**ST. ANNE’S COLLEGE OF ENGINEERING AND TECHNOLOGY**

Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai

Accredited by NAAC

ANGUCHETTYPALAYAM, PANRUTI – 607 106.

**Department of Electronics & Communication Engineering**

**CEC331 - 4G / 5G COMMUNICATION NETWORKS**

**LAB MANUAL**

**III YEAR / V SEMESTER**

**2021 REGULATION**

**Prepared by: D. UMAMAHESWARI, AP/ECE**

**LIST OF EXPERIMENTS**

**SIMULATION USING MATLAB**

1. 5G-Compliant waveform generation and testing

2. Modelling of 5G Synchronization signal blocks and bursts

3. Channel Modelling in 5G networks

4. Multiband OFDM demodulation

5. Perfect Channel estimation

6. Development of 5g New Radio Polar Coding

|  |  |
| --- | --- |
| **EXPT NO: 1** | **5G-COMPLIANT WAVEFORM GENERATION AND TESTING** |

**AIM:**

To perform the 5G- compliant waveform generation and testing in Matlab software.

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Setting Parameters for Carrier frequency, Sample rate, Number of samples in the waveform, Signal-to-noise ratio
2. Generate random binary data for QPSK modulation.
3. QPSK Modulation
4. Create a time vector based on the number of samples and sample rate
5. Combine the in-phase and quadrature components to form the QPSK signal.
6. Generate the carrier signal.
7. Modulate the QPSK symbols with the carrier signal to get the transmitted signal.
8. Add AWGN (Additive White Gaussian Noise) to the transmitted signal.
9. Divide the received signal by the carrier signal to perform demodulation.
10. Extract the phase information from the received signal.
11. Convert the demodulated symbols back to bits by comparing the phase.
12. Plot the transmitted signal, and received signal with noise, and the comparison of transmitted and decoded data.

**MATLAB CODE:**

clc;

clear all;

close all;

**% Parameters**

carrier-frequency = 3.5e9; **% Carrier frequency in Hz (e.g., 3.5 GHz for sub-6GHz 5G)**

Sample Rate = 30.72e6; **% Sample rate in Hz**

Num Samples = 1024; **% Number of samples in the waveform**

snr = 20; **% Signal-to-noise ratio in dB**

**% Generate a simple 5G waveform (QPSK modulation)**

data = randi([0, 1], 2, numSamples); **% Generate random bits for QPSK modulation**

qpskSymbols = 2 \* data - 1; **% Map bits to QPSK symbols (-1, 1)**

**% Create a time vector**

time = (0:numSamples - 1) / sampleRate;

**% Modulate the QPSK symbols**

qpskSignal = qpskSymbols(1, :) + 1j \* qpskSymbols(2, :);

**% Generate the carrier signal**

carrierSignal = exp(1j \* 2 \* pi \* carrierFrequency \* time);

**% Generate the transmitted signal**

transmittedSignal = qpskSignal .\* carrierSignal;

**% Add noise to the transmitted signal**

noisySignal = awgn(transmittedSignal, snr, 'measured');

**% Receiver**

received signal = noisy signal / carrier signal;

**% Demodulate the received signal**

demodulatedSymbols = angle(receivedSignal);

**% Decode the demodulated symbols back to bits**

decodedData = demodulatedSymbols> 0;

**% Plot the results**

subplot(3, 1, 1);

plot(time, real(transmittedSignal));

title('Transmitted Signal (I Component)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

plot(time, real(noisySignal));

title('Received Signal with Noise');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 3);

stem(data(:), 'rx');

hold on;

stem(decodedData(:), 'bo');

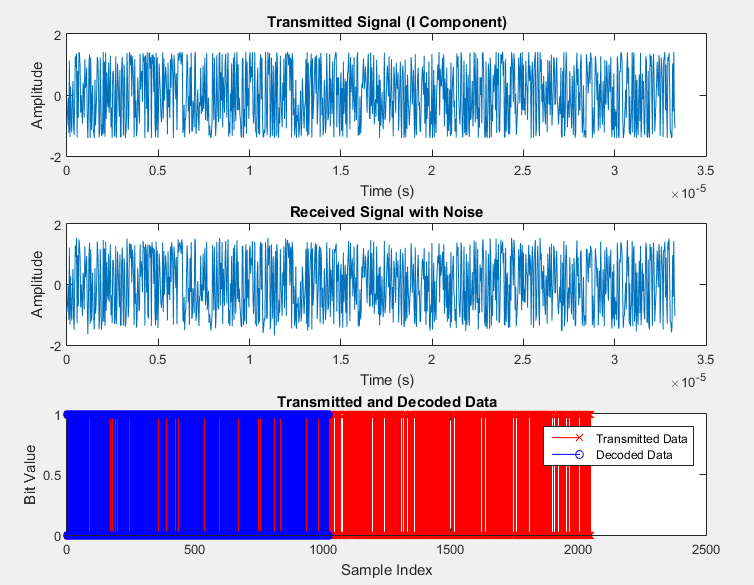
title('Transmitted and Decoded Data');

xlabel('Sample Index');

ylabel('Bit Value');

legend('Transmitted Data', 'Decoded Data');

**Output:**



**Result:**

Thus the 5G-Compliant Waveform Generation and Testing in MATLAB was successfully executed.

|  |  |
| --- | --- |
| **EXPT NO: 2** | **MODELING OF 5G SYNCHRONIZATION SIGNAL BLOCKS AND BURSTS** |

**AIM:**

To perform the Model 5G Synchronization Signal Blocks (SSBs) and Bursts in MATLAB software.

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Set Parameters for sampling frequency, cyclic prefix duration, sampling period
2. Generate Primary Synchronization Signal (PSS)
3. Generate Secondary Synchronization Signal (SSS)
4. Repeat the PSS and SSS sequences to construct the full synchronization signal burst.
5. Modulate the PSS and SSS sequences with the specified cyclic prefix duration to create the burst signals.
6. Visualize the amplitude of the PSS and SSS bursts
7. Generate a simulated received signal by adding noise to the sum of PSS and SSS bursts
8. Perform cross-correlation of the received signal with PSS and SSS bursts to detect synchronization
9. Visualize the correlation results for PSS and SSS

**MATLAB CODE:**

clc;

clear all;

close all;

**% Parameters**

fs = 30.72e6; **% Sampling frequency (Hz)**

Tc = 1/4.6e6; **% Cyclic Prefix duration (s)**

Ts = 1/30.72e6; **% Sampling period (s)**

**% Generate Primary Synchronization Signal (PSS)**

N\_id\_1 = 0; **% PSS identity (0 to 127)**

n = 0:127;

pss = exp(1j \* pi \* N\_id\_1 \* (n.\*(n+1)/2));

**% Generate Secondary Synchronization Signal (SSS)**

N\_id\_2 = 0; % SSS identity (0 or 1)

m = 0:31;

sss = exp(1j \* pi \* N\_id\_2 \* m);

**% Generate Burst**

n\_burst = 0:255; **% Burst duration in samples**

pss\_sequence = repmat(pss, 1, length(n\_burst)/length(pss));

sss\_sequence = repmat(sss, 1, length(n\_burst)/length(sss));

**% Construct full synchronization signal burst**

pss\_burst = pss\_sequence .\* exp(1j \* 2 \* pi \* n\_burst \* Tc / Ts);

sss\_burst = sss\_sequence .\* exp(1j \* 2 \* pi \* n\_burst \* Tc / Ts);

**% Plot the bursts**

figure;

subplot(2,1,1);

plot(n\_burst, abs(pss\_burst));

title('Primary Synchronization Signal Burst');

xlabel('Sample Index');

ylabel('Amplitude');

subplot(2,1,2);

plot(n\_burst, abs(sss\_burst));

title('Secondary Synchronization Signal Burst');

xlabel('Sample Index');

ylabel('Amplitude');

**% Correlation with received signal (for synchronization detection)**

received\_signal = awgn(pss\_burst + sss\_burst, 10); % Simulated received signal with noise

**% Correlation with PSS**

correlation\_pss = abs(xcorr(received\_signal, pss\_burst));

figure;

subplot(2,1,1);

plot(correlation\_pss);

title('Correlation with Primary Synchronization Signal');

xlabel('Sample Index');

ylabel('Correlation');

**% Correlation with SSS**

correlation\_sss = abs(xcorr(received\_signal, sss\_burst));

subplot(2,1,2);

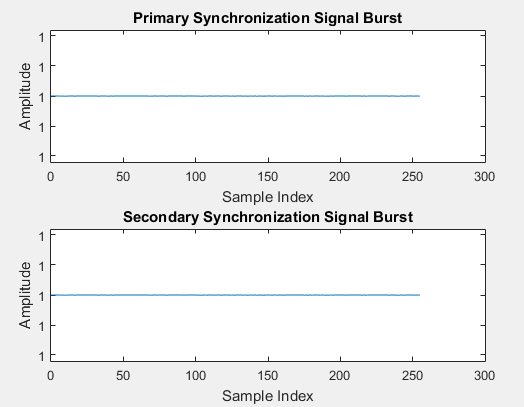
plot(correlation\_sss);

title('Correlation with Secondary Synchronization Signal');

xlabel('Sample Index');

ylabel('Correlation');

**Output:**



A diagram of a graph

Description automatically generated with medium confidence

**RESULT:**

Thus the Modeling 5G Synchronization Signal Blocks (SSBs) and Bursts using MATLAB was successful executed.

|  |  |
| --- | --- |
| **EXPT NO: 3** | **CHANNEL MODELLING IN 5G NETWORKS** |

**AIM:**

To simulate and analyze the propagation characteristics of wireless signals in 5G networks using MATLAB.

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Set simulation parameters.
2. Define channel impulse response and normalize it.
3. Generate random QPSK symbols for transmission.
4. Convolve QPSK symbols with channel impulse response.
5. Add AWGN noise to the received waveform.
6. Plot transmitted QPSK symbols and received waveform.

**MATLAB CODE:**

clc;

clear all;

close all;

**% Parameters**

num\_samples = 1000; **% Number of samples in the waveform**

snr\_db = 20; **% Signal-to-noise ratio in dB**

**% Generate a simple channel impulse response**

channel\_impulse\_response = [0.1, 0.5, 0.8, 0.5, 0.1]; **% Example channel coefficients**

**% Normalize the channel response**

channel\_impulse\_response = channel\_impulse\_response / norm(channel\_impulse\_response);

**% Generate a random QPSK waveform**

qpsk\_symbols = 2 \* (randi([0, 1], 1, num\_samples) - 0.5) +

1j \* (2 \* (randi([0, 1], 1, num\_samples) - 0.5));

**% Convolve the waveform with the channel impulse response**

received\_waveform = conv(qpsk\_symbols, channel\_impulse\_response);

**% Add AWGN noise**

noise\_power = 10^(-snr\_db / 10);

noise = sqrt(noise\_power / 2) \* (randn(size(received\_waveform)) + 1j \* randn(size(received\_waveform)));

received\_waveform = received\_waveform + noise;

**% Display the transmitted and received waveforms**

figure;

subplot(2, 1, 1);

plot(real(qpsk\_symbols), imag(qpsk\_symbols), 'o');

title('Transmitted QPSK Symbols');

xlabel('I (In-phase)');

ylabel('Q (Quadrature)');

grid on;

subplot(2, 1, 2);

plot(real(received\_waveform), imag(received\_waveform), 'x');

title('Received Waveform after Channel and Noise');

xlabel('Real Part');

ylabel('Imaginary Part');

grid on;

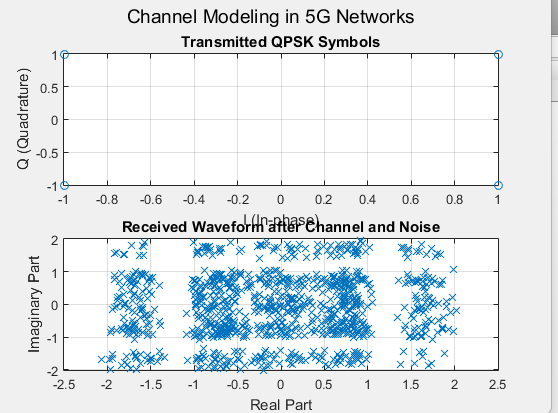
**% Add a title for the entire figure**

figure\_title = 'Channel Modeling in 5G Networks';

h = suptitle(figure\_title);

set(h, 'FontSize', 14);

**Output:**

****

**Result:**

Thus the Channel modelling in 5G Networks using MATLAB was successful executed.

|  |  |
| --- | --- |
| **EXPT NO: 4** | **MULTIBAND OFDM DEMODULATION** |

**AIM:**

To implement efficient and accurate demodulation of Multi-Band Orthogonal Frequency Division Multiplexing (MB-OFDM) signals using MATLAB.

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Set simulation parameters.
2. Generate Random QPSK Symbols
3. Perform IFFT
4. Add a cyclic prefix to the time-domain waveform.
5. Up sample the waveform to account for oversampling.
6. Generate and Add Noise to the Transmitted Waveform.
7. Down sample the received waveform to account for the oversampling
8. Remove the cyclic prefix from the received waveform.
9. Perform a Fast Fourier Transform (FFT) on the received waveform & Demodulate QPSK Symbols.

**MATLAB CODE:**

clc;

clear all;

close all;

**% Parameters**

num\_subcarriers = 64; **% Number of subcarriers**

num\_symbols = 100; **% Number of symbols**

oversampling\_factor = 4; **% Oversampling factor**

sampling\_rate = 1e6; **% Sampling rate in Hz**

symbol\_rate = 10e3; **% Symbol rate in Hz**

snr\_db = 20; **% Signal-to-noise ratio in dB**

**% Generate random QPSK symbols**

qpsk\_symbols = randi([0, 3], num\_subcarriers, num\_symbols);

qpsk\_symbols = exp(1j \* pi / 2 \* qpsk\_symbols);

**% Perform IFFT to generate time-domain waveform**

time\_domain\_waveform = ifft(qpsk\_symbols, num\_subcarriers) \* sqrt(num\_subcarriers);

**% Add cyclic prefix (CP)**

cp\_length = 16;

cp = time\_domain\_waveform(end - cp\_length + 1:end, :);

time\_domain\_waveform\_with\_cp = [cp; time\_domain\_waveform];

**% Upsample the waveform**

tx\_waveform = upsample(time\_domain\_waveform\_with\_cp, oversampling\_factor);

**% Generate AWGN noise**

noise\_power = 10^(-snr\_db / 10);

noise = sqrt(noise\_power / 2) \* (randn(size(tx\_waveform)) + 1j \* randn(size(tx\_waveform)));

**% Add noise to the transmitted waveform**

rx\_waveform = tx\_waveform + noise;

**% Downsample the received waveform**

rx\_waveform\_downsampled = downsample(rx\_waveform, oversampling\_factor);

**% Remove cyclic prefix**

rx\_waveform\_no\_cp = rx\_waveform\_downsampled(cp\_length + 1:end, :);

**% Perform FFT to obtain frequency-domain symbols**

rx\_freq\_symbols = fft(rx\_waveform\_no\_cp, num\_subcarriers);

**% Demodulate QPSK symbols**

rx\_qpsk\_symbols = angle(rx\_freq\_symbols);

**% Choose a subcarrier index for plotting (e.g., the first subcarrier)**

subcarrier\_index = 1;

**% Display the received symbols before and after demodulation**

subplot(2, 1, 1);

plot(real(rx\_freq\_symbols(:)), imag(rx\_freq\_symbols(:)), 'o');

title('Received Symbols (Frequency Domain)');

xlabel('Real Part');

ylabel('Imaginary Part');

grid on;

subplot(2, 1, 2);

**% Plot the demodulated QPSK symbols for the chosen subcarrier**

plot(1:num\_symbols, rx\_qpsk\_symbols(subcarrier\_index, :), 'o');

title(['Demodulated QPSK Symbols (Subcarrier ', num2str(subcarrier\_index), ')']);

xlabel('Symbol Index');

ylabel('Phase Angle (radians)');

grid on;

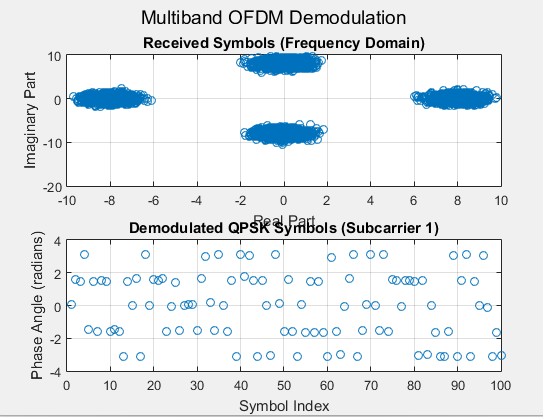
**% Add a title for the entire figure**

figure\_title = 'Multiband OFDM Demodulation';

h = suptitle(figure\_title);

set(h, 'FontSize', 14);

**Output:**

****

**Result:**

Thus the Demodulation of Multi-Band Orthogonal Frequency Division Multiplexing (MB-OFDM) signals using MATLAB was successful executed.

|  |  |
| --- | --- |
| **EXPT NO: 5** | **PERFECT CHANNEL ESTIMATION** |

**AIM:**

To achieve accurate and optimal channel estimation for 5G communication systems using MATLAB

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Set simulation parameters.
2. Generate Random Channel Matrix & Random Symbols
3. Modulate the symbols using QPSK modulation.
4. Transmit the modulated symbols through the generated channel matrix.
5. Add Noise to Received Symbols.
6. Perform Channel Estimation & Calculate Mean square error (MSE).
7. Plot the true and estimated channel matrices.

**Matlab Code:**

clc;

clear all;

close all;

**% Parameters**

num\_antennas = 4; **% Number of antennas at the transmitter / receiver**

num\_symbols = 100; **% Number of symbols**

snr\_db = 20; **% Signal-to-noise ratio in dB**

**% Generate random channel matrix**

H\_true = (randn(num\_antennas, num\_antennas) + 1i \* randn(num\_antennas, num\_antennas)) / sqrt(2);

**% Generate random symbols**

symbols = randi([0, 1], num\_antennas, num\_symbols);

**% Modulate symbols (e.g., using QPSK)**

modulated\_symbols = 2 \* symbols - 1;

**% Transmit symbols through channel**

received\_symbols = H\_true \* modulated\_symbols;

**% Add noise to received symbols**

noise\_power = 10^(-snr\_db / 10);

noise = sqrt(noise\_power/2) \* (randn(num\_antennas, num\_symbols) + 1i \* randn(num\_antennas, num\_symbols));

received\_symbols\_with\_noise = received\_symbols + noise;

**% Perform channel estimation using received and transmitted symbols**

estimated\_H = received\_symbols\_with\_noise \* pinv(modulated\_symbols);

**% Calculate mean squared error (MSE) of the estimated channel**

mse = mean(mean(abs(H\_true - estimated\_H).^2, 'omitnan'));

**% Display MSE**

disp(['Mean Squared Error: ', num2str(mse)]);

**% Plot the true and estimated channel matrices**

subplot(1, 2, 1);

imagesc(abs(H\_true));

colormap('hot');

colorbar;

title('True Channel Matrix');

xlabel('Transmit Antennas');

ylabel('Receive Antennas');

subplot(1, 2, 2);

imagesc(abs(estimated\_H));

colormap('hot');

colorbar;

title('Estimated Channel Matrix');

xlabel('Transmit Antennas');

ylabel('Receive Antennas');

**% Add a title for the entire figure**

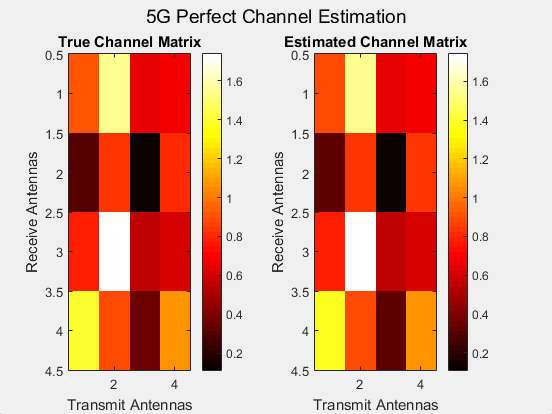
figure\_title = '5G Perfect Channel Estimation';

h = suptitle(figure\_title);

set(h, 'FontSize', 14);

**Output:**

Mean Squared Error: **0.00010273**



**Result:**

Thus the accurate and optimal channel estimation for 5G communication systems using MATLAB was successful executed.

|  |  |
| --- | --- |
| **EXPT NO: 6** | **DEVELOPMENT OF 5G NEW RADIO POLAR CODING** |

**AIM:**

To perform Polar Coding and Decoding using MATLAB.

**SOFTWARE USED:**

MATLAB

**PROCEDURE:**

1. Initializes the parameters for the polar code, including the length of information bits and CRC bits
2. Generate Random information bits are generated for transmission
3. Create Polar Code
4. Encode Information Bits
5. Add CRC Bits to the encoded codeword for error detection
6. Simulate Channel Errors by adding AWGN (Additive White Gaussian Noise) to the transmitted codeword.
7. Perform Polar Decoding to recover the information bits, considering the CRC for error checking.
8. Check CRC for Error Detection
9. Display transmitted and received information bits

**Matlab Code:**

clc;

clear all;

close all;

**% Set the parameters for polar coding**

infoLength = 16; **% Number of information bits**

crcLength = 8; **% Number of CRC bits**

**% Generate random information bits (0s and 1s)**

infoBits = randi([0, 1], infoLength, 1);

**% Create a polar code using the 5G NR code construction rules**

polarCode = nrPolarCode(infoLength, infoLength + crcLength);

**% Encode the information bits using the polar code**

codeword = nrPolarEncode(infoBits, polarCode);

**% Add CRC bits to the codeword**

crc = comm.CRCGenerator('Polynomial', 'z^8 + z^2 + 1');

crcBits = crc(codeword);

**% Simulate channel errors (for demonstration purposes)**

receivedCodeword = awgn(codeword, 10); % Add AWGN noise (10 dB SNR)

**% Perform polar decoding to recover the information bits**

decodedInfoBits = nrPolarDecode(receivedCodeword, polarCode, crc);

**% Check CRC to verify the correctness of the decoded information bits**

crcDetector = comm.CRCDetector('Polynomial', 'z^8 + z^2 + 1');

isCRCValid = crcDetector(receivedCodeword);

**% Display the results**

disp('Transmitted Information Bits:');

disp(infoBits.');

disp('Received Information Bits:');

disp(decodedInfoBits.');

if isCRCValid

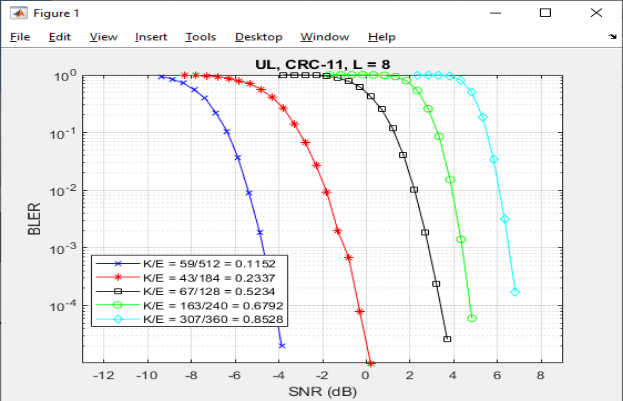
disp('CRC Check: Passed (Decoded information is correct).');

else

disp('CRC Check: Failed (Decoded information has errors).');

end

**Output:**

****

**Result:**

Thus the Polar Coding and Decoding using MATLAB was successful executed.