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).

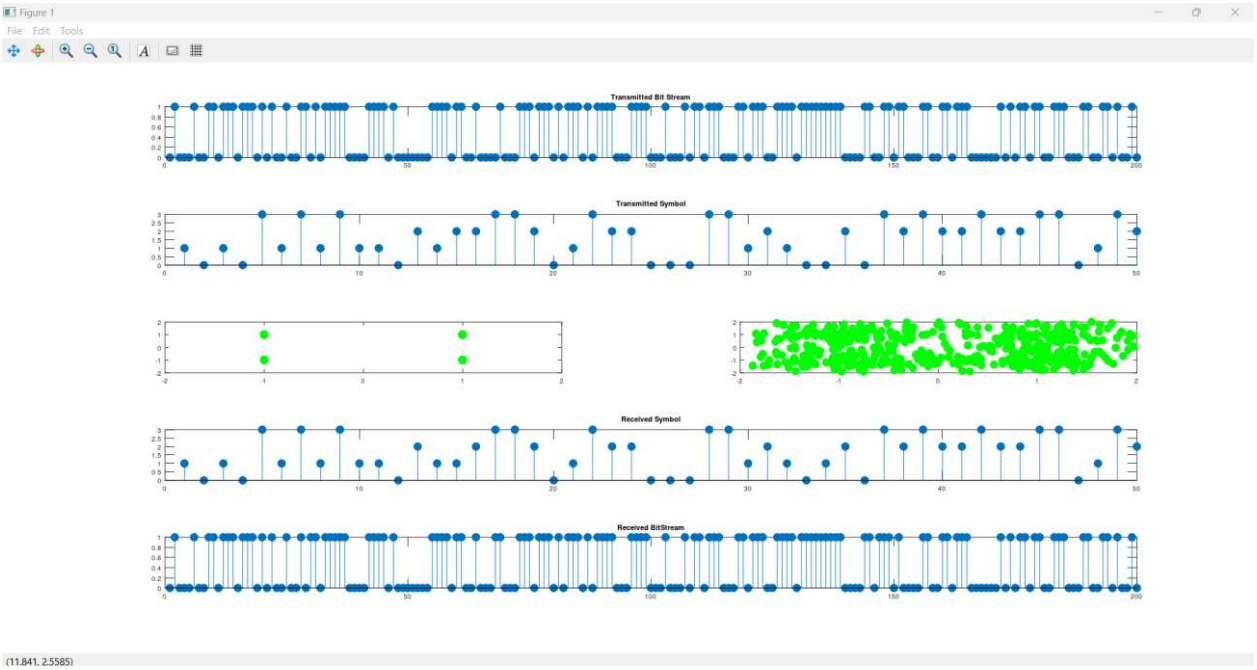
10. Visualization: Plot various stages of the modulation process including the transmitted bit stream, transmitted symbols, constellation diagram of the modulated and received signals, received symbols, and the received bit stream.

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.

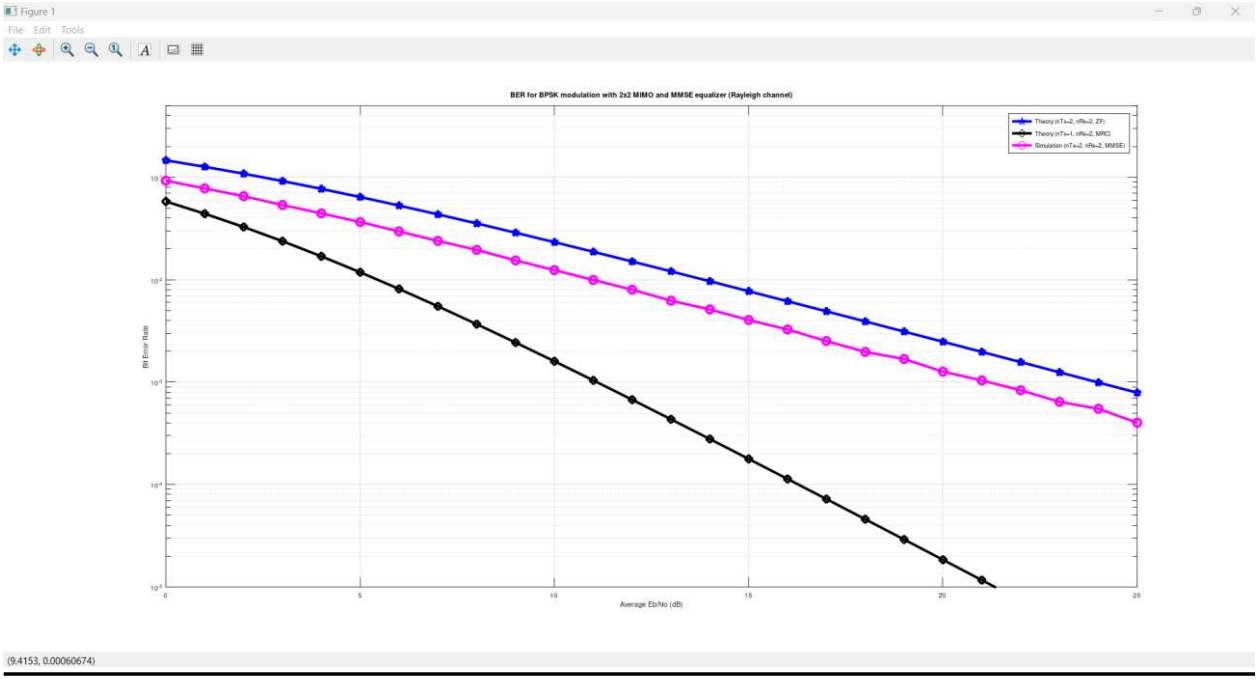
PROGRAM:

```
% 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").

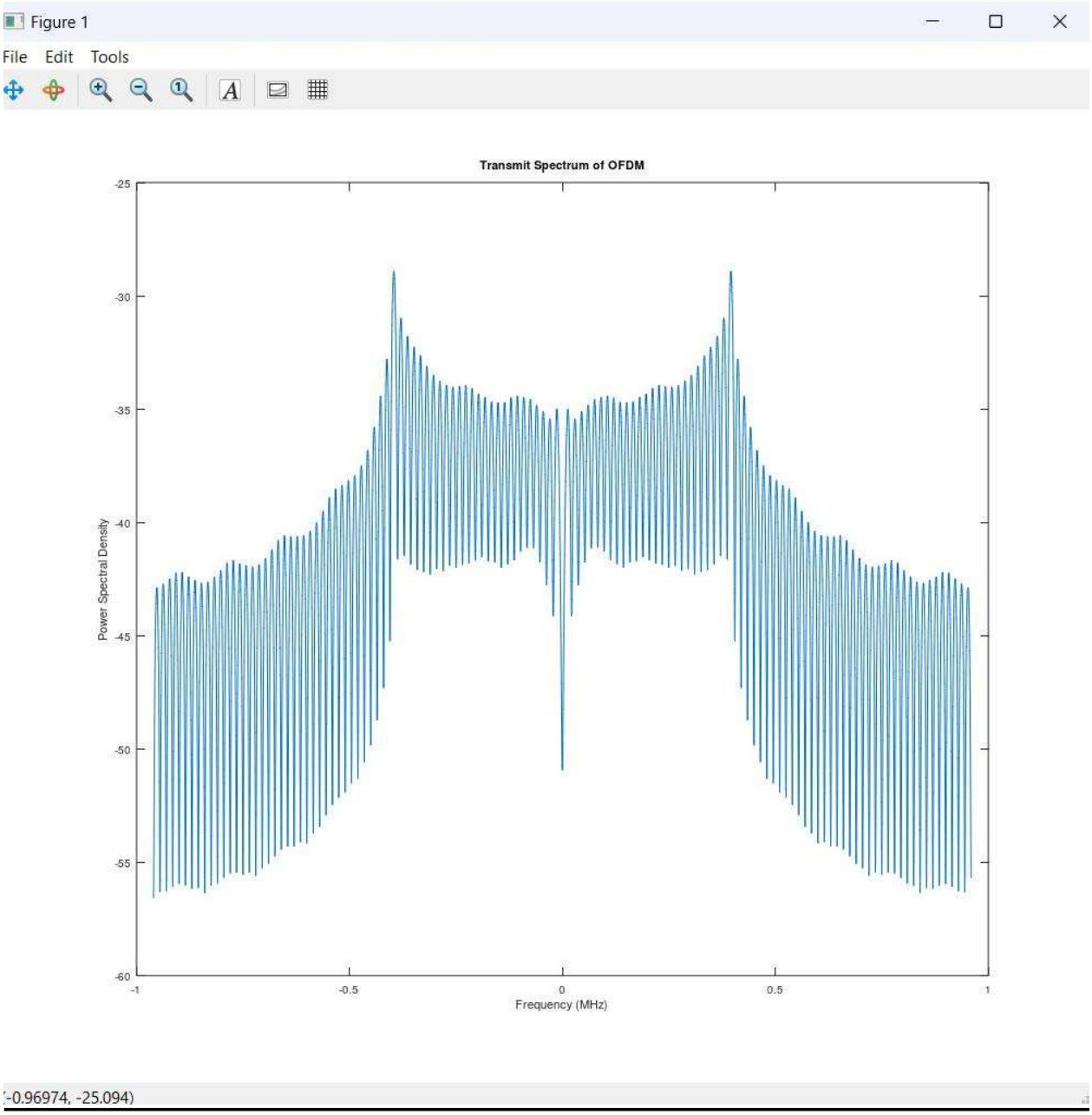
PROGRAM:

```
% 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.: 5	eMBB OPTIMIZATION WORKSHOP
Date:	

AIM:

To design and simulate eMBB Optimization Workshop using octave.

APPARATUS REQUIRED:

Octave Simulation

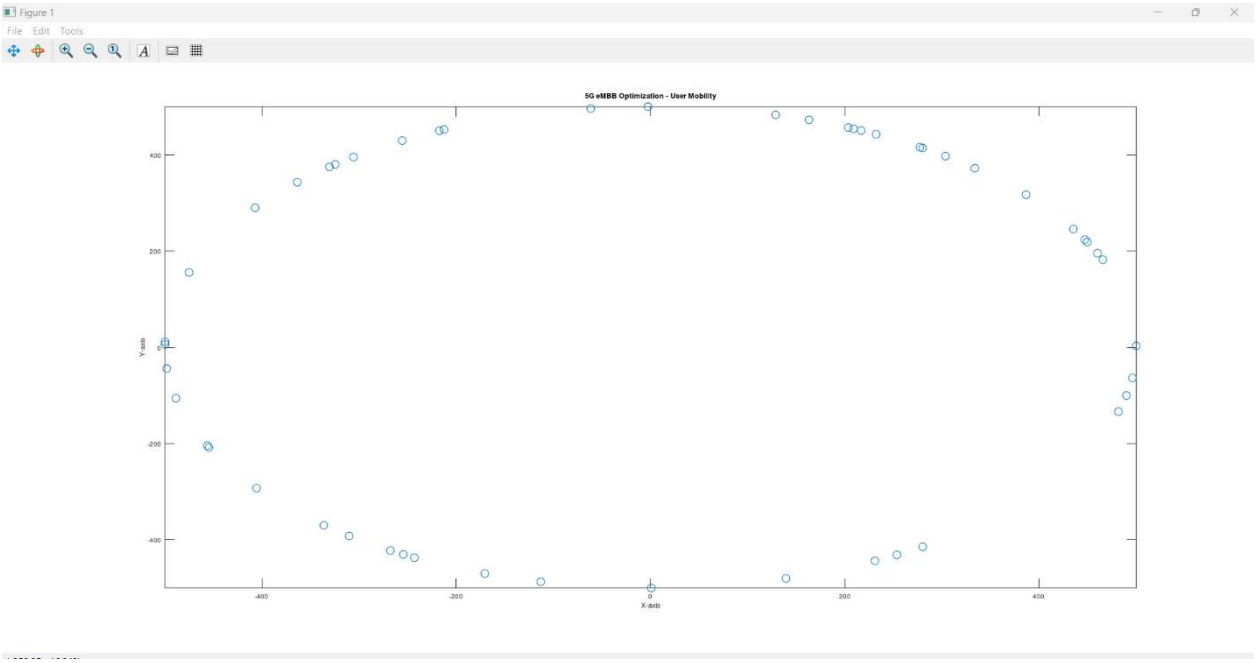
ALGORITHM:

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.

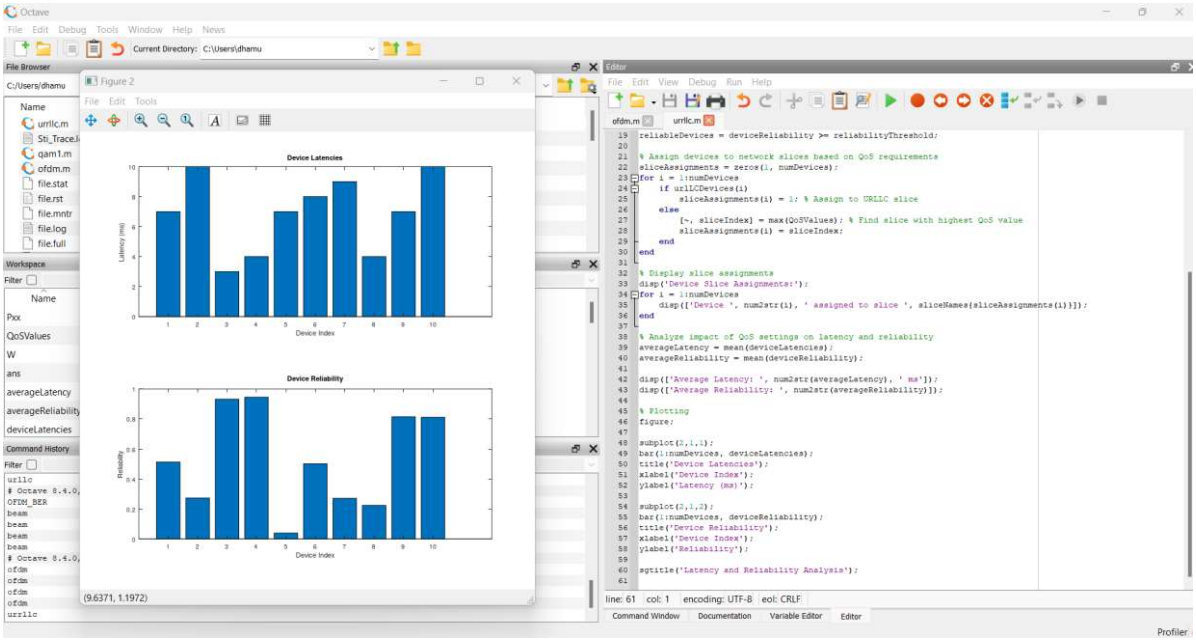
6. Plotting:
- Visualize latency and reliability distributions:
  - Plot device latencies and reliabilities using bar charts.
  - Display separate plots for latency and reliability.
  - Include appropriate axis labels, titles, and a title for the entire figure.

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.: 8	5G NETWORK SLICING WORKSHOP
Date:	

AIM:

To design and simulate NETWORK SLICING Workshop using octave.

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

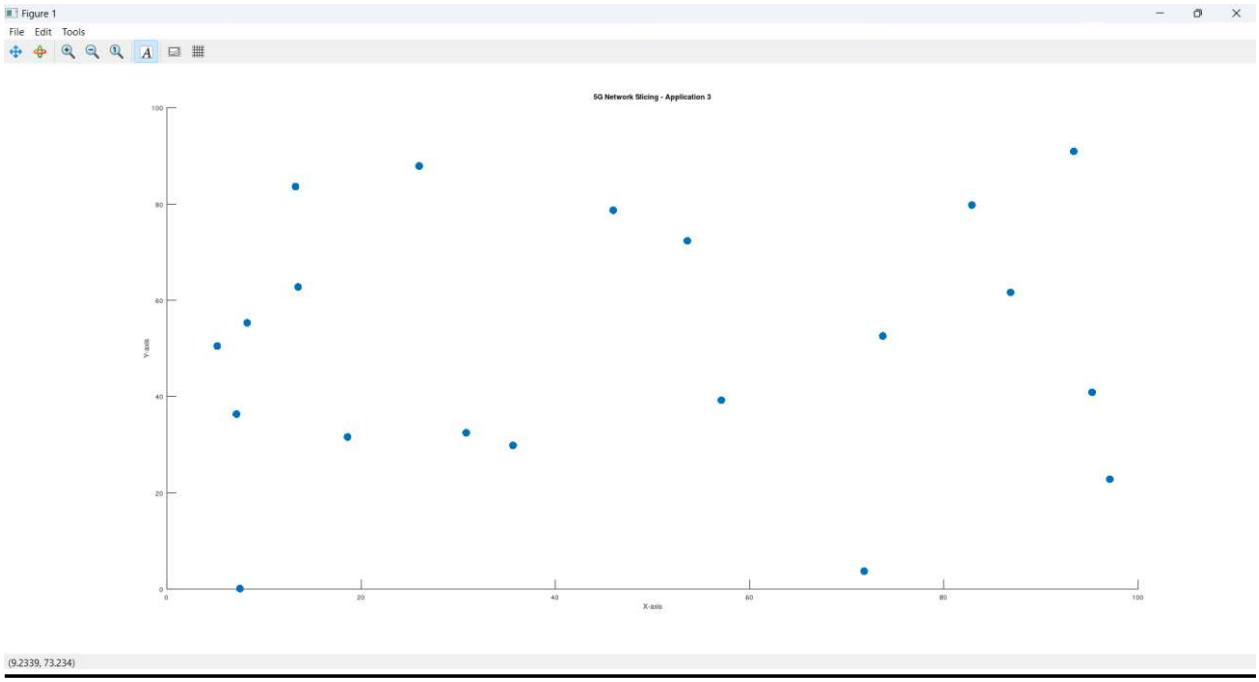
1. Parameter Definition:
   - Define parameters such as the number of diverse applications (`num_applications`) and the number of users per network slice (`num_users_per_slice`).
2. Network Slicing Simulation:
   - Generate random characteristics for each network slice associated with diverse applications:
     - `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.: 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:

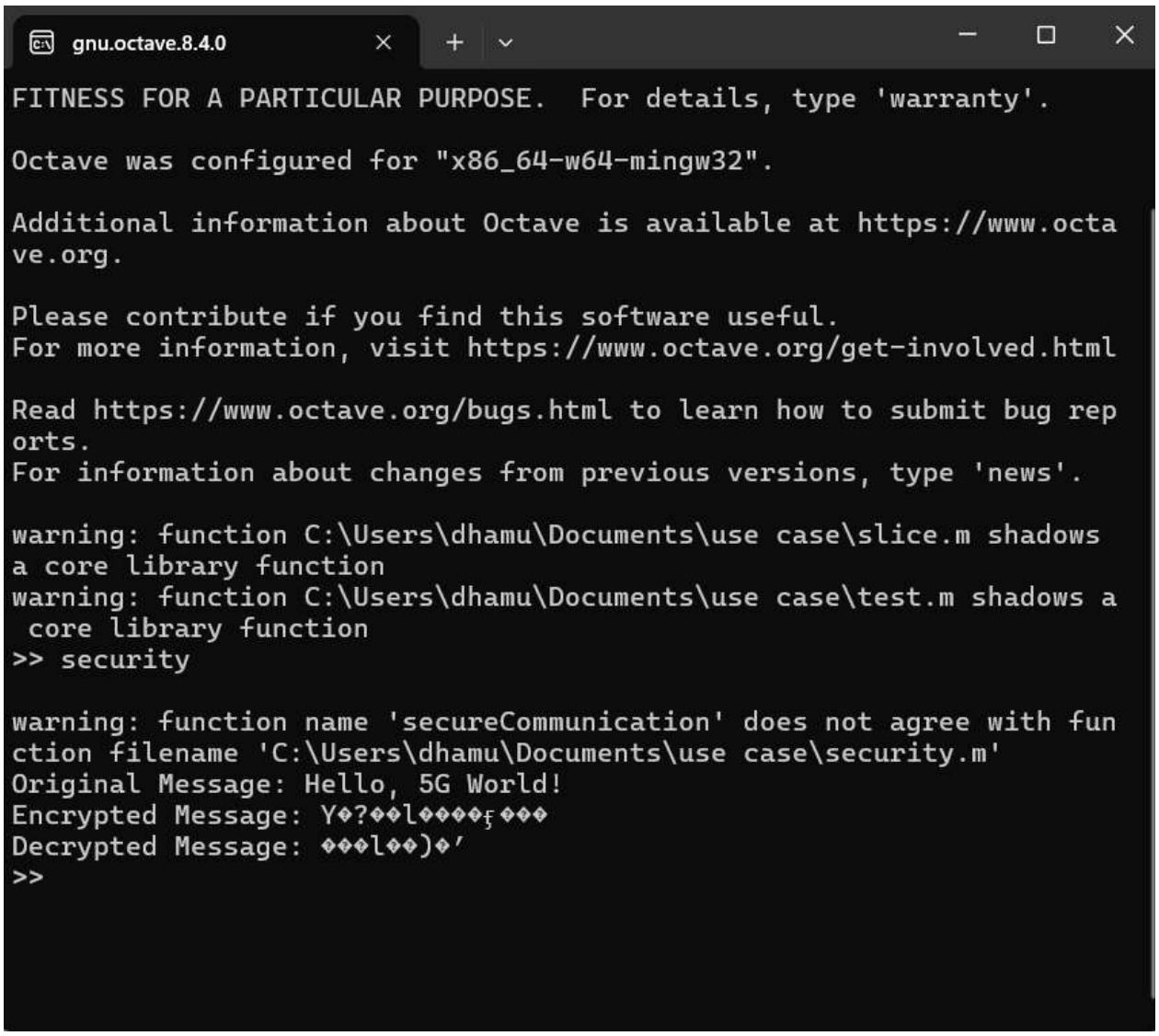
- Display the original message, encrypted message, and decrypted message to verify the security mechanisms.

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◆◆◆◆ƒ◆◆◆
Decrypted Message: ◆◆◆l◆◆)◆'
>>

RESULT:

Thus the design and simulation NETWORK SECURITY 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:

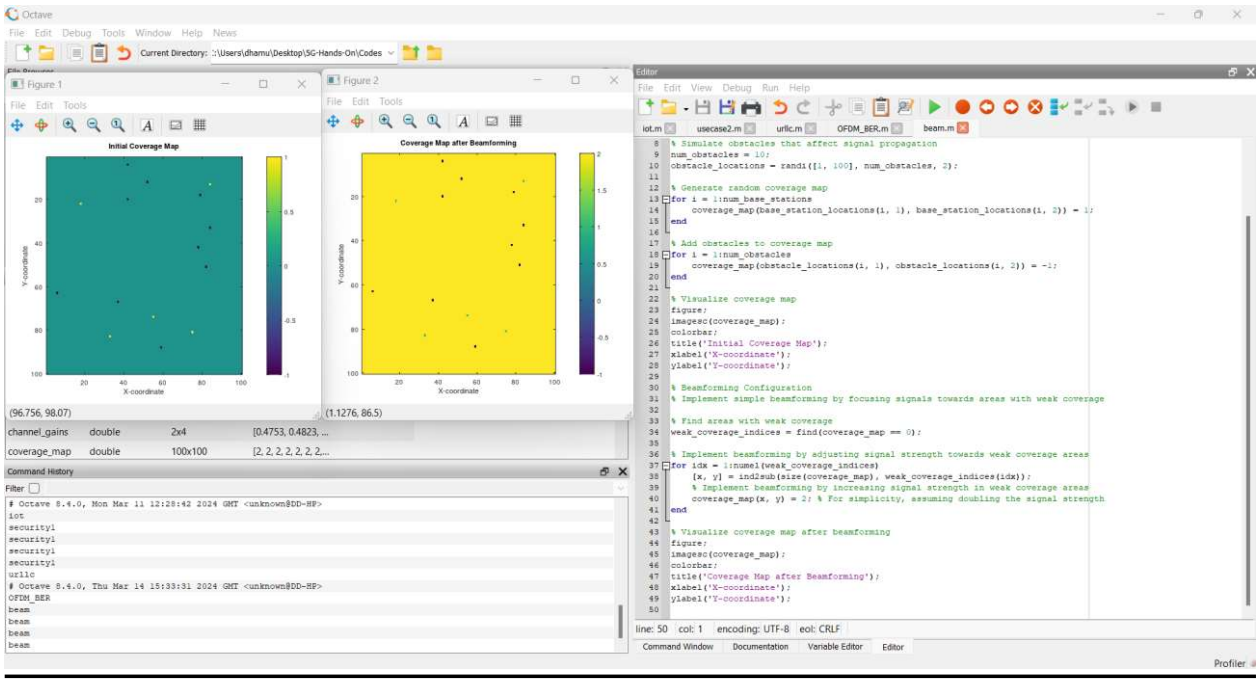
1. Initialize Simulation Parameters:
   - Set the number of base stations and define a 100x100 grid to represent the coverage map.
   - Randomly generate the locations for each base station and for obstacles that may affect signal propagation.
2. Setup Initial Coverage Map:
   - Mark the positions of 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:

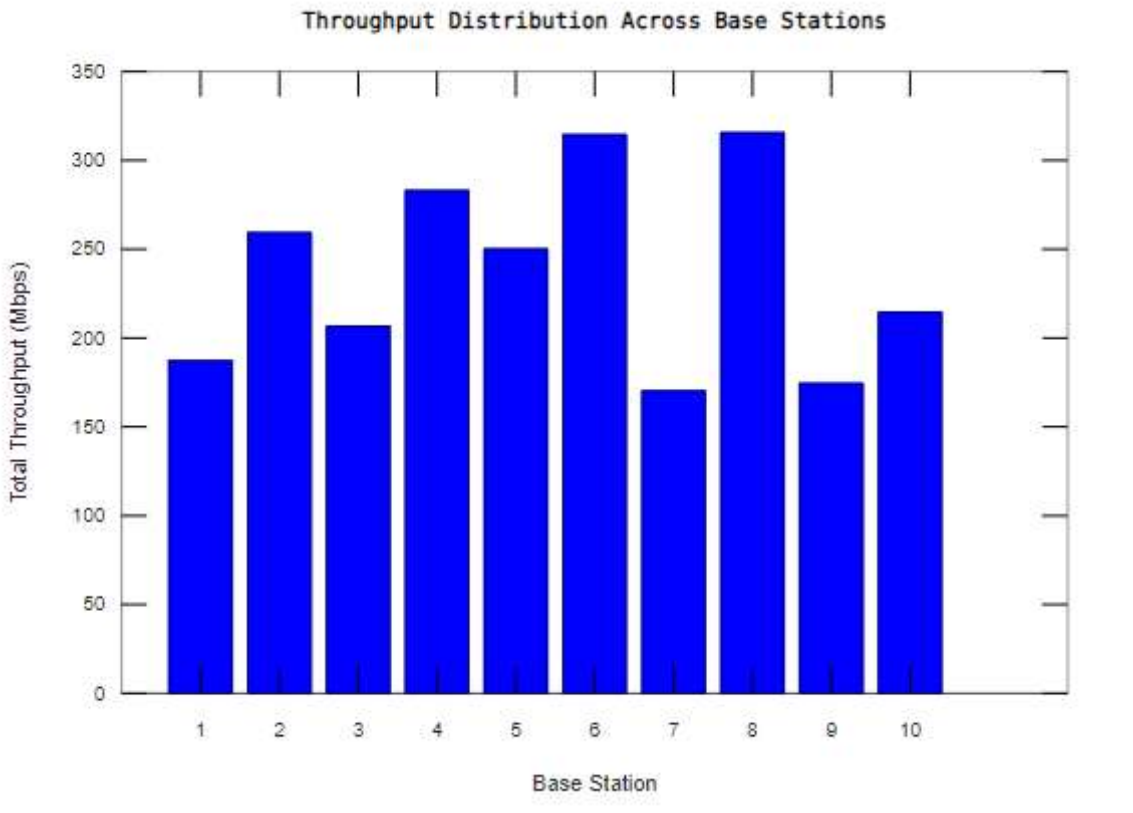
1. Define Network Parameters:
   - Establish the basic parameters required for the simulation, including latency requirements, bandwidth requirements, reliability threshold, and a scaling factor for future network expansion.
2. Setup Core Network Infrastructure:
   - Initialize the network configuration 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.: 17	5G TELEMEDICINE DEPLOYMENT CHALLENGE
Date:	

AIM:

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

APPARATUS REQUIRED:

Octave Simulation

ALGORITHM:

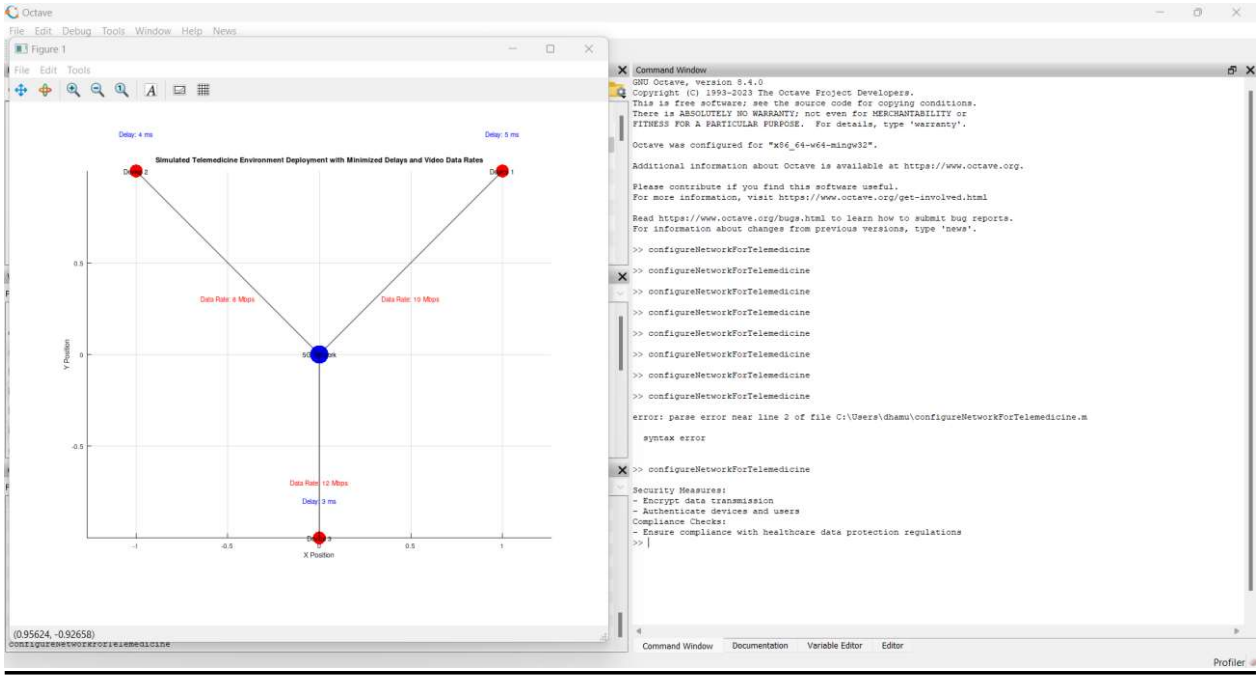
1. Define Network Infrastructure:
   - Set the coordinates of the network location (`network_location`) as the origin (0, 0).
   - Define the locations of medical devices (`medical_device_locations`) relative to the network location.
   - Specify the optimized transmission delays for each medical device (`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.: 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:

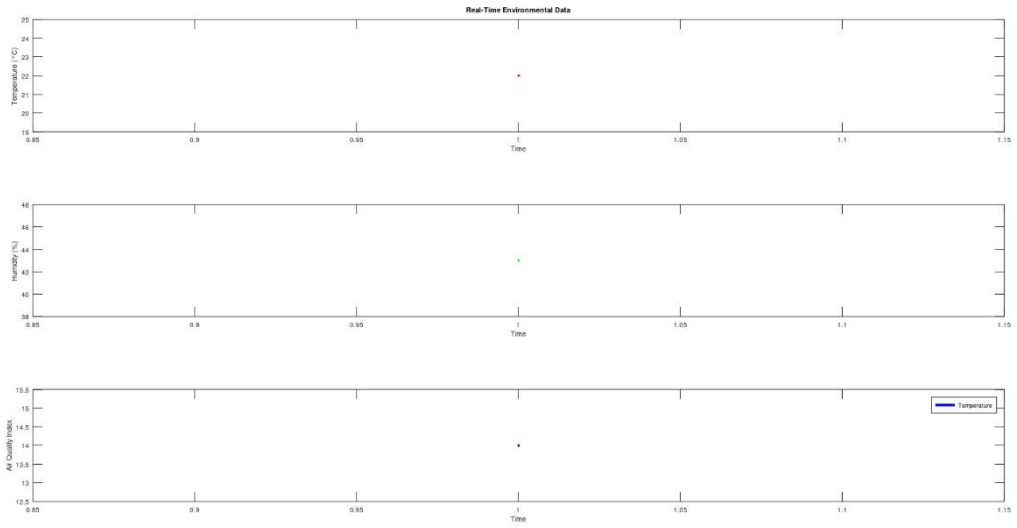
Step 1: Define the parameters and objectives for environmental monitoring with 5G.
Step 2: Set up the Octave environment for data collection and analysis.
Step 3: Develop algorithms to process and analyze environmental data in real-time.
Step 4: Implement sensors and communication protocols for data 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.: 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:

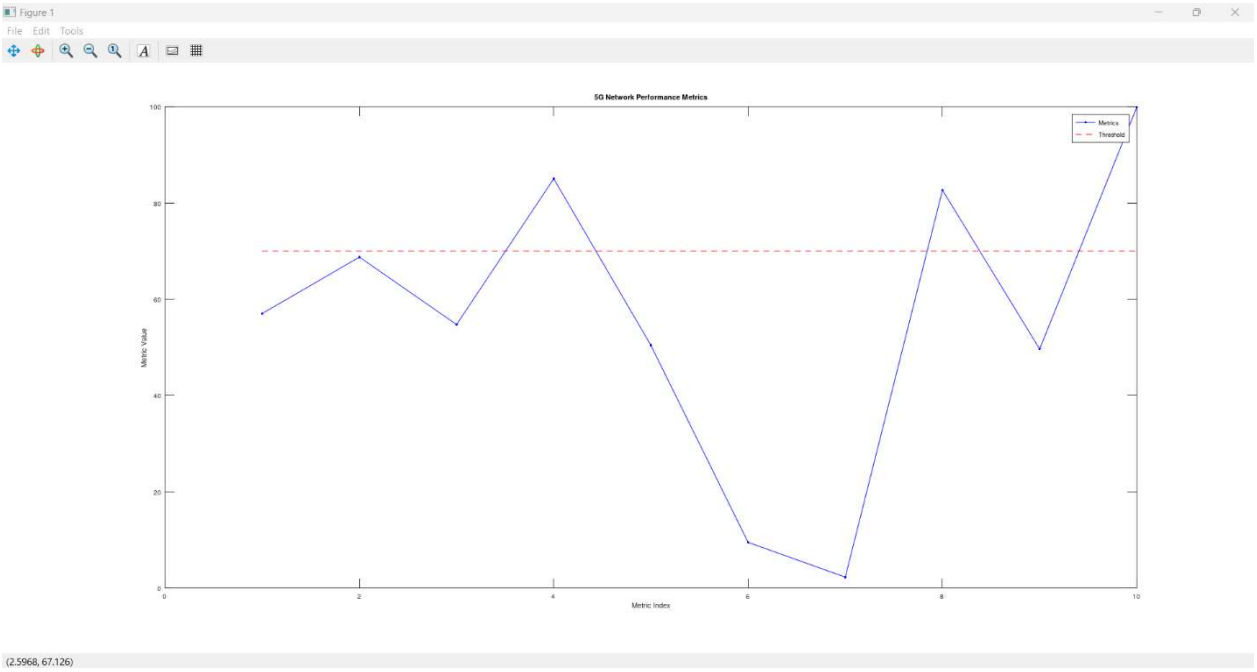
Step 1: Define key performance indicators (KPIs) for network performance monitoring.
Step 2: Collect network performance data using monitoring tools.
Step 3: Analyze collected data to identify performance bottlenecks and areas for improvement.
Step 4: Develop algorithms to optimize network performance 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.