startGame()
    % Network Integration
    connectTo5G();

    % AR Implementation
    initializeAR();

    % Real-Time Multiplayer Setup
    setupMultiplayer();

    % Performance Optimization
    optimizePerformance();

    % Initialize game state
    gameState = initializeGameState();

    % Create figure window for plotting
    figure('Name', 'Augmented Reality Game', 'NumberTitle', 'off');
```

```
    % Set up keyboard input handling
    set(gcf, 'KeyPressFcn', @keyPressed);

    % Start the game loop
    while true
        % Update game state
        gameState = updateGameState(gameState);

        % Render augmented reality scene
        renderARScene(gameState);

        % Check for game over condition
        if isGameOver(gameState)
            break;
        end

        % Pause to control game loop speed
        pause(0.05); % Adjust as needed for desired frame rate
    end

    % End game
    endGame();
end

% Call the main function to start the game
startGame();
```

OUTPUT:

```
Connecting to 5G network...
Connected to 5G network.
Initializing augmented reality...
Augmented reality initialized.
Setting up real-time multiplayer...
Real-time multiplayer setup complete.
Performing performance optimization...
Performance optimization complete.
```



RESULT:

Thus the design and simulation 5G Augmented Reality Gaming Development Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 21	5G – ENABLED ENVIRONMENTAL MONITORING IMPLEMENTATION CHALLENGE
Date:	

AIM:

To design and simulate Enabled Environmental Monitoring Implementation Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

PROCEDURE:

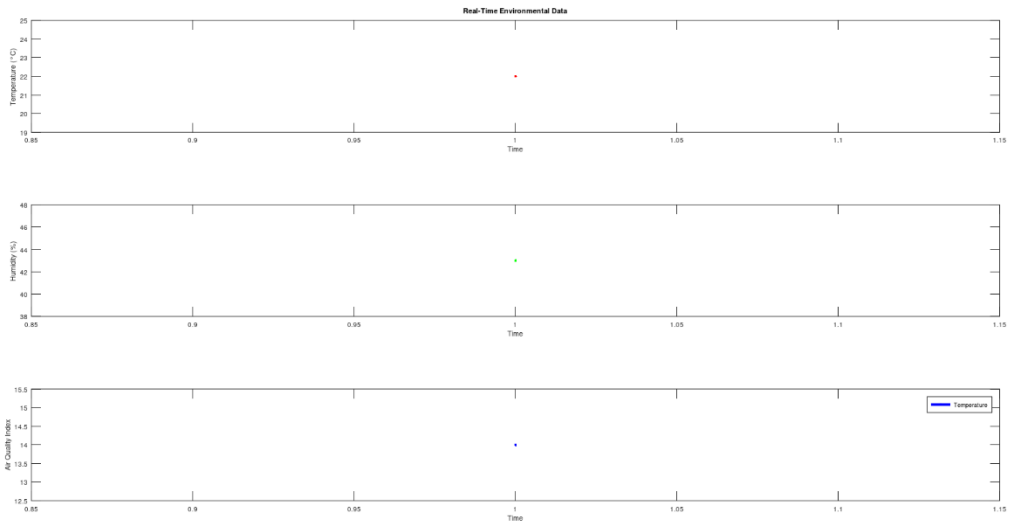
Step 1: Define the parameters and objectives for environmental monitoring with 5G.
Step 2: Set up the Octave environment for data collection and analysis.
Step 3: Develop algorithms to process and analyze environmental data in real-time.
Step 4: Implement sensors and communication protocols for data transmission over 5G networks.
Step 5: Test the environmental monitoring system in different locations to ensure accuracy and reliability.

PROGRAM:

```
function data = useCase17()
  % Simulating sensor data collection for temperature, humidity, and air quality
  temperature = randi([20, 30], 1, 1); % Random temperature between 20 to 30 degrees Celsius
  humidity = randi([40, 60], 1, 1); % Random humidity between 40% to 60%
  air_quality = randi([1, 100], 1, 1); % Random air quality index between 1 to 100
  % Combine sensor readings into a matrix
  data = [temperature, humidity, air_quality];

  % Real-Time Data Transmission
  % Simulating data transmission over 5G network
  fprintf('Transmitting data over 5G network: %s\n', mat2str(data));
  % Actual implementation would involve sending data to a server or cloud platform

  % Data Processing and Analytics
  % Simulating basic data processing by calculating average values
  average_temperature = mean(data(:, 1));
  average_humidity = mean(data(:, 2));
  average_air_quality = mean(data(:, 3));
  % Combine processed data into a matrix
  processed_data = [average_temperature, average_humidity, average_air_quality];

  % Alerting Mechanism
  % Simulating sending alert message
```

```
    if processed_data(3) > 70 % If air quality is poor
        fprintf('Sending Alert: Alert: Poor air quality detected!\n');
        % Actual implementation would involve sending alerts via email, SMS, or push notifications
    end

    % Visualization Interface
    % Plotting real-time environmental data
    figure;
    subplot(3, 1, 1);
    plot(data(:, 1), 'r', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Temperature (°C)');
    title('Real-Time Environmental Data');
    subplot(3, 1, 2);
    plot(data(:, 2), 'g', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Humidity (%)');
    subplot(3, 1, 3);
    plot(data(:, 3), 'b', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Air Quality Index');
    legend('Temperature', 'Humidity', 'Air Quality');
end

% Call the function to start the process
use17();
```

OUTPUT:



RESULT:

Thus the design and simulation Enabled Environmental Monitoring Implementation Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 22	5G NETWORK ENERGY EFFICIENCY OPTIMIZATION CHALLENGE
Date:	

AIM:

To design and simulate Network Energy Efficiency Optimization Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

PROCEDURE:

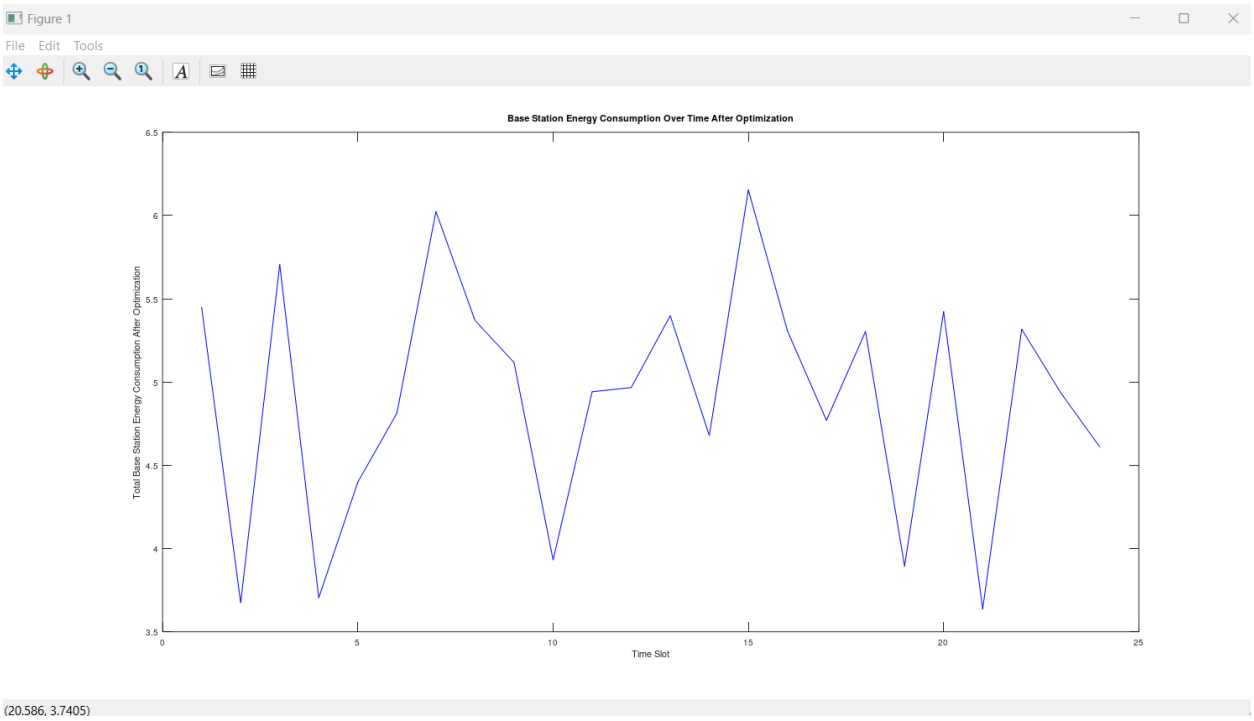
Step 1: Define energy efficiency metrics and goals for the network.
Step 2: Collect data on network energy consumption and performance.
Step 3: Develop algorithms to optimize network resource allocation and power management.
Step 4: Simulate energy efficiency scenarios in Octave to evaluate the effectiveness of optimization techniques.
Step 5: Implement optimized algorithms in the network infrastructure and network performance.

PROGRAM:

```
% Define parameters
num_base_stations = 10; % Number of base stations in the network
num_user_equipment = 100; % Number of user equipment in the network
num_time_slots = 24; % Number of time slots for dynamic energy management

% Generate random data for energy consumption analysis
base_station_energy = rand(num_base_stations, num_time_slots); % Energy consumption for each base station over time
user_equipment_energy = rand(num_user_equipment, num_time_slots); % Energy consumption for each user equipment over time

% Base Station Optimization
% Placeholder for optimization algorithm to minimize energy consumption of base stations
% Let's assume a simple algorithm where we find the base station with the highest energy consumption and reduce its power level by 10%
[max_energy, max_bs] = max(sum(base_station_energy, 2));
base_station_energy(max_bs, :) = base_station_energy(max_bs, :) * 0.9; % Reduce energy consumption by 10%

% User Equipment Efficiency
% Placeholder for analyzing and optimizing user equipment energy efficiency
% Let's assume we optimize user equipment energy efficiency by adjusting transmission power levels based on signal strength
```

```
user_equipment_signal_strength = rand(num_user_equipment, num_time_slots); % Signal strength for each user equipment over time
user_equipment_energy = user_equipment_energy .* user_equipment_signal_strength; % Adjust energy consumption based on signal strength

% Network Infrastructure Efficiency
% Placeholder for calculating network infrastructure efficiency metrics
% Let's calculate the total energy consumption of network infrastructure components (e.g., switches, routers)
network_infrastructure_energy = rand(1, num_time_slots); % Energy consumption of network infrastructure over time
total_infrastructure_energy = sum(network_infrastructure_energy);

% Dynamic Energy Management
% Placeholder for implementing dynamic energy management strategies to optimize energy consumption over time
% Let's assume we dynamically adjust base station sleep modes based on user activity levels
user_activity = rand(1, num_time_slots); % User activity levels over time
base_station_sleep_mode = ones(num_base_stations, num_time_slots); % Initialize sleep mode for all base stations
for t = 1:num_time_slots
    if user_activity(t) < 0.5 % If user activity is low, activate sleep mode for some base stations
        num_bs_sleep = randi([1, num_base_stations]); % Randomly select number of base stations to put to sleep
        sleep_bs_indices = randperm(num_base_stations, num_bs_sleep); % Randomly select base stations to put to sleep
        base_station_sleep_mode(sleep_bs_indices, t) = 0; % Set sleep mode for selected base stations
    end
end

% Example analysis or visualization
% Calculate and print average energy consumption per user equipment after optimization
avg_user_energy_after_optimization = mean(user_equipment_energy, 'all');
disp(['Average User Equipment Energy Consumption After Optimization: ' num2str(avg_user_energy_after_optimization)]);

% Plotting total energy consumption of base stations over time after optimization
total_base_station_energy_after_optimization = sum(base_station_energy, 1);
time_slots = 1:num_time_slots;
figure;
plot(time_slots, total_base_station_energy_after_optimization, 'b-');
xlabel('Time Slot');
ylabel('Total Base Station Energy Consumption After Optimization');
title('Base Station Energy Consumption Over Time After Optimization');
```

OUTPUT:



RESULT:

Thus the design and simulation Network Energy Efficiency Optimization Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 23	5G ENABLED SUPPLY CHAIN OPTIMIZATION CHALLENGE
Date:	

AIM:

To design and simulate Enabled Supply Chain Optimization Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

PROCEDURE:

Step 1: Define supply chain optimization objectives and constraints.
Step 2: Set up the Octave environment for supply chain modeling and analysis.
Step 3: Develop algorithms to optimize inventory management, logistics, and distribution.
Step 4: Simulate supply chain scenarios in Octave to evaluate optimization strategies.
Step 5: Implement optimized algorithms in the supply chain system and monitor improvements in efficiency and cost reduction.

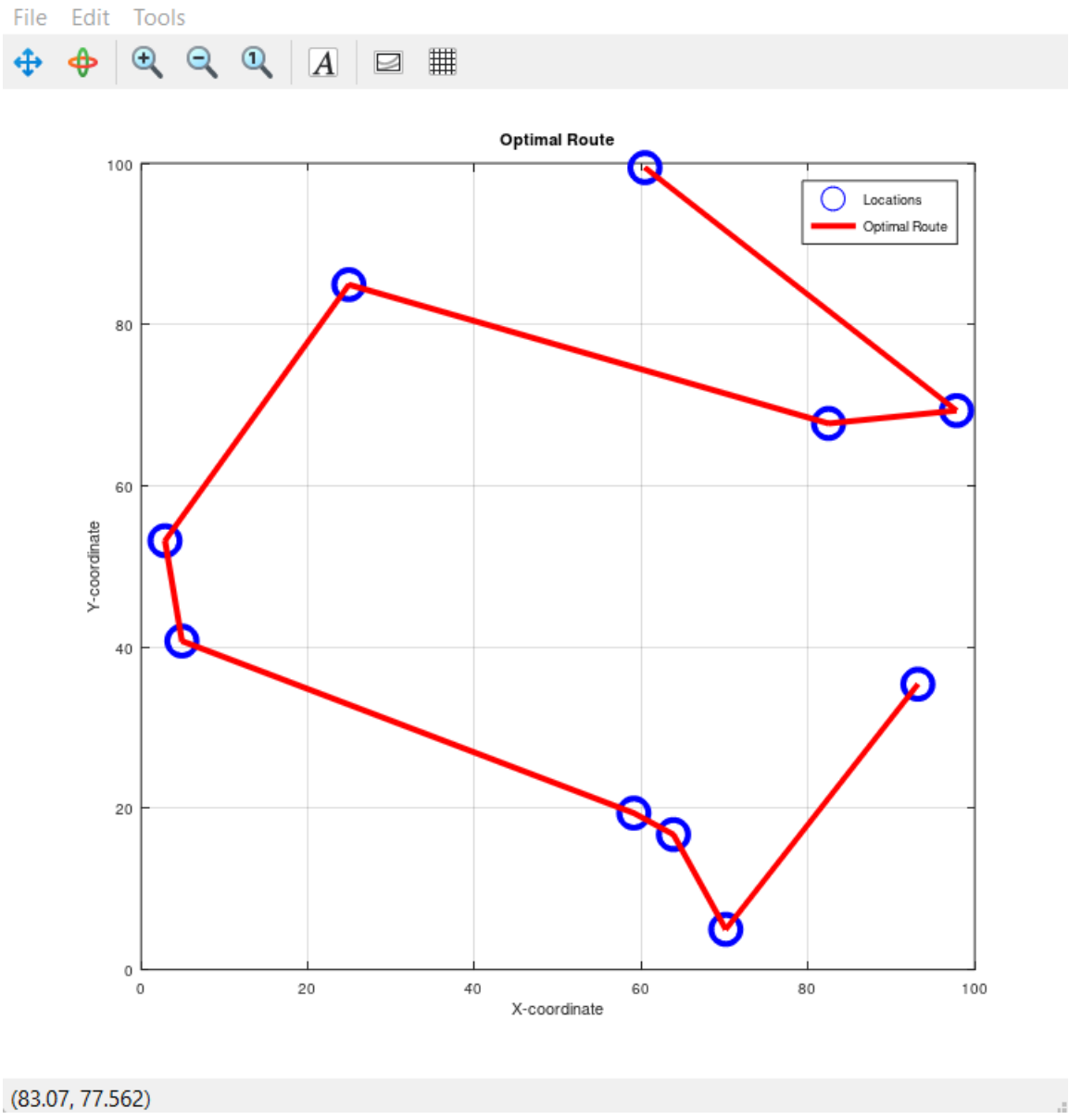
PROGRAM:

```
% Define coordinates of locations (in this example, randomly generated)
num_locations = 10;
coordinates = rand(num_locations, 2) * 100; % Generate random coordinates within a 100x100 grid

% Define initial asset information (e.g., randomly generated initial quantities and usages)
initial_assets = randi([10, 50], 1, num_locations); % Random initial quantities of assets at each location
initial_usages = zeros(1, num_locations); % Initial usages of assets at each location

% Define environmental conditions for each location (e.g., randomly generated)
environmental_conditions = rand(num_locations, 1); % Random environmental conditions

% Define initial route (e.g., starting from location 1 and visiting each location once)
initial_route = [1:num_locations 1];

% Function to calculate the total cost of a route
function cost = calculate_route_cost(route, coordinates)
   cost = 0;
   for i = 1:length(route)-1
      % Calculate distance between consecutive locations
      cost = cost + norm(coordinates(route(i), :) - coordinates(route(i+1), :));
   end
end
```

```
% Simulated Annealing Algorithm for Dynamic Route Optimization with Asset Tracking, Inventory Management, and Predictive Maintenance
T_initial = 100; % Initial temperature
T_final = 0.1; % Final temperature
cooling_rate = 0.99; % Cooling rate

T = T_initial;
current_route = initial_route;
current_cost = calculate_route_cost(current_route, coordinates);

while T > T_final
    % Generate a neighboring solution by randomly swapping two locations in the current route
    neighbor_route = current_route;
    idx1 = randi(num_locations);
    idx2 = randi(num_locations);
    neighbor_route([idx1, idx2]) = neighbor_route([idx2, idx1]);

    % Calculate cost of the neighboring solution
    neighbor_cost = calculate_route_cost(neighbor_route, coordinates);

    % Decide whether to accept the neighboring solution
    if neighbor_cost < current_cost || rand() < exp((current_cost - neighbor_cost) / T)
        current_route = neighbor_route;
        current_cost = neighbor_cost;
    end

    % Simulate predictive maintenance
    for i = 1:num_locations
        % Increase usage of assets at each location
        initial_usages(i) = initial_usages(i) + 1;

        % Check if maintenance is required based on usage threshold and environmental conditions
        usage_threshold = 100; % Threshold for usage count
        environmental_threshold = 0.5; % Threshold for environmental conditions
        if initial_usages(i) >= usage_threshold || environmental_conditions(i) > environmental_threshold
            fprintf('Maintenance required for assets at location %d\n', i);
            % Reset usage count after maintenance            initial_usages(i) = 0;
        end
    end

    % Cool down the temperature
    T = T * cooling_rate;
end

fprintf('Optimal Route: %s\n', mat2str(current_route));
```

```
fprintf('Optimal Cost: %.2f\n', current_cost);

% Plot the optimal route on a map
figure;
plot(coordinates(:, 1), coordinates(:, 2), 'bo', 'MarkerSize', 10, 'LineWidth', 2);
hold on;
plot(coordinates(current_route, 1), coordinates(current_route, 2), 'r-', 'LineWidth', 2);
xlabel('X-coordinate');
ylabel('Y-coordinate');
title('Optimal Route');
legend('Locations', 'Optimal Route');
grid on;
hold off;
```

OUTPUT:



RESULT:

Thus the design and simulation Enabled Supply Chain Optimization Challenge of using octave has been executed successfully and output was verified.

Exp. No.: 24	5G NETWORK PERFORMANCE MONITORING AND OPTIMIZATION CHALLENGE
Date:	

AIM:

To design and simulate Network Performance Monitoring and Optimization Challenge using octave.

APPARATUS REQUIRED:

Octave Simulation

PROCEDURE:

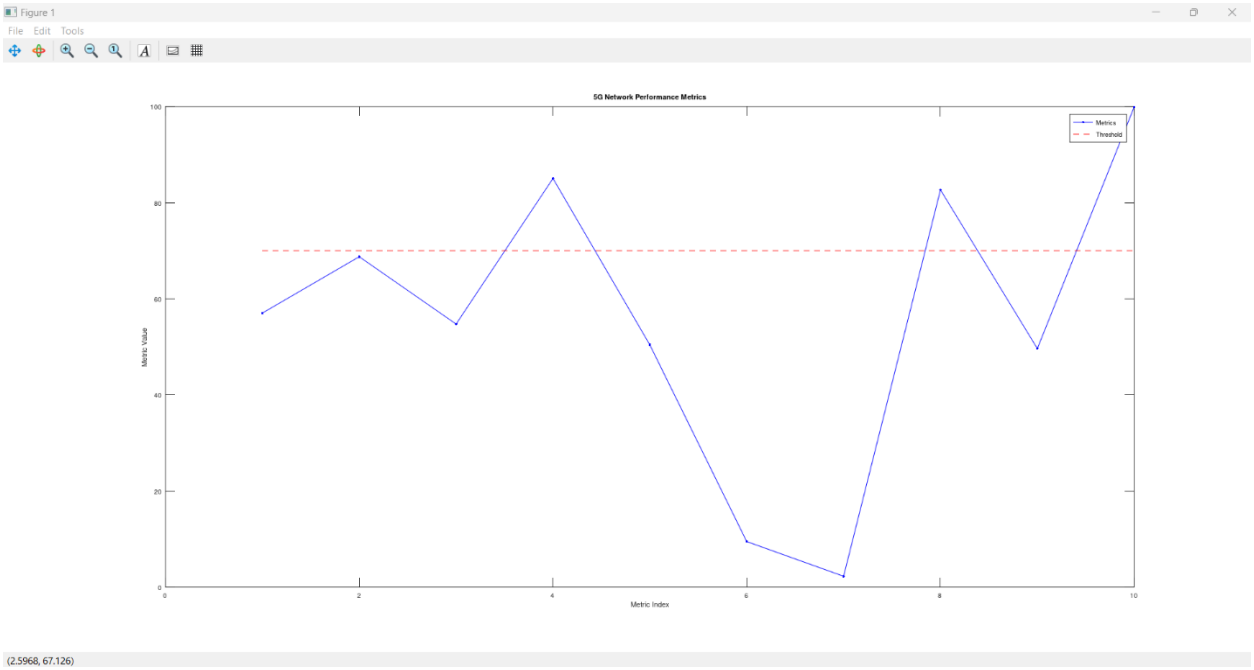
Step 1: Define key performance indicators (KPIs) for network performance monitoring.
Step 2: Collect network performance data using monitoring tools.
Step 3: Analyze collected data to identify performance bottlenecks and areas for improvement.
Step 4: Develop algorithms to optimize network performance based on analysis results.
Step 5: Implement optimized algorithms in the network infrastructure and further optimization.

PROGRAM:

```
function simulate_5G_network()
   % Generate random performance metrics
   metrics = rand(1, 10) * 100; % Example: 10 random metrics

   % Set thresholds
   threshold = 70; % Example: Threshold set to 70

   % Display metrics
   disp("Performance Metrics:");
   disp(metrics);

   % Plot metrics and threshold
   figure;
   plot(metrics, 'b.-');
   hold on;
   plot([1, length(metrics)], [threshold, threshold], 'r--');
   xlabel('Metric Index');
   ylabel('Metric Value');
   title('5G Network Performance Metrics');
   legend('Metrics', 'Threshold');
   hold off;
   % Check metrics against threshold
   for i = 1:length(metrics)
```

```
        if metrics(i) > threshold
            disp(['Alert: Metric ', num2str(i), ' crossed threshold at ', num2str(metrics(i))]);
            % Call troubleshooting function
            troubleshooting(i, metrics(i));
        end
    end
    % Propose optimization strategy
    disp("Optimization Strategy: Implement load balancing to distribute traffic evenly.");
end

function troubleshooting(metric_index, metric_value)
    % Simulate troubleshooting scenarios based on the metric crossing threshold
    switch metric_index
        case 1
            disp("Troubleshooting Scenario: High latency detected.");
            disp("Action: Check for interference, adjust antenna orientation.");
        case 2
            disp("Troubleshooting Scenario: High packet loss detected.");
            disp("Action: Check for network congestion, optimize routing.");
        otherwise
            disp("Troubleshooting Scenario: Unknown issue detected.");
            disp("Action: Perform general network health check.");
    end
end
% Main function call
simulate_5G_network();
```

OUTPUT:



RESULT:

Thus the design and simulation Network Performance Monitoring and Optimization Challenge of using octave has been executed successfully and output was verified.