

SRM VALLIAMMAI ENGINEERING COLLEGE

**(An Autonomous Institution)
SRM NAGAR, KATTANKULATHUR – 603 203.**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**



LAB MANUAL

**CU3165 DIGITAL COMMUNICATION SYSTEMS
LABORATORY**

I-YEAR M.E /I-SEMESTER

ACADEMIC YEAR: 2025-2026 (ODD SEMESTER)

Prepared by

Mr. D. MURUGESAN, AP- Sel.G/ECE

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM Nagar, Kattankulathur -603 203

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION OF THE INSTITUTE

“Educate to excel in social transformation”

To accomplish and maintain international eminence and become a model institution for higher learning through dedicated development of minds, advancement of knowledge and professional application of skills to meet the global demands.

MISSION OF THE INSTITUTE

- To contribute to the development of human resources in the form of professional engineers and managers of international excellence and competence with high motivation and dynamism, who besides serving as ideal citizen of our country will contribute substantially to the economic development and advancement in their chosen areas of specialization.
- To build the institution with international repute in education in several areas at several levels with specific emphasis to promote higher education and research through strong institute-industry interaction and consultancy.

VISION OF THE DEPARTMENT

To excel in the field of electronics and communication engineering and to develop highly competent technocrats with global intellectual qualities.

MISSION OF THE DEPARTMENT

- To educate the students with the state of art technologies to compete internationally, able to produce creative solutions to the society's needs, conscious to the universal moral values, adherent to the professional ethical code
- To encourage the students for professional and software development career
- To equip the students with strong foundations to enable them for continuing education and research.

PROGRAMME OUTCOMES (POs)

- PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- PO2: Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- PO6: The Engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOMES (PSOs) of ECE DEPARTMENT

- PSO1:** Ability to apply the acquired knowledge of basic skills, mathematical foundations, and principles of electronics, modeling and design of electronics based systems in solving engineering Problems.
- PSO2:** Ability to understand and analyze the interdisciplinary problems for developing innovative sustained solutions with environmental concerns.
- PSO3:** Ability to update knowledge continuously in the tools like MATLAB, NS2, XILINIX and technologies like VLSI, Embedded, Wireless Communications to meet the industry requirements.
- PSO4:** Ability to manage effectively as part of a team with professional behavior and ethics.

OBJECTIVES:

- To study & measure the performance of digital communication systems.
- To explore the modulation schemes used in Wireless Communication.
- To learn about the design of digital filter.
- To enable the student to learn about adaptive filtering algorithms.
- To develop the student knowledge in error control codes.
- To design and conduct experiments on software defined radio.

LIST OF EXPERIMENTS:

1. Performance Evaluation of digital modulation schemes
2. Spread Spectrum communication system-Pseudo random binary sequence generation-Baseband DSSS.
3. Digital Filter Design.
4. CDMA signal generation and RAKE receiver design using MATLAB/ SCILAB/LABVIEW
5. Channel equalizer design – LMS
6. Channel equalizer design – RLS
7. OFDM transceiver design
8. Design and performance analysis of error control encoder and decoder (Block and Convolutional Codes)
9. Noise / Echo cancellation using MATLAB
10. Wireless channel characterization.

EXPERIMENT NO: 1

PERFORMANCE EVALUATION OF DIGITAL MODULATION SCHEMES

AIM

To analyze ASK, FSK and PSK modulation techniques using MATLAB.

THEORY

The performance evaluation of digital modulation schemes measures how effectively a system transmits data over a given channel, balancing spectral efficiency (data rate per bandwidth) with power efficiency (resilience to noise). Schemes are primarily evaluated using Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) curves.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
clc;
clear; close all;

% Parameters
N = 1e5;      % Number of bits
EbN0dB = 0:2:20; % Eb/No range in dB

% Generate random bit stream data = randi([0 1],
N, 1);
% BPSK Modulation

bpskMod = pskmod(data, 2); bpskBER =
zeros(size(EbN0dB));

for i = 1:length(EbN0dB)
    rxSig = awgn(bpskMod, EbN0dB(i), 'measured');
    rxData = pskdemod(rxSig, 2);
    [~, bpskBER(i)] = biterr(data, rxData); end
% QPSK Modulation
qpskMod = pskmod(data, 4, pi/4, 'gray'); qpskBER
= zeros(size(EbN0dB));
for i = 1:length(EbN0dB)
    rxSig = awgn(qpskMod, EbN0dB(i), 'measured');
    rxData = pskdemod(rxSig, 4, pi/4, 'gray');
```

```

[~, qpskBER(i)] = biterr(data, rxData); end
% 16-QAM Modulation

M = 16;
k = log2(M);
dataQAM = randi([0 M-1], N/k, 1);
qamMod = qammod(dataQAM, M, 'gray'); qamBER =
zeros(size(EbN0dB));

for i = 1:length(EbN0dB)
    rxSig = awgn(qamMod, EbN0dB(i), 'measured');
    rxData = qamdemod(rxSig, M, 'gray');
    [~, qamBER(i)] = biterr(dataQAM, rxData); end
% Plot BER Performance
semilogy(EbN0dB, bpskBER, 'r-o', 'LineWidth',
1.5); hold on; semilogy(EbN0dB, qpskBER, 'b-s',
'LineWidth', 1.5);
semilogy(EbN0dB, qamBER, 'g-d', 'LineWidth',
1.5); grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
title('Performance Comparison of Digital
Modulation Schemes'); legend('BPSK', 'QPSK', '16-
QAM', 'Location', 'southwest');

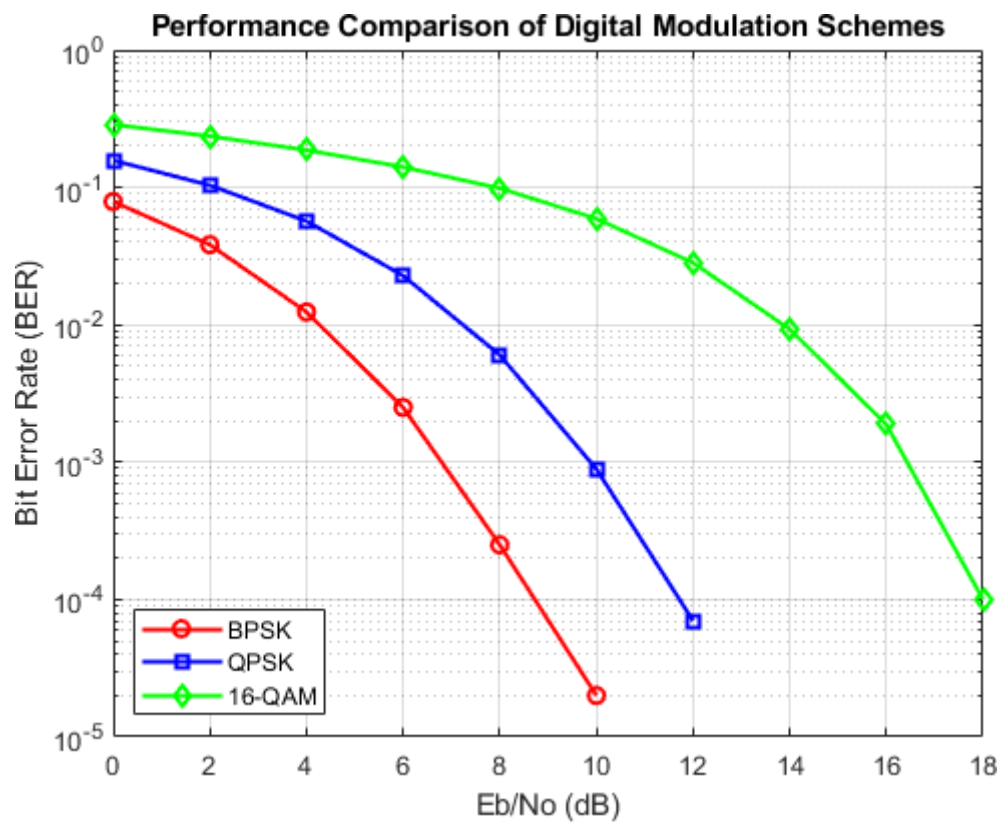
```

OUTPUT

ASK, FSK and PSK modulation techniques using MATLAB is observed and analyzed successfully.

RESULT

Thus the experiment 'Performance Evaluation of Digital Modulation Schemes' was implemented and verified successfully using MATLAB.



EXPERIMENT NO: 2

SPREAD SPECTRUM COMMUNICATION SYSTEM – PSEUDO RANDOM BINARY SEQUENCE GENERATION – BASEBAND DSSS

AIM

To generate PN sequence and implement DSSS system.

THEORY

Direct Sequence Spread Spectrum (DSSS) is a transmission technique that expands a signal's bandwidth far beyond the original message data rate. It achieves this by multiplying the baseband data with a distinct, deterministic Pseudo-Random Binary Sequence (PRBS). This ensures secure, interference-resistant communication.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
PRBS (PN sequence) generation + Baseband
spreading clc; clear; close all;
%PARAMETERS
N_bits = 10; % Number of data bits
m = 7; % PN length (m-sequence LFSR
size) poly_taps = [7 3]; % Feedback taps:
x^7 + x^3 + 1
%RANDOM DATA SEQUENCE

data_bits = randi([0 1], 1, N_bits); % Random
binary data data_bipolar = 2*data_bits - 1; %
Convert {0,1} → {-1,+1}
%PN SEQUENCE GENERATION (LFSR)
reg = [1 0 0 0 0 0 1]; % Initial state (cannot be all zeros)
Nchips = 2^m - 1; % PN sequence length =
127 pn_seq = zeros(1, Nchips);

for i = 1:Nchips
    pn_seq(i) = reg(end);
    feedback = xor(reg(m - (poly_taps(2)-1)),
    reg(end)); reg = [feedback reg(1:end-1)];
end

pn_bipolar = 2*pn_seq - 1; % Convert to bipolar chips
%DSSS SPREADING
spread_signal = [];

for i = 1:N_bits
    spread_signal = [spread_signal data_bipolar(i) *
    pn_bipolar]; end
%ADD NOISE (optional)
SNR = 10; % dB
rx = awgn(spread_signal, SNR, 'measured');
```

```

%DESPREADING
received_bits = zeros(1, N_bits);

for i = 1:N_bits
    chip_segment = rx((i-1)*Nchips + 1 :
        i*Nchips); corr_value = sum(chip_segment
        .* pn_bipolar); received_bits(i) =
        corr_value > 0; % Decision
end
% RESULTS
disp('Original    bits:');
disp(data_bits);

disp('Recovered   bits:');
disp(received_bits);
Original bits:
    0    1    0    0    1    1    1    0    1    0

Recovered bits:
    0    1    0    0    1    1    1    0    1    0

%PLOTS
figure;

subplot(3,1,1)
stem(data_bipolar,'filled'
); title('Original Bipolar
Data');

subplot(3,1,2)
stem(pn_bipolar(1:50),'filled');
title('PN Sequence (first 50 chips)');

subplot(3,1,3)
plot(spread_signal(1:300));
title('DSSS Spread Signal (first 300
samples)'); grid on;

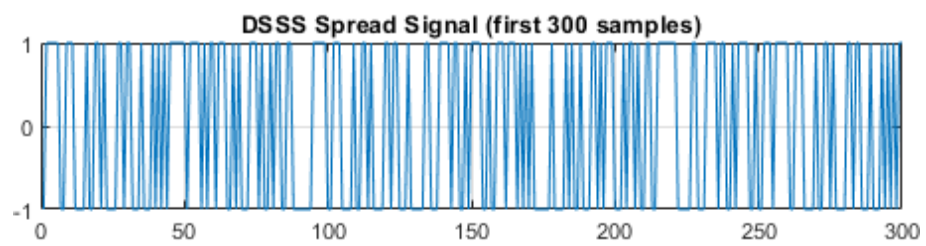
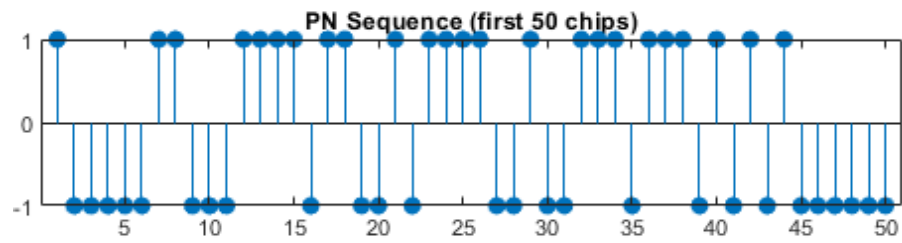
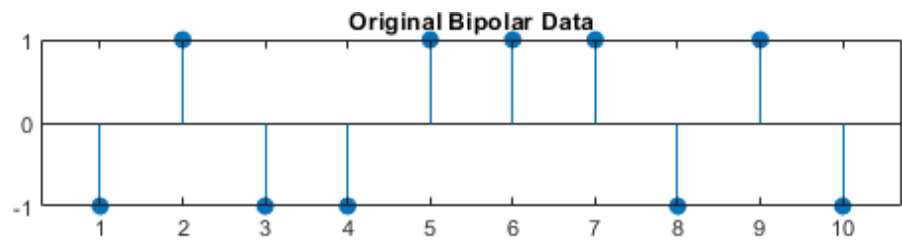
```

OUTPUT

The generation of PN sequence and implement DSSS system is observed and analyzed successfully.

RESULT

Thus the experiment 'Spread Spectrum Communication System – Pseudo Random Binary Sequence Generation – Baseband DSSS' was implemented and verified successfully using MATLAB.



EXPERIMENT NO: 3

DIGITAL FILTER DESIGN

AIM

To design FIR and IIR digital filters using MATLAB.

THEORY

Digital filters in MATLAB are designed and applied primarily using the Signal Processing Toolbox. The workflow involves two main steps: defining the filter specifications (using `designfilt`) to generate filter coefficients, and applying these coefficients to a signal using the `filter` function.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
% Digital Filter Design (FIR & IIR Example) clc; clear; close all;
Specifications
Fs = 1000;           % Sampling frequency (Hz) Fp = 150;           % Passband
frequency (Hz) Fsl = 200;           % Stopband frequency (Hz) Rp = 1;           % Passband
ripple (dB)
Rs = 40;           % Stopband attenuation (dB)
FIR Filter Design (Window Method)
N = fir1(60, Fp/(Fs/2), 'low', hamming(61)); % FIR Low-pass filter [H_fir, f_fir] =
freqz(N, 1, 1024, Fs);
IIR Filter Design (Butterworth Example)
[n, Wn] = buttord(Fp/(Fs/2), Fsl/(Fs/2), Rp, Rs); [b, a] = butter(n, Wn, 'low'); %
Low-pass IIR filter [H_iir, f_iir] = freqz(b, a, 1024, Fs);
Test Signal
t = 0:1/Fs:0.1;           % Time vector
x = cos(2*pi*50*t) + cos(2*pi*250*t); % Signal with 50 Hz + 250 Hz components

y_fir = filter(N, 1, x);           % Apply FIR filter y_iir = filter(b, a, x); %
Apply IIR filter Plot Frequency Responses
figure; subplot(2,1,1);
plot(f_fir, 20*log10(abs(H_fir)), 'LineWidth', 2); hold on; yline(-Rp, '--r'); yline(-
Rs, '--g');
title('FIR Low-pass Filter Response'); xlabel('Frequency (Hz)'); ylabel('Magnitude
(dB)'); grid on;
```

```

subplot(2,1,2);
plot(f_iir, 20*log10(abs(H_iir)), 'LineWidth', 2); hold on; yline(-Rp, '--r'); yline(-
Rs, '--g');
title('IIR Low-pass (Butterworth) Filter Response'); xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)'); grid on;

Plot Signal Before and After Filtering
figure; subplot(3,1,1);
plot(t, x); title('Original Signal (50 Hz + 250 Hz)'); xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot(t, y_fir); title('After FIR Filter (Low-pass)'); xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,3);
plot(t, y_iir); title('After IIR Filter (Low-pass)'); xlabel('Time (s)');
ylabel('Amplitude');

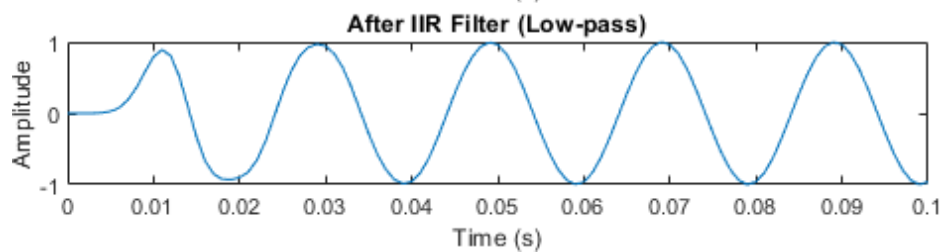
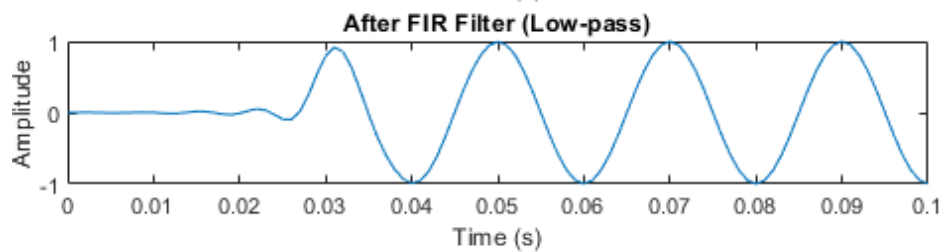
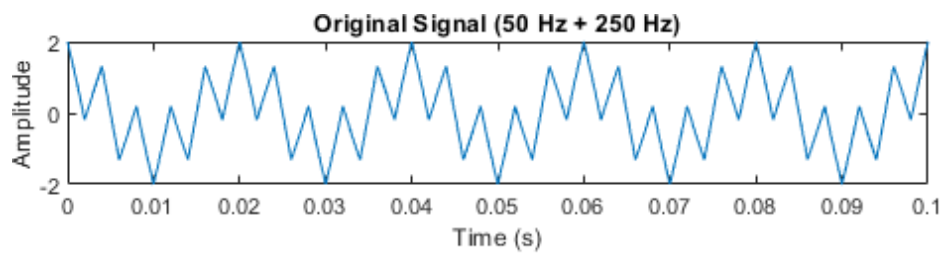
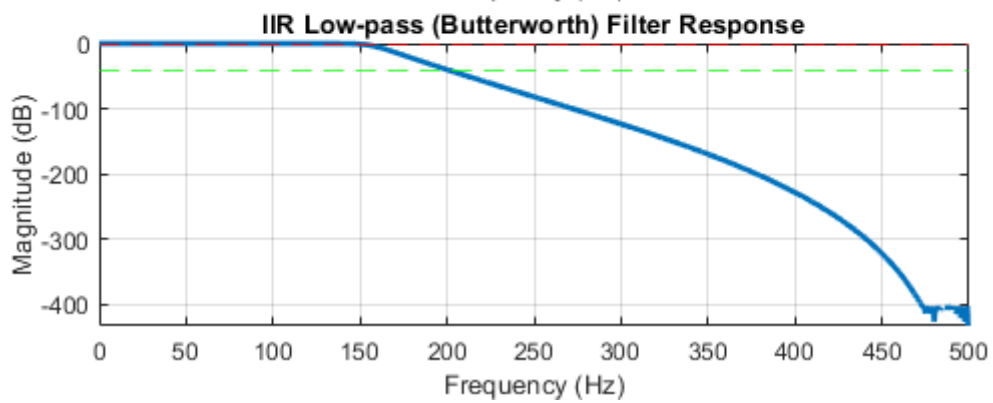
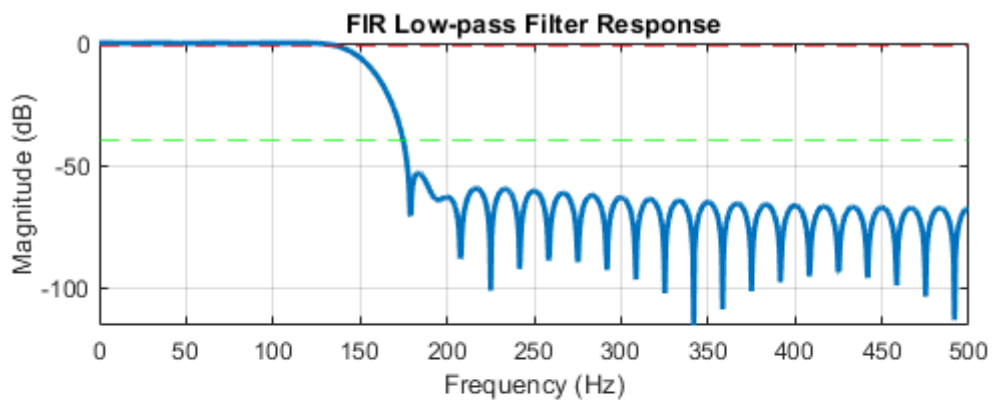
```

OUTPUT

The design FIR and IIR digital filters using MATLAB is observed and analyzed successfully.

RESULT

Thus the experiment 'Digital Filter Design' was implemented and verified successfully using MATLAB.



EXPERIMENT NO: 4

CDMA SIGNAL GENERATION AND RAKE RECEIVER DESIGN USING MATLAB / SCILAB / LABVIEW

AIM

To generate CDMA signals and design RAKE receiver.

THEORY

In Code Division Multiple Access (CDMA) systems, a RAKE receiver solves multipath fading by utilizing multiple correlators ("fingers") to independently process time-delayed copies of a transmitted signal, enhancing overall signal strength through Maximal Ratio Combining

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
clc;
% cdma_rake.m
% CDMA signal generation + RAKE receiver (baseband, BPSK)
% Save as cdma_rake.m and run in MATLAB R2024b

clear;
close all;
clc;
%PARAMETERS
numUsers = 4;      % total number of users (including
desiredUser = 1;    % index of user we
want to detect (1..numUsers) Nbits      = 2000;      %
bits per user
SF      = 31;      % spreading factor (chips per bit) - choose m-seq
length like 31 chipRate = 1;      % chip rate (arbitrary
unit)
EbN0_dB = 8;      % Eb/N0 (dB) for AWGN
fs = chipRate;    % sampling frequency in chips/sec (1 sample per chip)

% Multipath channel for desired user (relative delays in chips and complex gains)
% Example: 3-path channel with delays and complex
amplitudes path_delays_chips = [0, 1, 3];      % in
chips (integer delays)

path_gains = [1.0, 0.6*exp(1j*0.2), 0.4*exp(1j*-0.5)]; % complex gains

% RAKE
numFingers = length(path_delays_chips); % usually equals number of significant paths

% Other
```

```

rng(1); % seed for reproducibility
%UTILITY: m-sequence generator (LFSR)
Simple function to generate m-sequence of length (2^m
-1) function seq = mseq(taps, seed, len)
    % taps: vector of feedback tap positions (1-based, indexing from left MSB)
    % seed: binary seed vector of length m (non-zero)
    % len: desired output
    length m = length(seed);
    state =
    seed(:).'; seq
    = zeros(1,len);
    for i = 1:len
        % output bit is last
        state bit seq(i) =
        state(end);
        % feedback (XOR of tapped
        bits) fb = 0;
        for t = taps
            fb = xor(fb,
            state(t)); end
        % shift right and insert fb
        at start state = [fb,
        state(1:end-1)];
    end
end
%Generate spreading sequences (Gold-like)
Use two different m-seqs and XOR them to get Gold-like sequences
for users m = nextpow2(SF+1); % choose m such that 2^m-1 >= SF

m = max(m,5); % ensure reasonable m
(>=5) seqLen = 2^m - 1;
if seqLen < SF
    error('Choose larger SF or adjust m-seq
parameters.');
```

```

end
% choose simple taps for typical polynomials (for demonstration)
% these taps are example positions; better taps can be used for production
taps1 = [m, 1]; % primitive polynomial x^m + x + 1 (may not be primitive
for all m) taps2 = [m, m-1]; % another example

seed1 = ones(1,m); % non-zero seeds
seed2 = [1 0 ones(1,m-2)];

mseq1 = mseq(taps1, seed1, seqLen);
mseq2 = mseq(taps2, seed2, seqLen);

% Build spreading sequences for each user by circularly shifting the
XOR gold_base = xor(mseq1, mseq2); % base Gold-like sequence
spreadingCodes = zeros(numUsers,
SF); for u = 1:numUsers
    shift = mod(u-1, seqLen); % different
    phase shift seq_shifted =
    gold_base([1+shift:seqLen,1:shift]);
    spreadingCodes(u,:) = 2*seq_shifted(1:SF) - 1; % map 0->-1,1-
    >+1 (bipolar) end
%Generate user data and spread (BPSK)
data = randi([0 1], numUsers, Nbits);
data_bipolar = 2*data - 1; % map 0->-1, 1->+1

% Expand to chips: each bit -> SF chips (repmat or kron)
txSignalUsers = zeros(numUsers, Nbits*SF);

for u = 1:numUsers
    chipSeq = repmat(spreadingCodes(u,:), 1, Nbits); % chips
    sequence symbolChips = kron(data_bipolar(u,:), ones(1,SF)); %
    symbol mapped to chips txSignalUsers(u,:) = symbolChips .* chipSeq;
    % spread (BPSK on chips)
end

```

```

% Composite transmitted signal (summing
users) txComposite = sum(txSignalUsers,
1);

% For plotting convenience, convert to real (it
is real here) t_chips = (0:length(txComposite)-
1)/fs;
%Multipath channel & AWGN (only apply multipath to desired user)
Build desired user's tx signal and convolve with multipath (integer-
chip delays) desired_tx = txSignalUsers(desiredUser, :);

% Create channel impulse response at chip-rate
(discrete-time) maxDelay = max(path_delays_chips);
h = zeros(1, maxDelay+1);
for k = 1:length(path_delays_chips)
    h(path_delays_chips(k)+1) =
        path_gains(k);
end

% Convolve desired user's signal with
channel desired_tx_chan = conv(desired_tx,
h);

% For other users, assume either no multipath or same channel (here assume no
multipath)
% To simulate MAI fairly, other users transmitted directly (no multipath):
other_users = sum(txSignalUsers([1:desiredUser-1, desiredUser+1:end], :), 1);

% Align lengths: pad other_users to match conv
length L = length(desired_tx_chan);

if length(other_users) < L
    other_users = [other_users, zeros(1, L -
length(other_users))]; else
    other_users =
other_users(1:L); end

% Composite received signal is desired user's multipath + other
users + AWGN rx_no_noise = desired_tx_chan + other_users;

% Add AWGN based on Eb/N0
% Compute Eb from chip energy: assume 1 chip energy per chip: Eb = (chipEnergy *
SF) chipEnergy = mean(abs(txComposite).^2); % approximate chip energy (before
conv)
% Better: compute actual energy per bit for desired user after spreading
energy_per_bit = sum(abs(desired_tx).^2)/Nbits; % energy across
chips per bit Eb = energy_per_bit;
EbN0 = 10^(EbN0_dB/10);
noise_sigma = sqrt(Eb ./ (2*EbN0)); % one-sided? for real AWGN noise

% Since signal length changed with convolution, generate noise with same
length noise = noise_sigma * randn(1, L);
rx = rx_no_noise + noise;
%RAKE Receiver for desired user
We assume receiver knows path delays and gains approximately (or uses pilot). RAKE
correlates rx with delayed versions of desired user's spreading code.
% Build reference chip sequence for desired user aligned with tx
(length L) ref_chips = repmat(spreadingCodes(desiredUser,:), 1,
Nbits);
% After convolution, we need to align reference sequences for each finger:
% For finger with delay d: correlate rx with ref_chips shifted by d chips.

% Pre-allocate per-finger correlator outputs (one sample
per symbol) corrOutputs = zeros(numFingers, Nbits);

for f = 1:numFingers
    d = path_delays_chips(f); % chips
    % Build delayed reference: shift right by d (pre-pad with
zeros) ref_delayed = [zeros(1,d), ref_chips];

```

```

% Truncate or pad to match received length L
% ref_delayed = ref_delayed(1:L);
% Multiply received signal by ref_delayed (matched filter) and sum across chips per
bit
% Correlator: for each symbol k, compute dot(rx_segment,
ref_segment) for k = 1:Nbits
    idx_start = (k-1)*SF +
    1; idx_end = k*SF;
    if idx_end + d > L % if out of range because of convolution tail
        % safe indexing
        seg_rx = rx(idx_start: min(idx_end + d, L));
        seg_ref = ref_delayed(idx_start: min(idx_end
+ d, L)); else
        seg_rx = rx(idx_start + d : idx_end + d);
        seg_ref = ref_chips(idx_start: idx_end); % ref aligned without
extra zeros end
        corrOutputs(f,k) = sum(seg_rx .* seg_ref); % real-valued
correlation for BPSK end
end

% Combine fingers using Maximal Ratio Combining (use complex conjugate of estimated
path gains)
% Use known path_gains as combining weights (ideal MRC)
weights = conj(path_gains); % ideally proportional to channel complex conjugate
% Normalize weights so sum(weights^2) = 1 for SNR comparability
(optional) weights = weights / sqrt(sum(abs(weights).^2));

combined = weights(:).' * corrOutputs; % 1 x Nbits vector

% Decisions (BPSK): sign of
combined_decisions = real(combined)
> 0;
recovered_bits = double(decisions);

% Compute BER for desired user
BER = sum(recovered_bits ~= (data(desiredUser,:) == 1))/Nbits;
%PLOTS & RESULTS
fprintf('RAKE Receiver results:\n');
fprintf('Desired user: %d, Eb/N0 = %.1f dB, NumFingers = %d\n', desiredUser, EbN0_dB,
numFingers); fprintf('BER = %.5f\n', BER);

% Plot transmitted chips for first 4 bits
(desired user) figure;
subplot(3,1,1);
plot(1:4*SF, desired_tx(1:4*SF), '-o');
xlabel('Chip index'); ylabel('Amplitude'); title('Desired user transmitted chips
(first 4 symbols)'); grid on;

% Plot received composite (zoom into first 4 symbols)
subplot(3,1,2);
plot(1:4*SF, rx(1:4*SF), '-');
xlabel('Chip index'); ylabel('Amplitude'); title('Received signal
(first 4 symbols)'); grid on;

% Plot correlator outputs for first symbol across fingers subplot(3,1,3);
bar(1:numFingers, real(corrOutputs(:,1)));
xlabel('Finger index'); ylabel('Correlation value'); title('Per-finger correlator
outputs (symbol 1)'); grid on;

```

```

% Plot combined decision statistic (real part) for some symbols figure;
plot(1:200, real(combined(1:200)), '.-'); hold on; yline(0,'r--');
xlabel('Symbol index'); ylabel('Combined correlator (real)'); title('Combined RAKE
output (first 200 symbols)');
grid on;

% Plot BER summary text
figure('Name','BER','NumberTitle','off');
text(0.1,0.5, sprintf('RAKE BER for user %d = %.5f (Eb/N0 = %.1f dB)', desiredUser,
BER, EbN0_dB), 'FontSize', 14);
axis off;
RAKE Receiver results:
Desired user: 1, Eb/N0 = 8.0 dB, NumFingers = 3 BER = 0.00200
Warning: Imaginary parts of complex X and Y arguments ignored.
%OPTIONAL: Compare to single-finger (no RAKE)
Single-finger correlator using only path 1 single_corr = corrOutputs(1,:);
single_dec = real(single_corr) > 0;
BER_single = sum(single_dec ~= (data(desiredUser,')==1))/Nbits; fprintf('Single-
finger BER (path 1 only) = %.5f\n', BER_single);
Single-finger BER (path 1 only) = 0.01750
End of script

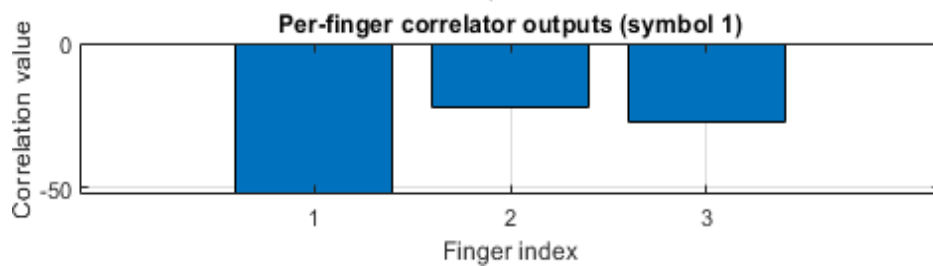
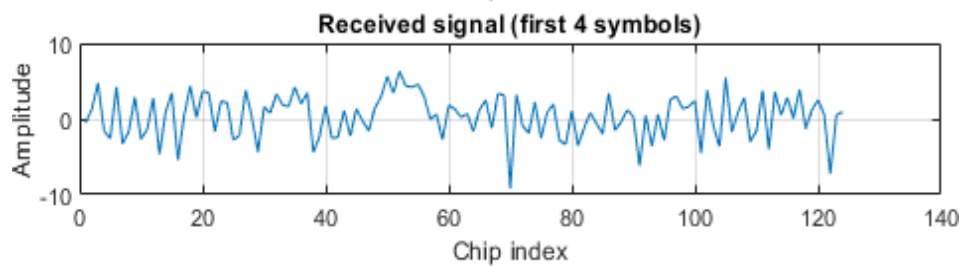
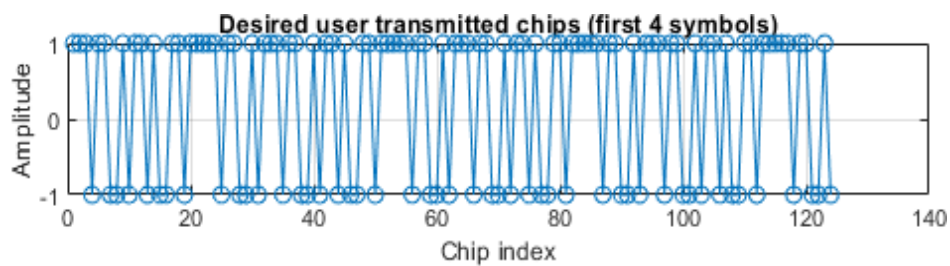
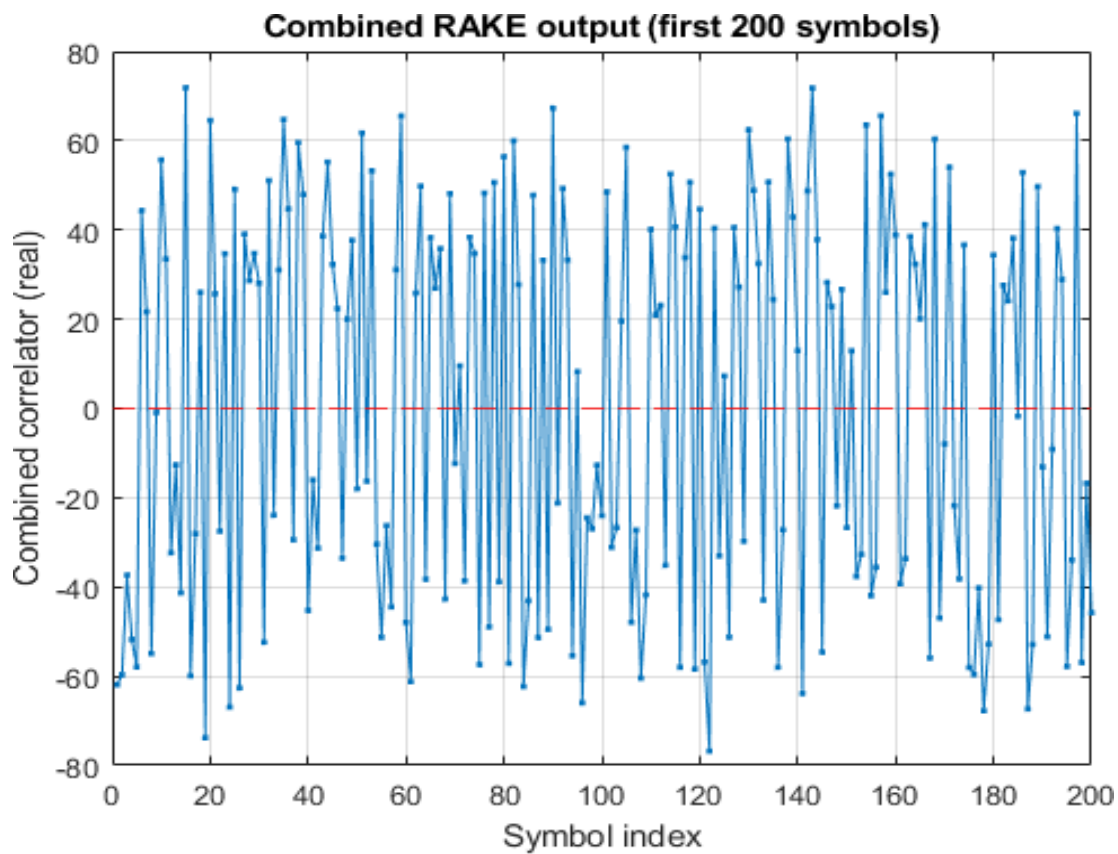
```

OUTPUT

The generation of CDMA signals and design RAKE receiver is observed and analyzed successfully.

RESULT

Thus the experiment 'CDMA Signal Generation and RAKE Receiver Design using MATLAB / SCILAB / LABVIEW' was implemented and verified successfully using MATLAB.



EXPERIMENT NO: 5

CHANNEL EQUALIZER DESIGN – LMS

AIM

To implement LMS adaptive equalizer.

THEORY

An LMS (Least Mean Square) adaptive equalizer is a self-adjusting digital filter designed to compensate for channel distortion (like multipath fading) in communication systems. It continuously updates its tap weights to minimize the error between the equalizer output and a known training sequence or reference signal

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
LMS Adaptive Equalizer for Channel Equalization clc; clear; close all;
Parameters
N = 5000;           % Number of symbols M = 16;           % Equalizer length
mu = 0.01;          % LMS step-size Generate BPSK data
data = randi([0 1], N, 1);
x = 2*data - 1;      % BPSK: 0->-1, 1->+1
Channel (Multipath) Example FIR channel: 3-path h = [0.9 0.3 -0.2]; % Channel
impulse response y = filter(h, 1, x); % Channel output (distorted)

% Add AWGN noise SNR = 20;
y_noisy = awgn(y, SNR, 'measured');
LMS Equalizer
w = zeros(M,1);      % Equalizer weights n0 = length(h);      % Decision delay
Ntrain = N - M;      % Training length

y_eq = zeros(Ntrain,1); e = zeros(Ntrain,1);
w_hist = zeros(M, Ntrain); % To visualize weight evolution

for n = 1:Ntrain

% Input vector to equalizer x_vec = y_noisy(n+M-1:-1:n);

% Equalizer output y_hat = w' * x_vec; y_eq(n) = y_hat;

% Desired training symbol (delayed) d = x(n + n0);

% Error
e(n) = d - y_hat;
```

```

% LMS update
w = w + mu * x_vec * conj(e(n));

w_hist(:, n) = w; end
Plots
% 1. Error convergence figure;
plot(abs(e));
xlabel('Iteration'); ylabel('|Error|'); title('LMS Error Convergence');

% 2. Equalizer weights convergence figure;
plot(w_hist);
xlabel('Iteration'); ylabel('Weight Value'); title('Equalizer Weight Evolution');

% 3. Before/After Equalization Comparison figure;
subplot(2,1,1);

plot(y_noisy(1:200));
title('Received Signal (Distorted)'); ylabel('Amplitude');

subplot(2,1,2);
plot(sign(y_eq(1:200)));
title('Equalized Signal (LMS Output)'); ylabel('Amplitude');

BER Calculation decisions = sign(y_eq);
decisions(decisions < 0) = 0; % Map -1 -> 0
decisions(decisions > 0) = 1; % Map +1 -> 1

BER = sum(decisions ~= data(1:length(decisions))) / length(decisions);

fprintf("\nLMS Equalizer BER = %.4f\n", BER); fprintf("Simulation Completed.\n");
LMS Equalizer BER = 0.4986 Simulation Completed.

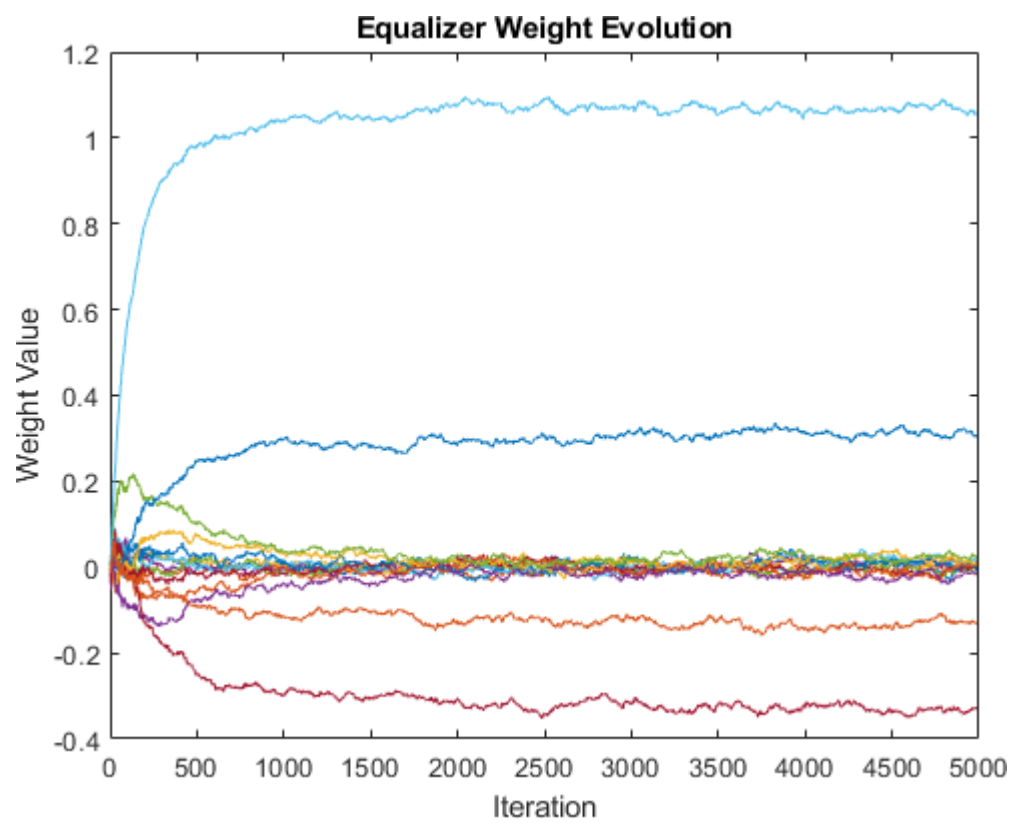
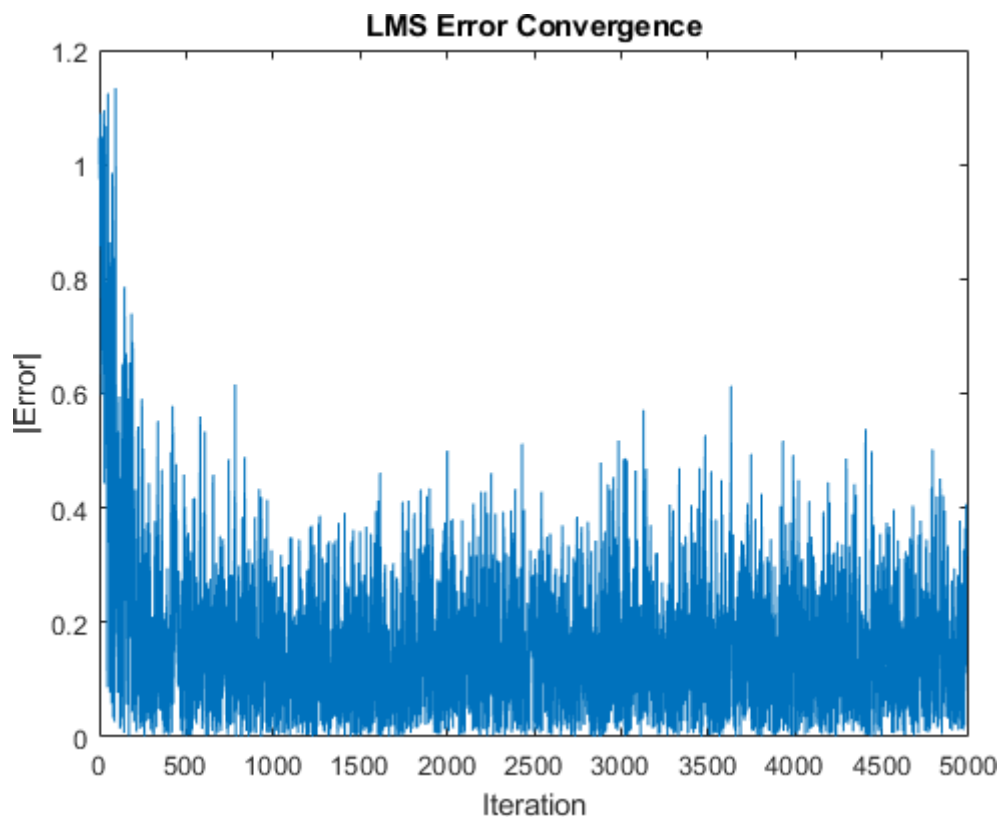
```

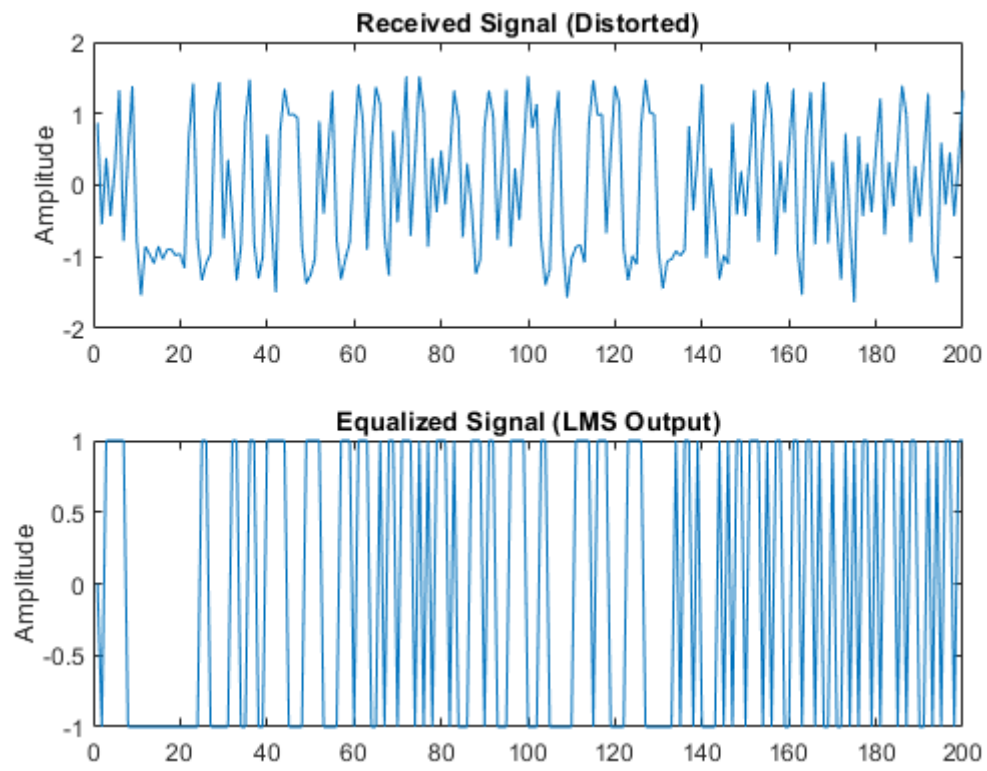
OUTPUT

The implementation of LMS adaptive equalizer is observed and analyzed successfully.

RESULT

Thus the experiment 'Channel Equalizer Design – LMS' was implemented and verified successfully using MATLAB.





EXPERIMENT NO: 6

CHANNEL EQUALIZER DESIGN – RLS

AIM

To implement RLS adaptive equalizer.

THEORY

A Recursive Least Squares (RLS) adaptive equalizer removes inter-symbol interference (ISI) in communication channels. It continuously updates the equalizer taps by minimizing the sum of squared errors over time, offering much faster convergence than traditional LMS methods.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
% rls_equalizer_demo.m
% Symbol-spaced RLS adaptive equalizer for QPSK symbols clear; close all; clc;
Parameters
Nsym = 5000;          % number of symbols M = 4; % QPSK
SNR_dB = 20; % channel SNR in dB
h = [0.9+0.0j, 0.45+0.15j, -0.2+0.1j]; % channel impulse response (example) eqLen =
7; % equalizer length (filter taps)
trainLen = 1000; % training length (symbols)
lambda = 0.995; % RLS forgetting factor (0.98-0.999 typical) delta = 1e-2;
% initialization P = (1/delta) * I
Generate QPSK symbols
Gray-coded QPSK, unit average power data = randi([0 M-1],Nsym,1);
symMap = exp(1j*(pi/4 + (pi/2)*data)); % QPSK: {pi/4,3pi/4,5pi/4,7pi/4} s =
symMap./sqrt(mean(abs(symMap).^2)); % normalize to unit power Pass through channel
and add noise
rx_full = conv(s, h); % channel
rx_full = rx_full(1:Nsym+length(h)-1); % full channel output length
% Add AWGN
SNR_lin = 10^(SNR_dB/10);
signal_power = mean(abs(rx_full).^2); noise_power = signal_power / SNR_lin;
noise = sqrt(noise_power/2) * (randn(size(rx_full)) + 1j*randn(size(rx_full))); r =
rx_full + noise;
Prepare for equalizer (build sliding input vectors)
We'll process symbol-by-symbol; pad received with zeros at beginning to form vectors
pad = eqLen-1;
r_padded = [zeros(pad,1); r]; % so first valid vector corresponds to r_padded(pad+1:-
1:1)

Nproc = Nsym; % number of symbol decisions (we'll handle indexing carefully)

% Pre-allocate
w = zeros(eqLen,1); % equalizer weights (column) P = (1/delta) * eye(eqLen);
% inverse correlation matrix y = zeros(Nproc,1); %
equalizer outputs
e = zeros(Nproc,1); % errors
mse = zeros(Nproc,1); % instantaneous error power
decisions = zeros(Nproc,1); % decision-directed outputs (symbols)
RLS loop: training then decision-directed for n = 1:Nproc
```

```

% Build input vector u(n) = [r(n); r(n-1); ... r(n-eqLen+1)] idx = pad + n;
u = r_padded(idx:-1:idx-eqLen+1); % column vector

% desired symbol: during training use transmitted s(n), after switch use decided
symbol if n <= trainLen
d = s(n); else
% decision-directed: use quantized previous output as desired
% We'll use previous decision as d if available; alternatively use tentative decision
of current y
% Here we use the decided symbol from previous iteration when n>trainLen.
% For simplicity we set desired to decision of current equalizer output (decision-
directed using hard decision).
% So compute yHat below first, then form d from quantizer.
% But RLS update requires a desired d; we compute y first using current w, then set
d via quantizer.
% (This is typical DD operation.)
% We'll compute y before calculating e below. end

% equalizer output (prior to update)

y_n = w' * u; y(n) = y_n;

if n > trainLen
% hard-decision: QPSK mapping nearest constellation point
% map to same constellation points used in transmitter angles = angle(y_n);
% Map angles to nearest QPSK points: boundaries at pi/4 + k*pi/2
% quantize to {pi/4, 3pi/4, -3pi/4, -pi/4}
% Alternatively, use sign of real and imag
d = (sign(real(y_n)) + 1j*sign(imag(y_n))) / sqrt(2); else
d = s(n); end

% compute error en = d - y_n; e(n) = en;
mse(n) = abs(en)^2;

% RLS gain vector Pu = P * u;
denom = lambda + (u' * Pu); k = Pu / denom;

% weight update w = w + k * en';

% P update
P = (P - k * (u' * P)) / lambda;

% store decision (for plotting)
% For training period, we can still form a hard decision for plotting if n <= trainLen
decisions(n) = (sign(real(y_n)) + 1j*sign(imag(y_n))) / sqrt(2); else
decisions(n) = d; end
end

Performance and plots figure;
subplot(2,2,1);
plot(10*log10(movmean(mse,50) + eps),'LineWidth',1.2); grid on; xlabel('Symbol
index'); ylabel('MSE (dB)');
title('Learning curve (moving average MSE, 50 samples)');

% Constellation before equalizer (received), pick a middle segment after channel delay
startPlot = trainLen + 200;
nPlot = 500;
idx_plot = startPlot:(startPlot+nPlot-1);
% sample received signal aligned to symbol timing: choose r samples starting at symbol
index
% received corresponding to symbol n is r(n + channel_delay) approx; for plotting use
r(pad+idx_plot) rx_sampled = r(1 + startPlot - 1 : startPlot + nPlot - 1); %
approximate sampling
subplot(2,2,2);
plot(real(rx_sampled), imag(rx_sampled), '.'); axis equal; grid on; xlabel('I');
ylabel('Q'); title('Received constellation (no equalization)');

% Equalized constellation subplot(2,2,3);
plot(real(y(idx_plot)), imag(y(idx_plot)), '.'); axis equal; grid on; xlabel('I');
ylabel('Q'); title('Equalized constellation (filter output)');

% Decisions (after equalization) subplot(2,2,4);

```

```

plot(real(decisions(idx_plot)), imag(decisions(idx_plot)), '.'); axis equal; grid on;
xlabel('I'); ylabel('Q'); title('Decisions (hard)');

% Display final SER (symbol error rate) in decision-directed section
% Evaluate over last portion after training decided = decisions(trainLen+1:end);
actual = s(trainLen+1:length(decisions)); ser = mean(decided ~= actual);
fprintf('Training length = %d, Equalizer length = %d, Final SER (post-training) = %g\n', trainLen, eqLen, ser);
Training length = 1000, Equalizer length = 7, Final SER (post-training) = 1 Optional:
show final equalizer taps
disp('Final equalizer taps (w):'); disp(w);
Final equalizer taps (w):
1.0623 + 0.0321i
-0.5290 + 0.1963i
0.4200 - 0.0331i
-0.3207 + 0.1011i
0.1909 - 0.0263i
-0.1411 + 0.0618i
0.0620 - 0.0036i

Helper: function sign for zero-handling (ensures non-zero) function sgn = sign(x)
sgn = zeros(size(x)); sgn(real(x) > 0) = 1;
sgn(real(x) < 0) = -1;
% zeros treated as +1 by default

sgn(real(x) == 0) = 1;
sgn = sgn + 1j*(zeros(size(x)));
% Imag part
imagsgn = zeros(size(x)); imagsgn(imag(x) > 0) = 1;
imagsgn(imag(x) < 0) = -1;
imagsgn(imag(x) == 0) = 1;
sgn = (real(sgn) + 1j*imagsgn); end

```

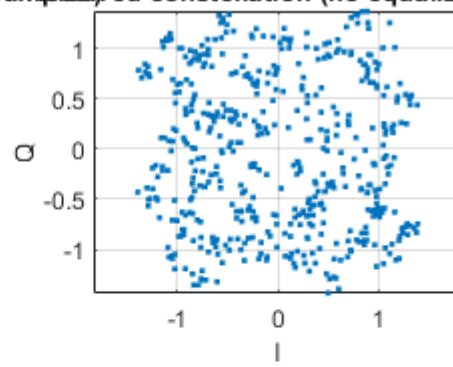
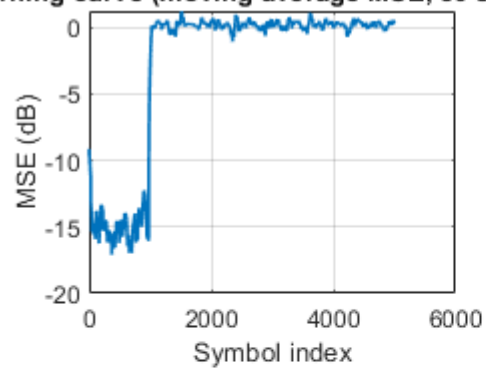
OUTPUT

The implementation of RLS adaptive equalizer is observed and analyzed successfully.

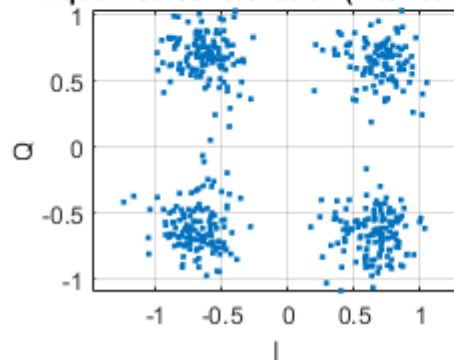
RESULT

Thus the experiment 'Channel Equalizer Design – RLS' was implemented and verified successfully using MATLAB.

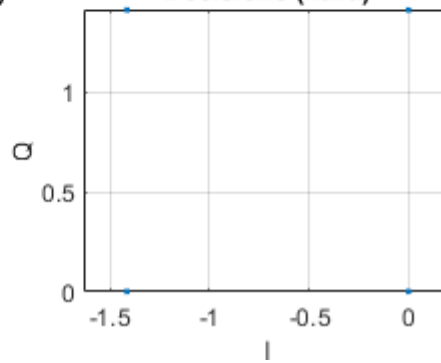
Learning curve (moving average MSE, 50 samples) Resolved constellation (no equalization)



Equalized constellation (filter output)



Decisions (hard)



EXPERIMENT NO: 7

OFDM TRANSCEIVER DESIGN

AIM

To design OFDM transceiver using MATLAB.

THEORY

An Orthogonal Frequency Division Multiplexing (OFDM) transceiver splits high-speed data streams across multiple closely-spaced, orthogonal subcarriers. This provides high spectral efficiency and robust immunity to multipath fading, making it the foundational technology for Wi-Fi, 4G LTE, and 5G networks.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
% Basic OFDM Transceiver Simulation (BPSK/16QAM option)

clc;
clear;
close all;

% OFDM PARAMETERS
N = 64;           % Number of subcarriers (IFFT size) CP = 16;           % Cyclic prefix
length
M = 4; % Modulation order (4 = QPSK, 2 = BPSK, 16 = 16QAM)

numSymbols = 1000; % Number of OFDM symbols

%TRANSMITTER
% Generate random bits
k = log2(M); % Bits per symbol bits = randi([0 1], numSymbols*N*k, 1);

% QAM Modulation
dataSymbols = qammod(bits2int(bits, k), M, 'UnitAveragePower', true);

% Reshape into OFDM symbols
dataMatrix = reshape(dataSymbols, N, numSymbols);

% IFFT
ifftData = ifft(dataMatrix, N);

% Add cyclic prefix
txData = [ifftData(end-CP+1:end,:); ifftData];

% Serialize for transmission txSignal = txData(:);
```

```

%CHANNEL
SNR_dB = 20;
rxSignal = awgn(txSignal, SNR_dB, 'measured');

%RECEIVER
% Reshape to matrix
rxMatrix = reshape(rxSignal, N+CP, numSymbols);

% Remove CP
rxNoCP = rxMatrix(CP+1:end, :);

% FFT
rxFFT = fft(rxNoCP, N);

% QAM Demodulation rxSymbols = rxFFT(:);
rxBits = int2bits(qamdemod(rxSymbols, M, 'UnitAveragePower', true), k);

% BER CALCULATION
[numErr, BER] = biterr(bits, rxBits);

fprintf('\n');
fprintf(' OFDM Simulation Results\n'); fprintf(' Modulation: %d-QAM\n', M); fprintf('
SNR: %d dB\n', SNR_dB); fprintf(' BER: %e\n', BER);
fprintf('\n');

% Helper Functions
function intVals = bits2int(bits, k)

bits = reshape(bits, k, []).'; intVals = bi2de(bits);
end

function bits = int2bits(intVals, k) bits = de2bi(intVals, k).';
bits = bits(:); end

```

OUTPUT

The design of OFDM transceiver using MATLAB is observed and analyzed successfully.

RESULT

Thus the experiment 'OFDM Transceiver Design' was implemented and verified successfully using MATLAB.

OUTPUT:

```

OFDM Simulation Results
Modulation: 4-QAM
SNR: 20 dB
BER: 0.000000e+00

```

EXPERIMENT NO: 8

DESIGN AND PERFORMANCE ANALYSIS OF ERROR CONTROL ENCODER AND DECODER (BLOCK AND CONVOLUTIONAL CODES)

AIM

To analyze error control coding techniques.

THEORY

Error control coding serves as the backbone of reliable digital communications, adding structured redundancy to original data sequences to isolate and correct transmission anomalies. The choice between Linear Block Codes (which partition data into distinct, memoryless chunks) and Convolutional Codes (which process continuous input streams via finite-state shift registers) depends entirely on the design constraints of the target system.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
clc;
clear;
close all;
%PARAMETERS
Nbits = 3e5; % Number of info bits EbN0dB = 0:2:12; % SNR range
msg = randi([0 1], Nbits, 1); % Random data bits
%UNCoded transmission (BPSK)
tx_unc = 1 - 2*msg; % BPSK mapping
%Hamming (7,4) BLOCK CODE G = [1 0 0 0 1 1 0;
0 1 0 0 1 0 1;
0 0 1 0 0 1 1;
0 0 0 1 1 1 1];

H = [1 1 0 1 1 0 0;
1 0 1 1 0 1 0;
0 1 1 1 0 0 1];

% trim message so it fits blocks of 4 L4 = 4 * floor(Nbits/4);
msg4 = msg(1:L4);

% group into 4-bit symbols
msg4_block = reshape(msg4, 4, []).';

% encode
code_ham = mod(msg4_block * G, 2); tx_ham = 1 - 2 * code_ham(:);
%Convolutional CODE (rate 1/2)

trellis = poly2trellis(7, [171 133]);
tbl = 5 * 7;

code_conv = convenc(msg, trellis); tx_conv = 1 - 2*code_conv;
```

```

%BER VECTORS
BER_unc = zeros(size(EbN0dB)); BER_ham = zeros(size(EbN0dB)); BER_conv =
zeros(size(EbN0dB));
%MAIN SIMULATION LOOP
for ii = 1:length(EbN0dB)
EbN0 = 10^(EbN0dB(ii)/10); N0 = 1/EbN0;
----- Uncoded -----
noise = sqrt(N0/2) * randn(size(tx_unc)); rx = tx_unc + noise;
dec = rx < 0;
BER_unc(ii) = sum(dec ~= msg) / Nbits;
----- Hamming (7,4) -----
noise = sqrt(N0/2) * randn(size(tx_ham)); rx = tx_ham + noise;
rx_bits = rx < 0;

rx_block = reshape(rx_bits, 7, []).'; syn = mod(rx_block * H.', 2);

err_pat = eye(7);

for j = 1:size(rx_block,1) s = syn(j,:);
if any(s)

[~, idx] = ismember(s, mod(err_pat*H.',2), 'rows'); if idx > 0
rx_block(j, idx) = ~rx_block(j, idx); end
end end

dec_ham = rx_block(:,1:4); dec_ham = dec_ham(:);
BER_ham(ii) = sum(dec_ham ~= msg4) / L4;
----- Convolutional (Viterbi) -----
noise = sqrt(N0/2) * randn(size(tx_conv)); rx = tx_conv + noise;
rx_hard = rx < 0;
dec_conv = vitdec(rx_hard, trellis, tbl, 'trunc', 'hard'); BER_conv(ii) =
sum(dec_conv ~= msg) / Nbits;
end

%PLOTTING BER CURVES
figure;
semilogy(EbN0dB, BER_unc, 'ko-', 'LineWidth', 1.5); hold on; semilogy(EbN0dB,
BER_ham, 'bs-', 'LineWidth', 1.5);
semilogy(EbN0dB, BER_conv, 'r^-', 'LineWidth', 1.5);

grid on; xlabel('Eb/N0 (dB)');
ylabel('Bit Error Rate');
title('BER Performance: Uncoded vs Hamming (7,4) vs Convolutional (1/2,K=7)');
legend('Uncoded','Hamming (7,4)','Convolutional (1/2)');

```

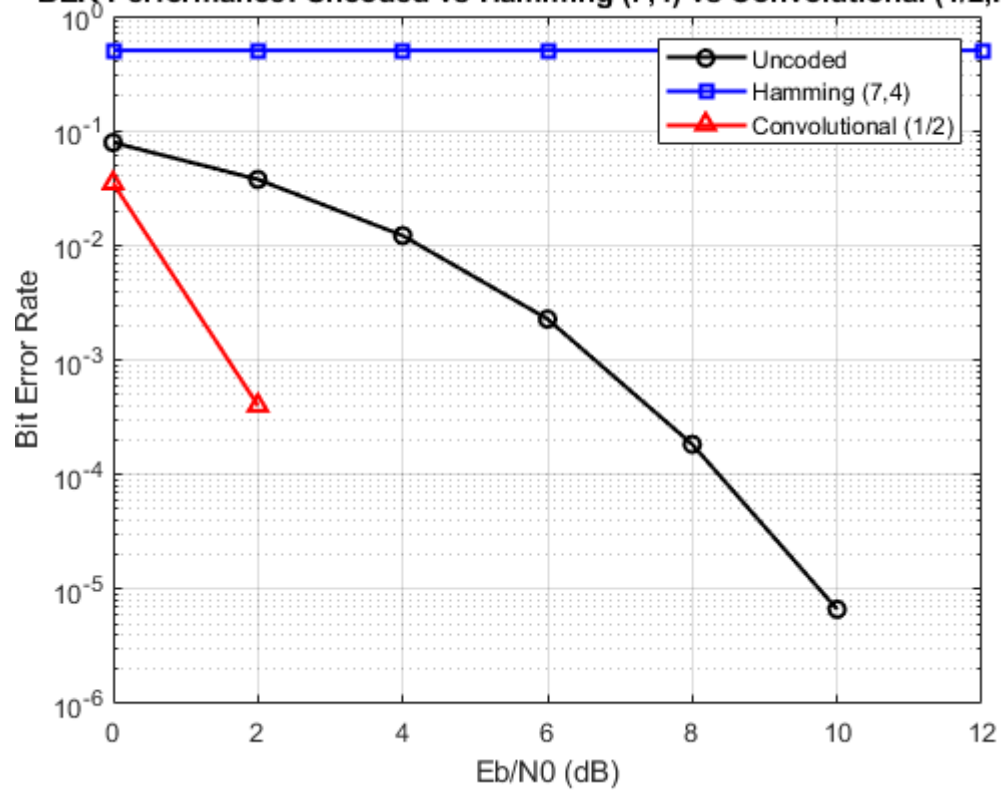
OUTPUT

The analyze error control coding techniques is observed and analyzed successfully.

RESULT

Thus the experiment 'Design and Performance Analysis of Error Control Encoder and Decoder (Block and Convolutional Codes)' was implemented and verified successfully using MATLAB.

BER Performance: Uncoded vs Hamming (7,4) vs Convolutional (1/2,K=7)



EXPERIMENT NO: 9

NOISE / ECHO CANCELLATION USING MATLAB

AIM

To implement adaptive noise and echo cancellation.

THEORY

Noise and echo cancellation in MATLAB rely on Digital Signal Processing (DSP) techniques. Common approaches include Wavelet Transform for static noise and Adaptive Filtering (LMS or RLS algorithms) for dynamic noise or acoustic echo. These methods are implemented using built-in functions, the Filter Design & Analysis Tool (fdatool), or Simulink models for real-time applications.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
clc;
%      NOISE + ECHO CANCELLATION USING LMS FILTER
clc;
clear;
close all;
%LOAD / GENERATE SPEECH SIGNAL
Fs = 8000;    % Sampling frequency
t = 0:1/Fs:3;% 3 sec duration
x = sin(2*pi*300*t)' + 0.5*sin(2*pi*600*t)'; % synthetic speech-like signal

% Add noise (for ANC experiment) noise = 0.4 * randn(length(x),1);
primary = x + noise;      % noisy speech
reference = noise;        % noise reference to ANC

%      NOISE CANCELLATION USING LMS ADAPTIVE FILTER

M = 32;          % Filter order
mu = 0.01;       % Step size

% Adaptive filter object
lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

% Apply adaptive filter
[~, e_ANC] = lms(reference, primary);

% ECHO CANCELLATION (AEC)
% Create echo path (FIR filter) echoPath = fir1(20, 0.3);

% Generate echo

echo = filter(echoPath, 1, x);
micSignal = x + echo;      % Microphone receives speech + echo
```

```

% Reference echo (loudspeaker signal) refEcho = x;

% LMS Filter for AEC
lmsAEC = dsp.LMSFilter('Length', 32, 'StepSize', 0.01); [~, e_AEC] = lmsAEC(refEcho,
micSignal);

%PLOT RESULTS

figure; subplot(3,1,1);
plot(x); title('Original Clean Signal');
subplot(3,1,2);
plot(primary); title('Noisy Signal (Speech + Noise)');

subplot(3,1,3); plot(e_ANC);
title('Noise Cancelled Output (ANC)');

figure; subplot(3,1,1);
plot(micSignal); title('Echoed Signal (Mic Input)');

subplot(3,1,2);
plot(echo); title('Echo Component');

subplot(3,1,3);

plot(e_AEC);
title('Echo Cancelled Output (AEC)');

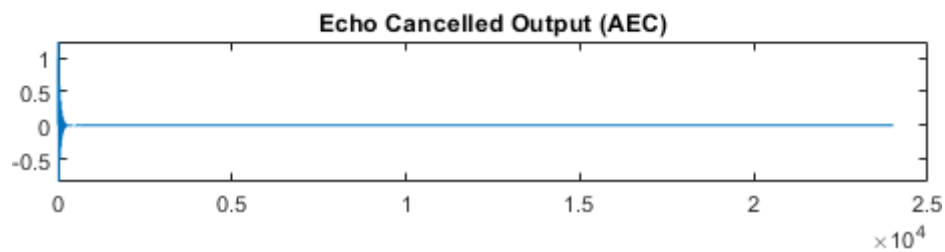
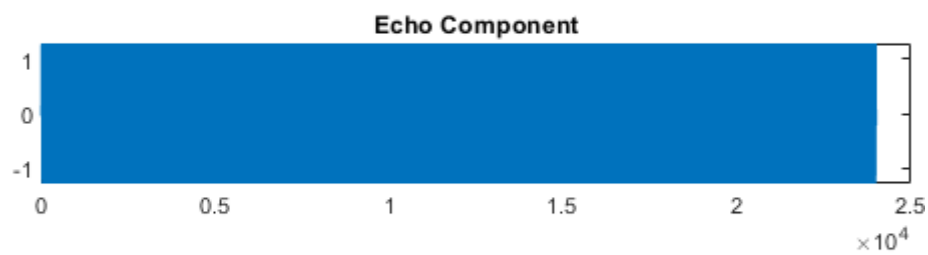
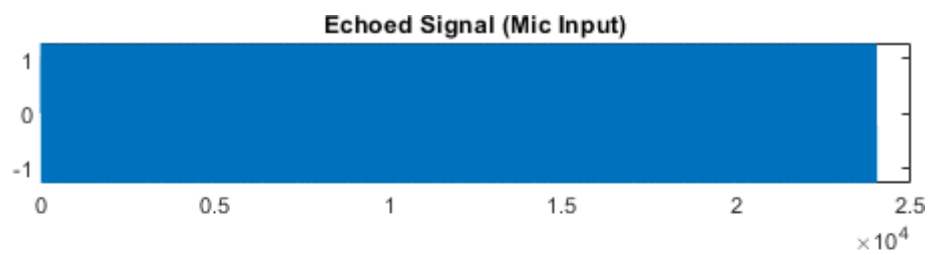
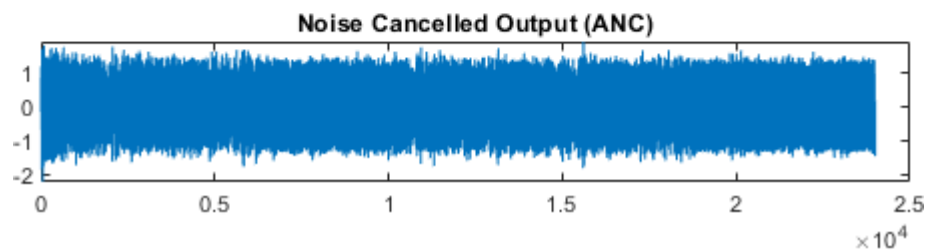
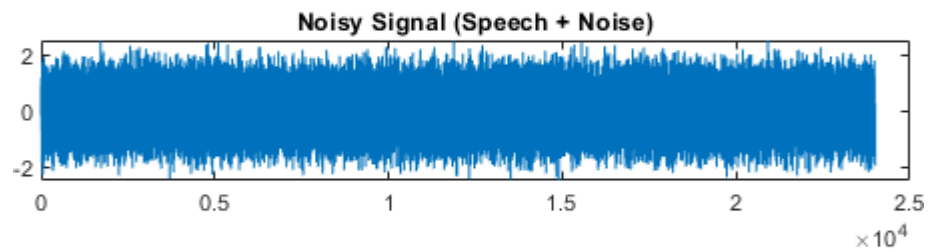
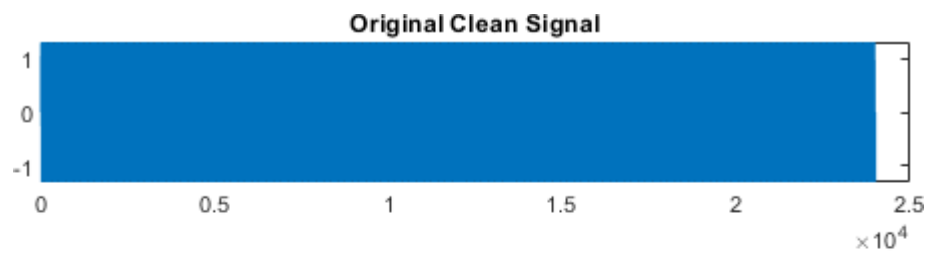
```

OUTPUT

The implementation of adaptive noise and echo cancellation is observed and analyzed successfully.

RESULT

Thus the experiment 'Noise / Echo Cancellation using MATLAB' was implemented and verified successfully using MATLAB.



EXPERIMENT NO: 10

WIRELESS CHANNEL CHARACTERIZATION

AIM

To study wireless channel characteristics.

THEORY

Wireless channel characterization in MATLAB involves simulating how environments distort transmitted signals. Using the Communications Toolbox, you can model multi-path fading, Doppler shift, and path loss to evaluate link performance.

ALGORITHM

1. Start the MATLAB environment.
2. Initialize system parameters.
3. Generate input signal/data.
4. Apply required processing/modulation.
5. Observe output characteristics.
6. Analyze the performance.
7. Stop the program.

MATLAB PROGRAM

```
Rayleigh, Rician, AWGN, PDP, Impulse Response, Frequency Resp. clc; clear; close all;
Simulation Parameters
Fs = 1e6;      % Sampling Frequency Ts = 1/Fs;
N = 5000;      % Number of samples
fc = 2.4e9;    % Carrier frequency (2.4 GHz)

% Generate random BPSK symbols data = randi([0 1], N, 1);
x = 2*data - 1; % BPSK mapping
AWGN Channel
SNR = 20;      % Signal-to-Noise Ratio y_awgn = awgn(x, SNR, 'measured');
Rayleigh Fading Channel
rayleighChan = comm.RayleighChannel( ... 'SampleRate', Fs, ...
'PathDelays', [0 1e-6 3e-6], ...
'AveragePathGains', [0 -3 -6]); % multipath profile y_rayleigh = rayleighChan(x);
Rician Fading Channel
ricianChan = comm.RicianChannel( ... 'SampleRate', Fs, ...
'KFactor', 5, ... % Line-of-sight component 'PathDelays', [0 2e-6], ...
'AveragePathGains', [0 -5]); y_rician = ricianChan(x);
%Power Delay Profile (PDP) pathDelays = rayleighChan.PathDelays;
pathGains = rayleighChan.AveragePathGains;

figure; stem(pathDelays*1e6, pathGains); xlabel('Delay (μs)'); ylabel('Average Gain (dB)'); title('Power Delay Profile (PDP)');

%Channel Impulse Response imp = [1; zeros(200,1)];
h_rayleigh = rayleighChan(imp);

figure; plot(abs(h_rayleigh)); xlabel('Samples'); ylabel('|h(t)|');
title('Rayleigh Channel Impulse Response');

%Frequency Response
```

```
H_f = fft(h_rayleigh, 1024); f_axis = linspace(0, Fs/2, 512);

figure; plot(f_axis/1e6, 20*log10(abs(H_f(1:512)))); xlabel('Frequency (MHz)');
ylabel('Magnitude (dB)'); title('Rayleigh Channel Frequency Response');

%Coherence Bandwidth Estimation R = abs(iff( abs(H_f).^2 ));
[~, idx] = find(R < max(R)/2, 1); % 50% coherence Bc = idx * (Fs/1024);

fprintf('\nEstimated Coherence Bandwidth: %.2f kHz\n', Bc/1e3); fprintf('Simulation
completed.\n');
Estimated Coherence Bandwidth: 0.98 kHz Simulation completed.v
```

OUTPUT

The study of wireless channel characteristics is observed and analyzed successfully.

RESULT

Thus the experiment 'Wireless Channel Characterization' was implemented and verified successfully using MATLAB.

