Πανεπιστήμιο Πατρών

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Ακαδημαϊκό Έτος 2024-2025

1η Εργαστηριακή Άσκηση

Table of Contents

Μέρος Α1: Κωδικοποίηση Πηγής με τη μέθοδο PCM

1) Υλοποίηση Σχήματος PCM

a) Ομοιόμορφος Κβαντιστής

Ο ομοιόμορφος κβαντιστής περιορίζει το σήμα στο εύρος [ελάχιστη τιμή, μέγιστη τιμή], χωρίζει το επίπεδο σε 2^N περιοχές για N bits ανά δείγμα και θέτει ως τιμές κβάντισης τα κέντρα των περιοχών. Το κάθε δείγμα κβαντίζεται στο κέντρο της περιοχής που ανήκει.

Προτείνεται να τρέξετε τον κώδικα A1 του παραρτήματος, ολόκληρο, με breakpoints στις γραμμές 51, 77, 103 και 116.

Παρατίθεται υλοποίησή του.

```
% Uniform Quantizer
function [xq, centers] = my_quantizer(x, N, min_value, max_value)

    % Preprocess the signal to be within [min_value, max_value]
    x = max(min(x, max_value), min_value);

    % Calculate the number of quantization areas
    number_of_areas = 2^N;

    % Compute Delta (width of each area)
    Delta = (max_value - min_value) / number_of_areas;

    % Initialize arrays
    lower_bound = zeros(1, number_of_areas);
    upper_bound = zeros(1, number_of_areas);
    centers = zeros(1, number_of_areas);
    xq = zeros(1, length(x));

    % Calculate bounds and centers
    for i = 0:number_of_areas - 1
        lower_bound(number_of_areas - i) = min_value + i * Delta;
        upper_bound(number_of_areas - i) = min_value + (i + 1) * Delta;
        centers(number_of_areas - i) = min_value + (i + 0.5) * Delta;
    end

    % Quantize the signal
    for i = 1:length(x)
        for j = 1:number_of_areas
```

```
            if (x(i) >= lower_bound(j) && (x(i) < upper_bound(j) || j == 1))
                xq(i) = j;
                break;
            end
        end
    end
end
```

b) Μη Ομοιόμορφος Κβαντιστής με Χρήση Αλγορίθμου Lloyd-Max

Ο κβαντιστής αυτός αρχικοποιείται όπως και ο ομοιόμορφος, αλλά για να υπολογίσει τα κέντρα και τα όρια των περιοχών χρησιμοποιεί τον επαναληπτικό αλγόριθμο Lloyd-Max. Ξεκινάει με ομοιόμορφα κατανεμημένα κέντρα και έπειτα, επαναληπτικά, υπολογίζει τα όρια των περιοχών κβάντισης ως τα κέντρα των τιμών κβάντισης και τις τιμές κβάνισης ως τα "κέντρα μάζας" των σημείων ανά περιοχή κβάντισης. Τερματίζει, όταν η διαφορά της μέσης παραμόρφωσης της τρέχουσας επανάληψης από αυτήν της προηγούμενης πέσει κάτω από κάποια τιμή κατωφλίου.

Παρατίθεται υλοποίησή του.

```
% Non Uniform Quantizer Using Lloyd-Max Algorihtm
function [xq, centers, D, lower_bound, upper_bound] = Lloyd_Max(x, N, min_value, max_value)

    % Tolerance for convergence
    tol = 1e-36;

    % Preprocess the signal to be within [min_value, max_value]
    x = max(min(x, max_value), min_value);

    % Calculate number of quantization areas
    number_of_areas = 2^N;

    % Compute Delta (width of each area)
    Delta = (max_value - min_value) / number_of_areas;

    % Initialize centers
    centers = zeros(1, number_of_areas);
    for i = 0:number_of_areas - 1
        centers(number_of_areas - i) = min_value + (i + 0.5) * Delta;
    end

    % Include extreme values to the centers temporarily
    centers = [min_value, centers, max_value];
```

```
    % Initialize vectors
    D = [];
    xq = zeros(1, length(x));
    lower_bound = zeros(1, number_of_areas);
    upper_bound = zeros(1, number_of_areas);

    iteration = 1;

    while true
        % Step 1: Calculate quantization bounds
        lower_bound(1) = centers(1);
        upper_bound(1) = (centers(2) + centers(3)) / 2;
        for i = 2:number_of_areas - 1
            lower_bound(i) = (centers(i) + centers(i+1)) / 2;
            upper_bound(i) = (centers(i+1) + centers(i+2)) / 2;
        end
        lower_bound(end) = (centers(end - 2) + centers(end - 1)) / 2;
        upper_bound(end) = centers(end);

        % Step 2: Quantize the signal
        for i = 1:length(x)
            [~, xq(i)] = min(abs(x(i) - centers(2:end - 1)));
        end

        % Step 3: Compute mean distortion
        quantized_values = centers(xq + 1);
        distortion = mean((x - quantized_values).^2);
        D = [D, distortion];

        % Step 4: Update centers
        for k = 1:number_of_areas
            points_in_area = x(x >= lower_bound(k) & x < upper_bound(k));
            if ~isempty(points_in_area)
                centers(k + 1) = mean(points_in_area);
            end
        end

        % Stop if the change in distortion is less than the tolerance
        if (iteration > 1 && abs(D(iteration) - D(iteration-1)) < tol)
            break;
        end

        % Increment iteration
        iteration = iteration + 1;
    end

    % Remove bounds from centers for final output
    centers = centers(2:end-1);

end
```

2) Αξιολόγηση Σχήματος PCM

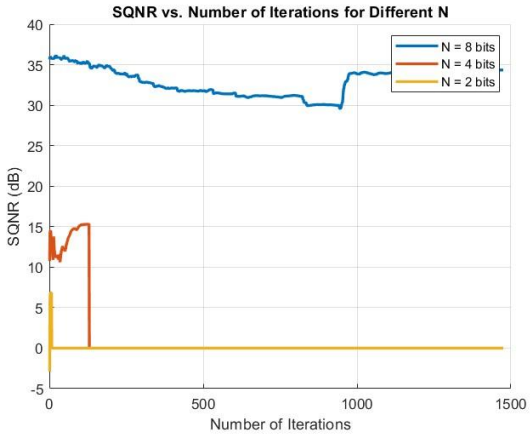
i) SQNR Ανά Επανάληψη Lloyd-Max

Ο κώδικας του ερωτήματος είναι ο παρακάτω.

```
% Question i)

% Load the audio signal
[y, fs] = audioread('speech.wav');

% Transpose for correct dimensions
y = y';

% Initializing parameters
min_value = -1;
max_value = 1;
N = [2, 4, 8];

% Compute signal power
signal_power = mean(y.^2);

% Initialize MSE matrix
MSE = [];

figure
hold on

% Loop over each value of N
for i = 3:-1:1

    % Non Uniform Quantizer
    [xq, centers, D, lower_bound, upper_bound] = Lloyd_Max(y, N(i), min_value, max_value);
    quantized(i, :) = centers(xq);
    MSE(i, 1:length(D)) = D;
    for j = 1:length(D)
        % Compute the SQNR
        noise_power = D(j);
        SQNR(i, j) = 10 * log10(signal_power / noise_power);
    end

    % Plot the SQNR for the current N
    plot(SQNR(i, :), 'DisplayName', sprintf('N = %d bits', N(i)), 'LineWidth', 2);
end
```

```
% Customize the plot
xlabel('Number of Iterations');
ylabel('SQNR (dB)');
title('SQNR vs. Number of Iterations for Different N');
legend show;
grid on;
hold off;
```

Το γράφημα που προκύπτει είναι το παρακάτω.



Η σταθερά που ελέγχει την έξοδο από το βασικό βρόχο του αλγόριθμου είναι $\varepsilon = 10^{-36}$. Ο αλγόριθμος τερματίζει σε 1475 επαναλήψεις για κωδικοποίηση με 8 bits, σε 129 για 4 bits και σε 7 για 2 bits. Επειδή χρησιμοποιούνται αναδρομικά οι περιοχές και οι τιμές κβάντισης για τον υπολογισμό των ίδιων, σε κάθε επανάληψη η συσσώρευση σφαλμάτων αριθμητικής κινητής υποδιαστολής οδηγεί στην έντονη διακύμανση του SQNR στην περίπτωση των 8 bits, δηλαδή των πολλών επαναλήψεων. Στα 4 και στα 2 bits το SQNR κορυφώνεται στις 129 και 7 επαναλήψεις αντίστοιχα και έπειτα το διάνυσμα του SQNR γεμίζει τις κενές θέσεις με μηδενικά. Εξού και η πτώση στο μηδέν μετά από τις επαναλήψεις αυτές.

ii) Σύγκριση SQNR Ομοιόμορφου και Μη Ομοιόμορφου Κβαντιστή

Δεδομένου πως έχει τρέξει ο κώδικας του προηγούμενου ερωτήματος, ο κώδικας του τρέχοντος ερωτήματος είναι ο παρακάτω.

```
% Question ii)

% Loop over each value of N
for i = 1:3
```

```
    % Uniform Quantizer
    [xq_u, centers_u] = my_quantizer(y, N(i), min_value, max_value);
    quantized_u(i, :) = centers_u(xq_u);

    % SQNR calculation
    noise_power_u = mean((y - quantized_u(i, :)).^2);
    signal_power_u = mean(y.^2);
    SQNR_u = 10 * log10(signal_power_u / noise_power_u);

    % MSE for later question
    MSE_u(i) = noise_power_u;

    row_sqnr = SQNR(i, :);
    non_zero_indices = find(row_sqnr ~= 0);
    % Get the last non-zero index
    last_non_zero_sqnr = row_sqnr(non_zero_indices(end));

    sprintf("N = %d bits. \n Uniform Quantizer: SQNR(dB) = %d \n" + ...
        " Non Uniform Quantizer SQNR(dB) = %d", N(i), SQNR_u, last_non_zero_sqnr)
end
```

Τα αποτελέσματα είναι τα εξής:

```
ans =

    "N = 2 bits.
      Uniform Quantizer: SQNR(dB) = -2.900066e+00
      Non Uniform Quantizer SQNR(dB) = 6.823450e+00"


ans =

    "N = 4 bits.
      Uniform Quantizer: SQNR(dB) = 1.075554e+01
      Non Uniform Quantizer SQNR(dB) = 1.529380e+01"


ans =

    "N = 8 bits.
      Uniform Quantizer: SQNR(dB) = 3.565353e+01
      Non Uniform Quantizer SQNR(dB) = 3.434674e+01"
```

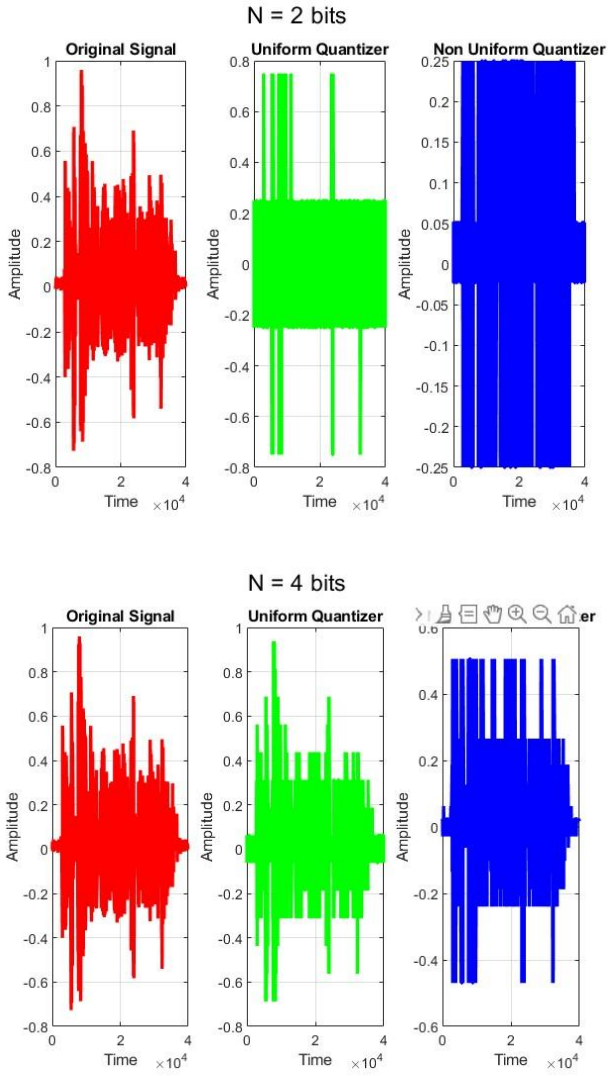
Η σύγκριση γίνεται μεταξύ ομοιόμορφου και μη ομοιόμορφου κβαντιστή στην τελευταία επανάληψη. Όπως φαίνεται παραπάνω, ο μη ομοιόμορφος και ο ομοιόμορφος κβαντιστής στα 4 και τα 8 bits αποδίδουν εξίσου καλά, το οποίο οφείλεται στην κατανομή των δεδομένων εισόδου. Στα 4 bits, όπως αναμένεται, ο μη ομοιόμορφος αποδίδει λίγο καλύτερα, ενώ στα 8 bits λόγω του πλήθους των επαναλήψεων και των σφαλμάτων αριθμητικής κινητής υποδιαστολής αποδίδει λίγο χειρότερα. Στα 2 bits υπερισχύει ο μη ομοιόμορφος κατά πολύ.

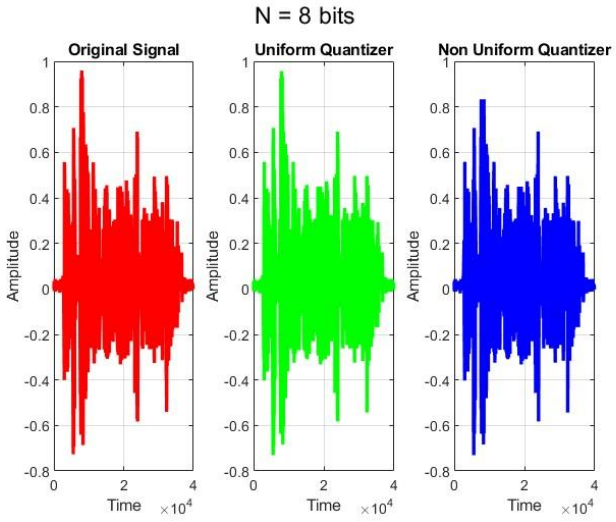
iii) Σύγκριση Κυμματομορφών και Ακουστικό Αποτέλεσμα

Δεδομένου πως έχει τρέξει ο κώδικας των προηγούμενων ερωτημάτων, ο κώδικας του τρέχοντος ερωτήματος είναι ο παρακάτω.

```
% Question iii)

% For all values of N
for i = 1:3
    % Plot the signals side by side
    subplot(1, 3, 1);
    plot(y, 'r', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Original Signal');
    grid on;

    subplot(1, 3, 2);
    plot(quantized_u(i, :), 'g', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Uniform Quantizer');
    grid on;

    subplot(1, 3, 3);
    plot(quantized(i, :), 'b', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Non Uniform Quantizer');
    grid on;

    sgtitle(sprintf('N = %d bits', N(i)));

    % Listen to the signals
    disp("Original Signal")
    sound(y, fs)
    pause(length(y) / fs)
    disp("Uniform Quantizer")
    sound(quantized_u(i, :))
    pause(length(quantized_u(i, :)) / fs)
    disp("Non Uniform Quantizer")
    sound(quantized(i, :))
    pause(length(quantized(i, :)) / fs)
end
```

Τα αποτελέσματα φαίνονται στα παρακάτω γραφήματα.





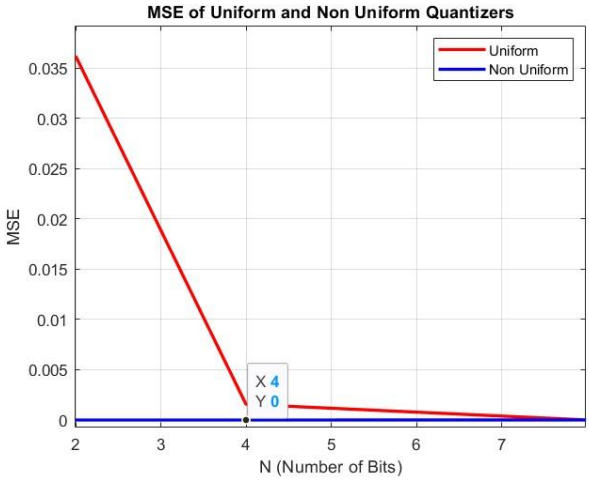
Όπως φαίνεται στα σχήματα, επιβεβαιώνονται τα συμπεράσματα του προηγούμενου ερωτήματος. Επίσης περισσότερα bits αντιστοιχούν σε πιο πιστή αναπαράσταση του σήματος. Αυτό επιβεβαιώνεται και ακουστικά καθώς ο ήχος των κβαντισμένων σημάτων για περισσότερα bits είναι πιο κοντά σε αυτόν του αρχικού.

iv) MSE Ανά Κβαντιστή και Πλήθος Bits

Δεδομένου πως έχει τρέξει ο κώδικας των προηγούμενων ερωτημάτων, ο κώδικας του τρέχοντος ερωτήματος είναι ο παρακάτω.

```
% Question iv)

plot(N, MSE_u, 'r', 'LineWidth', 2);
hold on;
plot(N, MSE(:, end), 'b', 'LineWidth', 2);
hold off;
title('MSE of Uniform and Non Uniform Quantizers');
xlabel('N (Number of Bits)');
ylabel('MSE');
legend('Uniform', 'Non Uniform');
grid on;
```

Το αποτέλεσμα φαίνεται στο παρακάτω γράφημα.



Το Μέσο Τετραγωνικό Σφάλμα αναμένεται να πέφτει όσο αυξάνεται το N και να είναι μικρότερο στον μη ομοιόμορφο κβαντιστή. Αυτό επιβεβαιώνεται από το σχήμα. Στον ομοιόμορφο κβαντιστή μειώνεται όσο το N αυξάνεται και ο μη ομοιόμορφος λόγω μικρού σφάλματος της τάξης του 10^{-6} το απεικονίζει συνεχώς στο μηδέν.

Μέρος Α2: Κωδικοποίηση Πηγής με τη μέθοδο DPCM

1) Υλοποίηση Συστήματος DPCM

Το σύστημα DPCM χωρίστηκε σε δύο συναρτήσεις, με την πρώτη να λειτουργεί ως πομπός και τη δεύτερη ως δέκτης. Η συνάρτηση πομπός δέχεται σαν ορίσματα το σήμα, την τάξη πρόβλεψης και τις παραμέτρους του ομοιόμορφου κβαντιστή και επιστρέφει το σφάλμα πρόβλεψης, την κβαντισμένη εκδοχή του, τους συντελεστές του προβλέπτη και την κβαντισμένη εκδοχή τους. Αρχικά χρησιμοποιεί τη συνάρτηση predictor_coefficients που παρατίθεται παρακάτω, για να υπολογίσει τους συντελεστές του προβλέπτη και στη συνέχεια τους κβαντίζει για να τους στείλει στο δέκτη. Στη συνέχεια υπολογίζει για τα πρώτα p δείγματα του σήματος το σφάλμα πρόβλεψης και την κβαντισμένη εκδοχή του για πρόβλεψη ίδια με το σήμα. Αυτό συμβαίνει γιατί ο προβλέπτης υπό κανονικές συνθήκες χρησιμοποιεί τις p προηγούμενες τιμές του σήματος για να προβλέψει την τρέχουσα, αλλά αυτή η δυνατότητα για τις πρώτες p τιμές του σήματος δεν υπάρχει γιατί οι p προηγούμενες τιμές αναφέρονται σε χρονικές στιγμές από το μηδέν και πίσω. Τέλος, για τις υπόλοιπες τιμές του σήματος υπολογίζονται τα παραπάνω με τους συμβατικούς τύπους. Η συνάρτηση δέκτης δέχεται σαν ορίσματα το κβαντισμένο σφάλμα πρόβλεψης, τους κβαντισμένους συντελεστές και την τάξη της πρόβλεψης και επιστρέφει το ανακατασκευασμένο σήμα. Όπως ο πομπός, έτσι και ο δέκτης για τα πρώτα p δείγματα του σήματος θέτει την πρόβλεψη ίση με το ίδιο το σήμα και για τα υπόλοιπα δείγματα χρησιμοποιεί τους συμβατικούς τύπους.

Παρατίθεται ο κώδικας της συνάρτησης predictor_coefficients.

```
% Computes the predictor coefficients by solving the Yule-Walker equations
```

```
function a = predictor_coefficients(x, p)

    % Use the built-in aryule function to solve Yule-Walker equations
    % Requires Singal Proccessing Toolbox
    [a_full, ~] = aryule(x, p);

    % Remove the leading 1 to get only the AR coefficients
    a = a_full(2:end);

    % Reverse the sign as aryule returns -a
    a = -a;

end
```

Χρησιμοποιεί τη συνάρτηση aryule η οποία υπολογίζει το μητρώο και το διάνυσμα αυτοσυσχέτισης και στη συνέχεια χρησιμοποιεί τον αλγόριθμο του Levinson για να λύσει τις εξισώσεις Yule-Walker που προκύπτουν. Απαιτεί την εγκατάσταση του Signal Processing Toolbox του Matlab.

Παρατίθεται ο κώδικας του πομπού.

```
% DPCM Sender function
function [y, y_hat, a_hat, a] = dpcm_sender(x, p, N, min_value, max_value)

    % Compute the length of the input
    l = length(x);

    % Error handling
    if p > l
        error('The predictor order cannot be greater than the signal length.');
    end

    % Initialize signal vectors
    y = zeros(1, l);
    y_hat = zeros(1, l);
    y_prime = zeros(1, l);
    y_hat_prime = zeros(1, l);

    % Compute the coefficients
    a = predictor_coefficients(x, p);

    % Quantize the coefficients to send them to the receiver
    [aq, centers] = my_quantizer(a, 8, -2, 2);
    a_hat = centers(aq);

    % Perform the first p iterations for hard-coded y_prime
```

```
    for n = 1:p
        y_prime(n) = x(n);
        y(n) = x(n) - y_prime(n);
        [yq, centers] = my_quantizer(y(n), N, min_value, max_value);
        y_hat(n) = centers(yq);
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

    % Perform the rest of the iterations
    for n = p + 1:l
        for i = 1:p
            y_prime(n) = y_prime(n) + a(i) * y_hat_prime(n - i);
        end
        y(n) = x(n) - y_prime(n);
        [yq, centers] = my_quantizer(y(n), N, min_value, max_value);
        y_hat(n) = centers(yq);
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

end
```

Παρατίθεται ο κώδικας του δέκτη.

```
% DPCM Receiver function
function y_hat_prime = dpcm_receiver(y_hat, a_hat, p)

    % Compute the length of the input
    l = length(y_hat);

    % Error handling
    if p > l
        error('The predictor order cannot be greater than the signal length.');
    end

     % Initialize signal vectors
    y_prime = zeros(1, l);
    y_hat_prime = zeros(1, l);

    % Perform the first p iterations for hard-coded y_prime
    for n = 1:p
        y_prime(n) = y_hat(n);
        y_hat_prime(n) = y_prime(n);
    end

    % Perform the rest of the iterations
    for n = p + 1:l
        for i = 1:p
```

```
            y_prime(n) = y_prime(n) + a_hat(i) * y_hat_prime(n - i);
        end
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

end
```

Ο κώδικας των ερωτημάτων 2-4 προτείνεται να τρέξει ολόκληρος όπως είναι στο παράρτημα με breakpoints στις γραμμές 38 και 59.

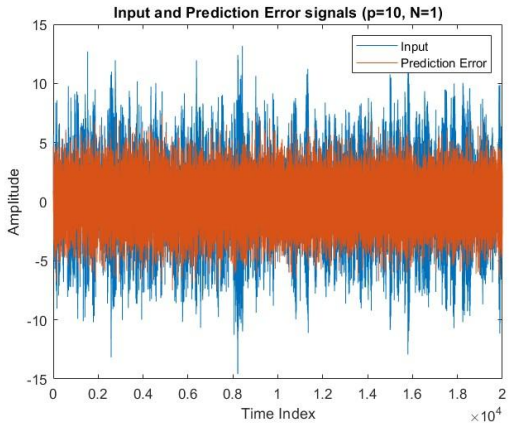
2) Σύγκριση Σήματος Εισόδου με Σφάλμα Πρόβλεψης

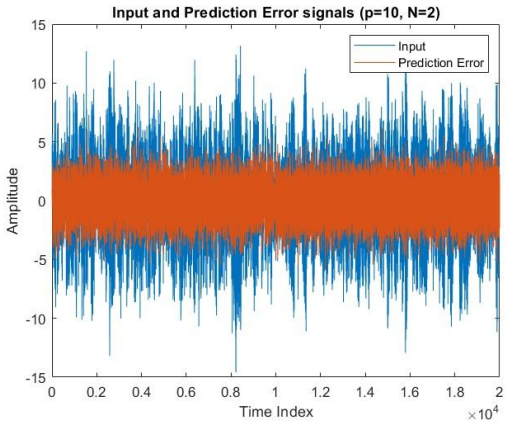
Οι τάξεις πρόβλεψης που επιλέχθηκαν ήταν 10 και 1000. Ο κώδικας του ερωτήματος είναι ο παρακάτω.

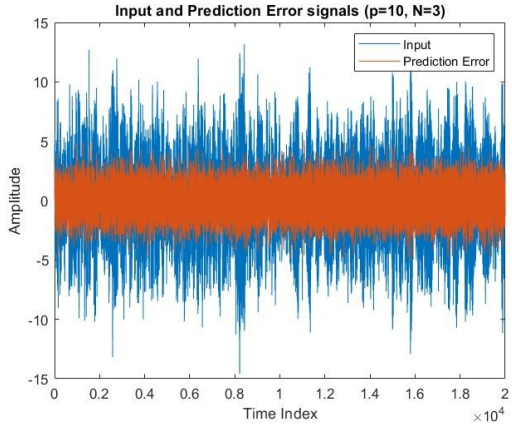
```
% Question 2)

% Load the input signal (t)
load source.mat
t = t';

% length of the input
l = length(t);

% Initialize variables
min_value = -3.5;
max_value = 3.5;
p = [10, 1000];
N = [1, 2, 3];

% Main loop to plot the signal next to the prediction error
for i = 1:length(N)
    for j = 1:length(p)
        % Call the DPCM sender function
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);

        % Plot the input signal next to the prediction error
        figure
        plot(t)
        hold on
        plot(y)
        hold off
        title(['Input and Prediction Error signals (p=' num2str(p(j)) ', N=' num2str(N(i)) ')'
        xlabel('Time Index')
        ylabel('Amplitude')
```

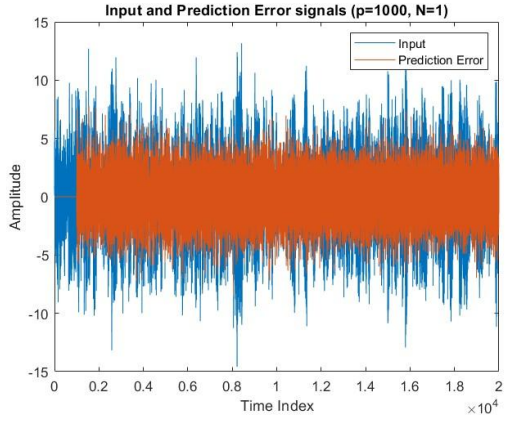
```
        legend('Input', 'Prediction Error')
    end
end
```

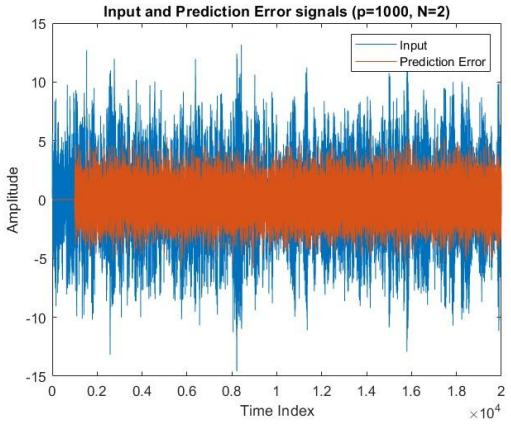
Τα γραφήματα που προκύπτουν είναι τα παρακάτω.

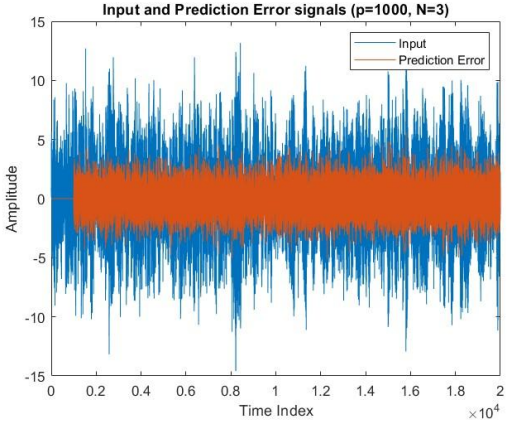






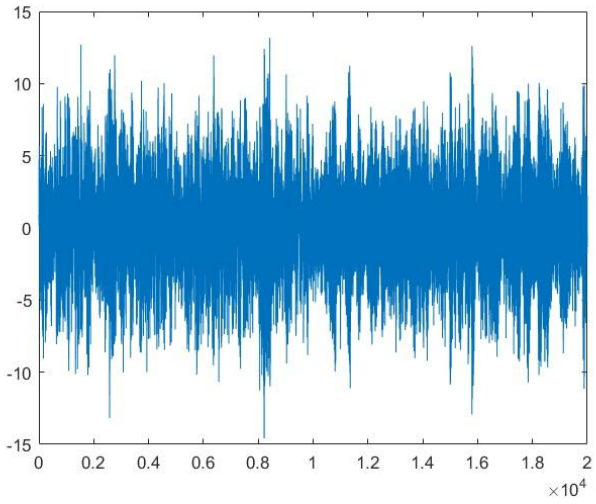






Όπως φαίνεται παραπάνω, με την αύξηση του αριθμού των bits μειώνεται και το σφάλμα πρόβλεψης γιατί ο κβαντιστής λειτουργεί αποδοτικότερα και κβαντίζει με μικρότερο σφάλμα κβάντισης τις τιμές που χρησιμοποιεί ο προβλέπτης για την πρόβλεψη. Η αύξηση της τάξης του προβλέπτη από 10 σε 1000 δεν εμφανίζεται να έχει

σημαντική επίδραση στο σφάλμα πρόβλεψης. Αυτό ισχύει γιατί το σήμα εισόδου δεν έχει μεγάλη αυτοσυσχέτιση και κατά συνέπεια οι παρελθοντικές τιμές του σήματος δε βοηθούν ιδιαίτερα στην πρόβλεψη της τρέχουσας. Η μεγάλη διακύμανση του σήματος εισόδου φαίνεται και στο παρακάτω γράφημα.

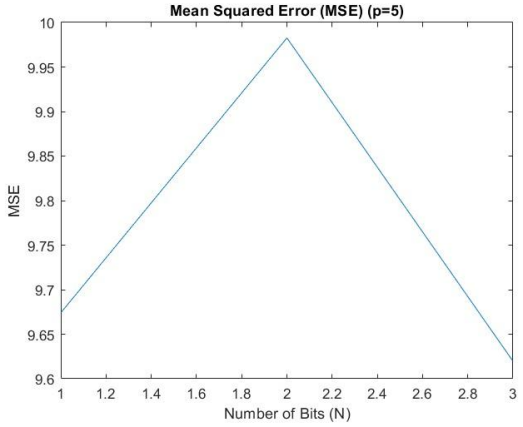


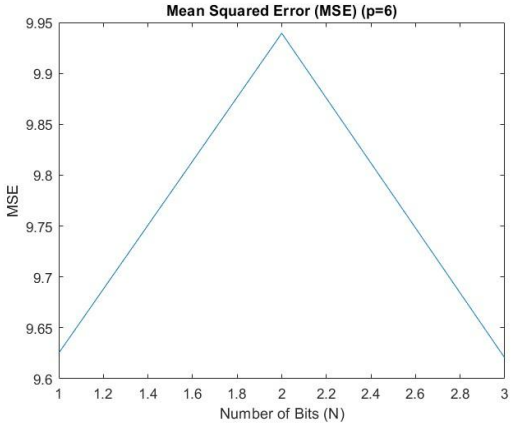
3) Αξιολόγηση Απόδοσης μέσω MSE

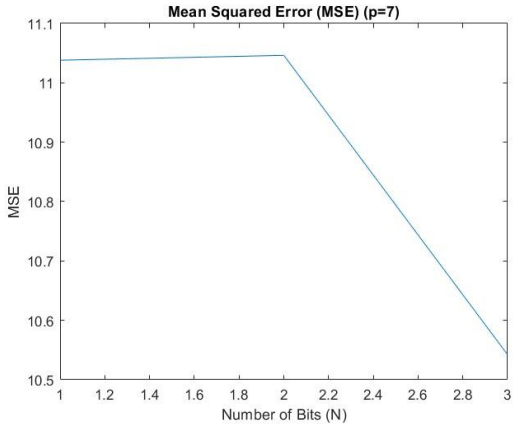
Ο κώδικας του ερωτήματος είναι ο παρακάτω.

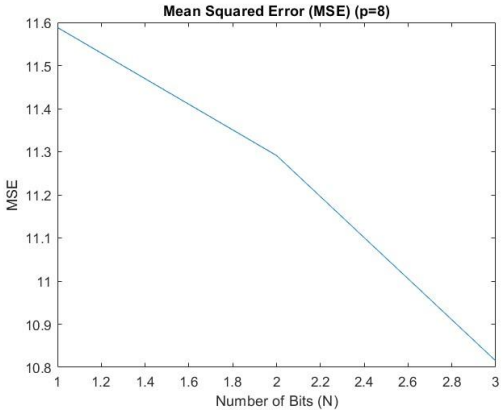
```
% Question 3)

% Initialize vectors
p = 5:10;
MSE = zeros(1, length(N));

% Main loop to plot the signal next to the prediction error
for j = 1:length(p)
    for i = 1:length(N)
        % Call the DPCM sender function
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);
        MSE(i) = mean((t - y).^2);
    end
    figure
    plot(MSE)
    title(['Mean Squared Error (MSE) (p=' num2str(p(j)) ')']);
    xlabel('Number of Bits (N)')
    ylabel('MSE')
    disp(a);
end
```

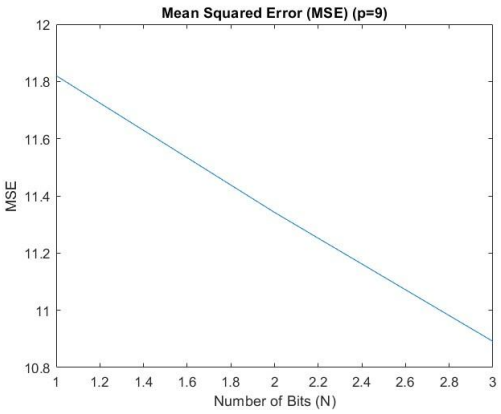
Τα γραφήματα που προκύπτουν είναι τα παρακάτω.

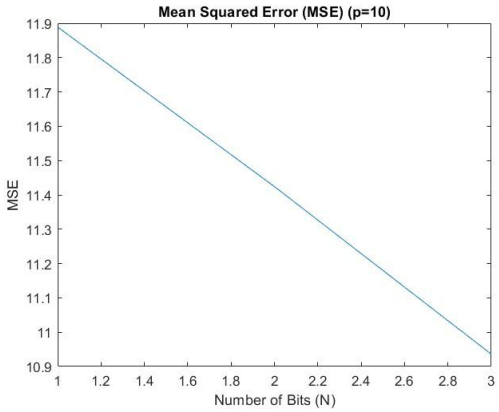












Στην πρόβλεψη μεγαλύτερης τάξης το μέσο τετραγωνικό σφάλμα παρουσιάζει μονότονη συμπεριφορά, κάτι που δεν ισχύει στις μικρότερες τάξεις, αλλα το μέγεθος του δε διαφέρει σημαντικά. Επιπρόσθετα, όσο αυξάνεται ο αριθμός των bits που χρησιμοποιεί ο κβαντιστής, τόσο μειώνεται το μέσο τετραγωνικό σφάλμα, όπως είναι και το αναμενόμενο, καθώς έτσι πέφτει το σφάλμα κβάντισης.

Οι συντελεστές του προβλέπτη για τάξη πρόβλεψης από 5 έως 10 είναι αντίστοιχα οι:

```
1.2853   -1.5857    0.9903   -0.5425   -0.0287

1.2865   -1.5627    0.9482   -0.4750   -0.0833    0.0425

1.2650   -1.5205    1.1884   -0.9544    0.7068   -0.6080    0.5056

1.0879   -1.3075    0.9407   -0.6200    0.2904   -0.0752    0.0623    0.3504

1.1436   -1.2976    0.9287   -0.5738    0.1917    0.0746   -0.1458    0.5236   -0.1592

1.1068   -1.1763    0.8950   -0.5565    0.2361   -0.0583    0.0693    0.2230    0.1057   -0.2316
```

Οι αντίστοιχοι συντελεστές για μεγαλύτερη τάξη πρόβλεψης έχουν γειτονικές τιμές. Αναμενόμενο αποτέλεσμα, καθώς οι συντελεστές προκύπτουν από το σήμα εισόδου που είναι πάντα ίδιο και για τάξη πρόβλεψης κατά ένα μεγαλύτερη, δηλαδή χρήση ενός ακόμη δείγματος, η αυτοσυσχέτιση δεν αλλάζει σημαντικά. Κατά συνέπεια,

ο αλγόριθμος του Levinson που χρησιμοποιεί την αυτοσυσχέτιση για να υπολογίσει τους συντελεστές δεν έχει κατά πολύ διαφρετικά αποτελέσματα.

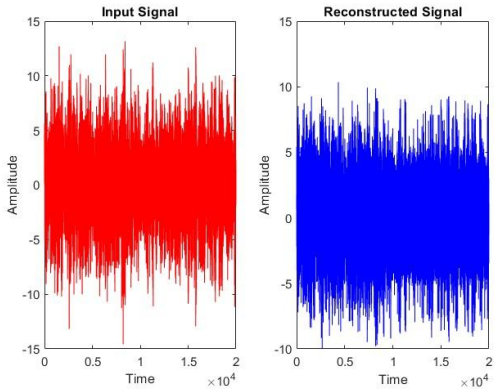
4) Ανακατασκευή

Ο κώδικας του ερωτήματος είναι ο παρακάτω.

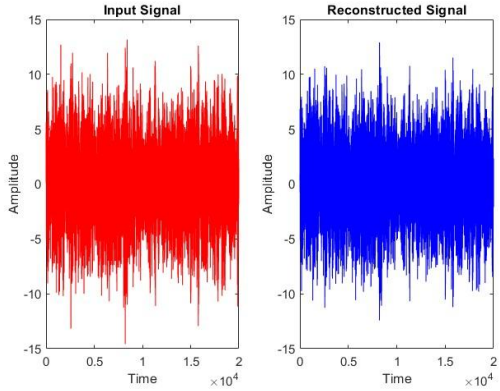
```
% Question 4)

% Initialize vectors
p = [5, 10];

for i = 1:length(N)
    for j = 1:length(p)
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);
        y_hat_prime = dpcm_receiver(y_hat, a_hat, p);
        figure
        subplot(1, 2, 1)
        plot(t, 'r')
        xlabel('Time')
        ylabel('Amplitude')
        title('Input Signal')
        subplot(1, 2, 2)
        plot(y_hat_prime, 'b')
        xlabel('Time')
        ylabel('Amplitude')
        title('Reconstructed Signal')
        sgtitle(['p=' num2str(p(j)) ', N=' num2str(N(i))]);
    end
end
```

Τα γραφήματα που προκύπτουν είναι τα παρακάτω.

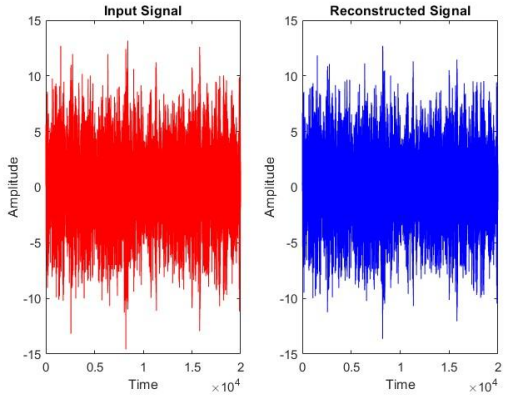
p=5, N=1



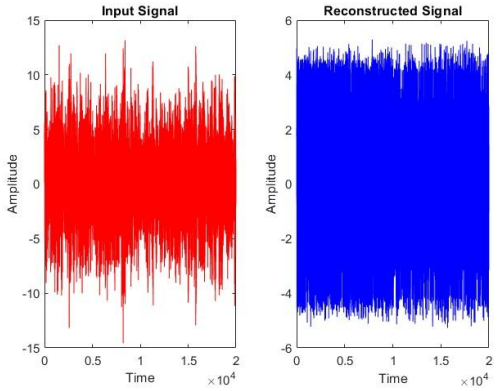
p=5, N=2



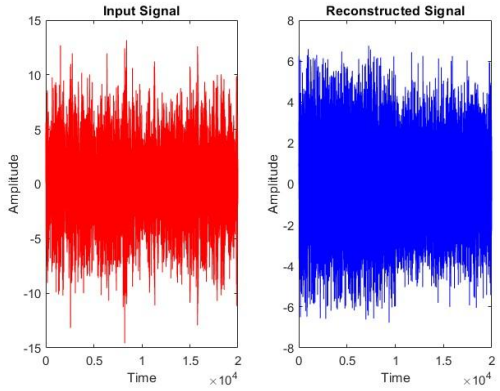
p=5, N=3



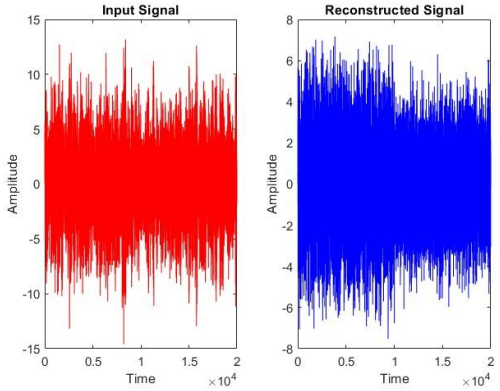
p=10, N=1



p=10, N=2



p=10, N=3



Όπως φαίνεται στα παραπάνω γραφήματα, περισσότερα bits για τον κβαντιστή μεταφράζονται σε πιο πιστή ανακατασκευή του σήματος εισόδου. Αυτό το αποτέλεσμα είναι και το αναμενόμενο καθώς περισσότερα bits συνεπάγονται μικρότερο σφάλμα κβάντισης. Όσον αφορά την τάξη πρόβλεψης, η μικρότερη αποδίδει καλύτερα λόγω της φύσης του σήματος εισόδου. Το σήμα δεν παρουσιάζει σημαντική συσχέτιστη σε μακρινές παρελθοντικές τιμές και έτσι αν ληφθούν υπόψην κατά την πρόβλεψη, η ανακατασκευή δε θα είναι καλή.

Μέρος Β: Μελέτη Απόδοσης Ομόδυνου Ζωνοπερατού Συστήματος M-PAM

1) Υλοποίηση Συστήματος M-PAM

Το σύστημα αποτελείται από πομπό, κανάλι και δέκτη, καθένα από τα οποία αποτελείται από περισσότερα υποσυστήματα. Κάθε υποσύστημα υλοποιείται ως ξεχωριστή συνάρτηση, της οποίας η υλοποίηση παρατίθεται παρακάτω.

Πρώτο σύστημα είναι ο πομπός. Ο κώδικας του είναι ο εξής.

```
% M-PAM transmitter system
function bandpass_signal = transmitter(binary_sequence, M, fc, Rs, encoding)

    % Compute basic parameters
    Tsymbol = 1 / Rs;
    fs = 4 * fc;
    Ts = 1 / fs;

    % Map the binary sequence to symbols using M-PAM mapper
    symbols = mapper(binary_sequence, M, encoding);

    % Shape the symbols using M-PAM pulse shaper
    baseband_signal = pulse_shaper(symbols, Tsymbol, Ts);

    % Take the signal to frequency fc using M-PAM modulator
    bandpass_signal = modulator(baseband_signal, fc, Ts);

end
```

Δέχεται σαν παράμετρο τη δυαδική ακολουθία, την τάξη διαμόρφωσης, τη συχνότητα διαμόρφωσης, το ρυθμό συμβόλων και την κωδικοποίηση και επιστρέφει το ζωνοπερατό σήμα. Αρχικά χρησιμοποιεί τον mapper και αντιστοιχίζει τη δυαδική ακολουθία σε σύμβολα. Στη συνέχεια χρησιμοποιεί τον pulse shaper για να

πολλαπλασιάσει τα σύμβολα με έναν παλμό ώστε να μπορέσουν να μετοδοθούν. Τέλος, χρησιμοποιεί τον modulator για να φέρει το σήμα στην επιθυμητή συχνότητα.

Από τα επιμέρους υποσυστήματα πρώτος είναι ο mapper. Ο κώδικας του είναι ο εξής.

```
% Mapper subsystem of the M-PAM transmitter
function symbols = mapper(binary_sequence, M, encoding)

    % Check if M is a power of 2
    if mod(M, 2) ~= 0 || log2(M) ~= floor(log2(M))
        error('M must be a power of 2.');
    end

    % Number of bits per symbol
    bits_per_symbol = log2(M);

    % Compute amplitude scaling factor A (Mean Energy should be 1)
    A = sqrt(3 / (M^2 - 1));

    % Define M-PAM amplitudes based on A computed above
    amplitudes = A * (-(M-1):2:(M-1));

    % Divide the binary sequence into groups of bits of size "bits_per_symbol"
    bit_groups = reshape(binary_sequence, bits_per_symbol, [])';

    % Choose encoding
    if strcmpi(encoding, "grey")

        % Convert bit groups to decimal numbers
        decimal_numbers = bin2dec(num2str(bit_groups));

        % Perform Gray encoding (grey = number XOR number>>)
        indices = bitxor(decimal_numbers, floor(decimal_numbers / 2));

    elseif strcmpi(encoding, "binary")

        % Convert bit groups to decimal numbers
        indices = bin2dec(num2str(bit_groups));

    else
        % Error, unknown encoding
        error("Choose between 'grey' and 'binary' encoding.");
    end

    % Map the indices to the amplitudes
    symbols = amplitudes(indices + 1);
```

```
end
```

Δέχεται σαν παράμετρο τη δυαδική ακολουθία, την τάξη διαμόρφωσης και την κωδικοποίηση και επιστρέφει μια ακολουθία συμβόλων. Αρχικά ελέγχει αν η τάξη διαμόρφωσης είναι δύναμη του 2. Στη συνέχεια, υπολογίζει τις πιθανές τιμές των συμβόλων που μπορούν να αντιστοιχιστούν. Έπειτα, ομαδοποιεί τα bits για να συνθέσει σύμβολα, και τέλος, ανάλογα με την κωδικοποίηση αντιστοιχίζει τις ομάδες bits σε σύμβολα δημιουργώντας την ακολουθία συμβόλων.

Το επόμενο υποσύστημα είναι ο pulse shaper. Ο κώδικα του είναι ο εξής.

```
% Pulse shaper subsystem of the M-PAM transmitter
function baseband_signal = pulse_shaper(symbols, Tsymbol, Ts)

    % Compute the rectangular pulse as round(Tsymbol / Ts) samples
    number_of_samples = round(Tsymbol / Ts);
    amplitude = sqrt(2 / Tsymbol);
    pulse = amplitude * ones(1, number_of_samples);

    % Make the symbols last as long as the pulse (zero padding in between samples)
    symbols_upsampled = upsample(symbols, length(pulse));

    % Convolve (multiply each symbol and then add) with the pulse
    baseband_signal = conv(symbols_upsampled, pulse, 'same');

end
```

Δέχεται ως όρισμα την ακολουθία συμβόλων, την περίοδο του ενός συμβόλου και την περίοδο δειγματοληψίας και επιστρέφει ένα σήμα βασικής ζώνης. Αρχικά κατασκευάζει τον ορθογώνιο παλμό. Στη συνέχεια, βάζει μηδενικά ανάμεσα στα σύμβολα της ακολουθίας συμβόλων ώστε να τους δώσει χρονική διάρκεια και να μπορέσουν να πολλαπλασιαστούν με τον παλμό. Τέλος, επειδή ο παλμός πρέπει να πολλαπλασιαστεί με κάθε ξεχωριστό σύμβολο, κατάλληλη πράξη είναι η συνέλιξη. Έτσι προκύπτει το σήμα βασικής ζώνης.

Τελευταίο υποσύτημα του δέκτη είναι ο modulator. Ο κώδικας του είναι ο εξής.

```
% Modulator subsystem of the M-PAM transmitter
function bandpass_signal = modulator(baseband_signal, fc, Ts)

    % Initialize time vector
    t = (0:length(baseband_signal)-1) * Ts;
```

```
    % Take the signal to the frequency fc
    bandpass_signal = baseband_signal .* cos(2 * pi * fc * t);

end
```

Δέχεται ως όρισμα το σήμα βασικής ζώνης, τη συχνότητα διαμόρφωσης και την περίοδο δειγματοληψίας και επιστρέφει το ζωνοπερατό σήμα. Η μόνη του λειτουργία είναι να πολλαπλασιάσει το σήμα με ένα συνημίτονο συχνότητας fc.

Αφού ο πομπός στείλει το ζωνοπερατό σήμα, για να φτάσει στο δέκτη, το σήμα πρέπει να περάσει μέσα από το κανάλι λευκού προσθετικού θορύβου. Ο κώδικας του είναι ο παρακάτω.

```
% AWGN Channel
function noisy_signal = channel(bandpass_signal, SNR, M)

    % Compute the noise variance σ^2
    variance = 2 / (log2(M) * 10^(SNR / 10));

    % Generate Gaussian noise with 0 mean and σ^2 variance
    noise = sqrt(variance) * randn(1, length(bandpass_signal));

    % Add the noise to the signal
    noisy_signal = bandpass_signal + noise;

end
```

Το κανάλι δέχεται σαν όρισμα το ζωνοπρατό σήμα, το SNR και την τάξη διαμόρφωσης και επιστρέφει το μολυσμένο από θόρυβο σήμα. Για αρχή υπολογίζει τη διασπορά του θορύβου βάσει του SNR και παράγει το θόρυβο, και στη συνέχεια μολύνει το σήμα με το θόρυβο δημιουργώντας ένα μολυσμένο σήμα.

Αφού παραχθεί το μολυσμένο σήμα, επόμενο βήμα είναι να το λάβει ο δέκτης. Ο κώδικας του είναι ο εξής.

```
% M-PAM receiver system
function receiver_sequence = receiver(noisy_signal, M, fc, Rs, encoding)

    % Compute basic parameters
    Tsymbol = 1 / Rs;
    fs = 4 * fc;
    Ts = 1 / fs;
```

```
    % Return the signal to the baseband using M-PAM demodulator
    baseband_signal = demodulator(noisy_signal, fc, Ts);

    % Perform matched filtering using M-PAM matched filter
    symbols = matched_filter(baseband_signal, Tsymbol, Ts);

    % Detect the received symbols using M-PAM detector
    indices = detector(symbols, M);

    % Map the symbols back to binary values using M-PAM demapper
    receiver_sequence = demapper(indices, M, encoding);

end
```

Δέχεται ως όρισμα το μολυσμένο σήμα, την τάξη διαμόρφωσης, τη συχνότητα διαμόρφωσης, το ρυθμό συμβόλων και την κωδικοποίηση και επιστρέφει μια δυαδική ακολουθία. Αρχικά περνάει το σήμα μέσα από τον demodulator και επιστρέφει το σήμα στη βασική ζώνη. Μετά το περνάει μέσα από το matched filter για να βγάλει τον παλμό και να μείνουν μόνο σύμβολα. Στη συνέχεια το περνάει μέσα από τον detector για να αποφασίσει ποιά σύμβολα στάλθηκαν, και τέλος, το περνάει μέσα από το demapper για να μετατρέψει την ακολουθία συμβόλων σε δυαδική ακολουθία.

Πρώτο από τα επιμέρους υποσυστήματα είναι ο demodulator. Ο κώδικας του είναι ο εξής.

```
% Mapper subsystem of the M-PAM receiver
function baseband_signal = demodulator(noisy_signal, fc, Ts)

    % Initialize time vector
    t = (0:length(noisy_signal)-1) * Ts;

    % Return the signal to the baseband
    baseband_signal = noisy_signal .* cos(2 * pi * fc * t);

end
```

Δέχεται ως όρισμα το μολυσμένο σήμα, τη συχνότητα διαμόρφωσης και την περίοδο δειγματοληψίας και επιστρέφει το σήμα βασικής ζώνης. Η μοναδική λειτουργία του είναι να πολλαπλασιάσει ξανά το σήμα με ένα συνημίτονο συχνότητας fc για να το επαναφέρει στη βασική ζώνη.

Το επόμενο υποσύστημα είναι το matched filter. Ο κώδικας του είναι ο εξής.

```
% Matched filter subsystem of the M-PAM receiver
function symbols = matched_filter(baseband_signal, Tsymbol, Ts)

    % Compute the rectangular pulse as round(Tsymbol / Ts) samples
    number_of_samples = round(Tsymbol / Ts);
    amplitude = sqrt(2 / Tsymbol);
    pulse = amplitude * ones(1, number_of_samples);

    % Perform matched filtering
    match_filtered_signal = conv(baseband_signal, pulse, 'same');

    % Recover the symbols (the corellation is higher at the middles of the intervals)
    symbol_indices = round(number_of_samples/2:number_of_samples:length(match_filtered_signal));
    symbols = match_filtered_signal(symbol_indices);

end
```

Δέχεται ως όρισμα το σήμα βασικής ζώνης, την περίοδο συμβόλου και την περίοδο δειγματοληψίας και επιστρέφει μια ακολουθία συμβόλων. Αρχικά υπολογίζει τον παλμό και συνελίσσει τον παλμό με το σήμα. Στην ουσία η συνέλιξη υπολογίζει την αυτοσυσχέτιση του παλμού με το σήμα και δίνει ένα φιλτραρισμένο σήμα (έχει αφαιρεθεί ο παλμός). Στα μέσα των περιοχών των συμβόλων η συσχέτιση είναι υψηλότερη, έτσι αν δειγματοληπτηθεί το φιλτραρισμένο σήμα στα σημεία αυτά, δημιουργείται μια ακολουθία από σύμβολα.

Επόμενο υποσύστημα είναι ο detector. Ο κώδικας του είναι ο εξής.

```
% Detector subsystem of the M-PAM receiver
function indices = detector(symbols, M)

    % Compute the amplitude scaling factor A
    A = sqrt(3 / (M^2 - 1));

    % Map the amplitudes to the nearest M-PAM levels
    amplitudes = A * (-(M-1):2:(M-1));

    % Initialize a vector to store the indices of the nearest amplitudes
    indices = zeros(size(symbols));

    % Loop through each symbol in symbols received
    for i = 1:length(symbols)

        % Calculate the absolute difference between the current symbol and all amplitudes
        differences = abs(symbols(i) - amplitudes);

        % Find the index of the amplitude with the smallest difference
        [~, indices(i)] = min(differences);
```

```
    end

    % Adjust the indices to start from 0
    indices = indices - 1;

end
```

Δέχεται ως όρισμα την ακολουθία συμβόλων και την τάξη διαμόρφωσης και επιστρέφει μια νέα ακολουθία συμβόλων. Αρχικά υπολογίζει τα πλάτη του M-PAM και στη συνέχεια αντιστοιχίζει τα σύμβολα της ακολουθίας συμβόλων στα κοντινότερα πλάτη. Αυτή την αντιστοίχιση επιστρέφει.

Τελευταίο υποσύστημα είναι ο demapper. Ο κώδικας του είναι ο εξής.

```
% Demapper subsystem of the M-PAM receiver
function receiver_sequence = demapper(indices, M, encoding)

    % Number of bits per symbol
    bits_per_symbol = log2(M);

    % Check the encoding
    if strcmpi(encoding, "grey")

        % Convert grey to binary
        binary_indices = zeros(size(indices));
        binary_indices(:, 1) = indices(:, 1);
        for bit = 2:log2(M)
            binary_indices(:, bit) = xor(binary_indices(:, bit-1), indices(:, bit));
        end
        decimal_numbers = binary_indices;

    elseif strcmpi(encoding, "binary")

        % Perform no conversion
        decimal_numbers = indices;

    else
        error("Choose between 'grey' and 'binary' encoding.");
    end

    % Convert decimal numbers to binary sequence
    receiver_sequence = zeros(1, length(decimal_numbers) * bits_per_symbol);
    for i = 1:length(decimal_numbers)
        value = decimal_numbers(i);
        for bit = bits_per_symbol:-1:1
            receiver_sequence((i-1) * bits_per_symbol + (bits_per_symbol - bit + 1)) = bitget(v
```

```
        end
    end
end
```

Δέχεται ως όρισμα την ακολουθία συμβόλων, την τάξη διαμόρφωσης και την κωδικοποίηση και επιστρέφει μια δυαδική ακολουθία. Ελέγει την κωδικοποίηση και αν δεν είναι ήδη δυαδική την μετατρέπει. Τέλος, μετατρέπει τα σύμβολα σε bits και επιστρέφει την τελική δυαδική ακολουθία.

2) Υπολογισμός BER ως προς SNR

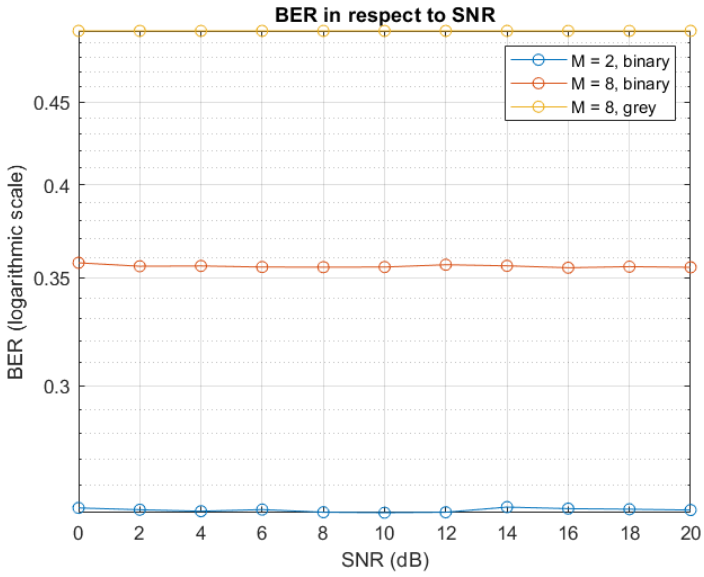
Ο κώδικας του ερωτήματος είναι ο παρακάτω.

```
% Question 2)

% Create a binary sequence
binary_sequence = randi([0, 1], 1, 100000);

% Initialize the sender parameters
fc = 25e5;
Rs = 25e4;
M = [2, 8];
SNR = 0:2:20;

% Initialize BER matrix
BER = zeros(length(M) + 1, length(SNR));

% Compute BER for each M value using binary encoding
for i = 1:length(M)

    % Number of bits per symbol for current M
    bits_per_symbol = log2(M(i));

    % Pad the binary sequence with zeros if necessary
    padding_length = mod(-length(binary_sequence), bits_per_symbol);
    if padding_length > 0
        binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
    else
        binary_sequence_padded = binary_sequence;
    end

    % Loop through SNR values
    for j = 1:length(SNR)

        % Pass the binary sequence through the sender system
```

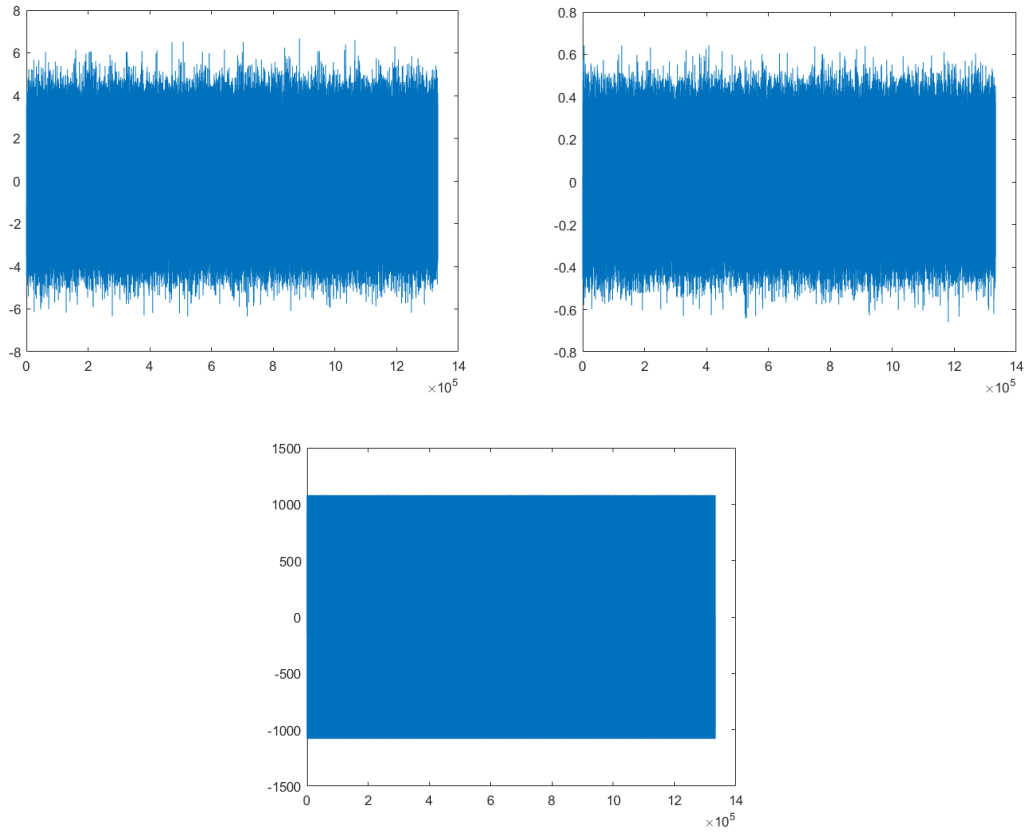
```
        bandpass_signal = transmitter(binary_sequence_padded, M(i), fc, Rs, 'binary');

        % Pass the bandpass signal through the channel
        noisy_signal = channel(bandpass_signal, SNR(j), M(i));

        % Pass the noisy signal through the receiver
        receiver_sequence = receiver(noisy_signal, M(i), fc, Rs, 'binary');

        % Compute the number of bit errors
        error_count = sum(binary_sequence_padded ~= receiver_sequence);

        % Compute the Bit Error Rate (BER)
        BER(i, j) = error_count / length(binary_sequence_padded);

    end
end

% Compute BER for M = 8 using grey encoding

% Number of bits per symbol for current M = 8
bits_per_symbol = log2(M(end));

% Pad the binary sequence with zeros if necessary
padding_length = mod(-length(binary_sequence), bits_per_symbol);
if padding_length > 0
    binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
else
    binary_sequence_padded = binary_sequence;
end

% Loop through SNR values
for j = 1:length(SNR)

    % Pass the binary sequence through the sender system
    bandpass_signal = transmitter(binary_sequence_padded, M(end), fc, Rs, 'grey');

    % Pass the bandpass signal through the channel
    noisy_signal = channel(bandpass_signal, SNR(j), M(end));

    % Pass the noisy signal through the receiver
    receiver_sequence = receiver(noisy_signal, M(end), fc, Rs, 'grey');

    % Compute the number of bit errors
    error_count = sum(binary_sequence_padded ~= receiver_sequence);

    % Compute the Bit Error Rate (BER)
    BER(end, j) = error_count / length(binary_sequence_padded);

end
```

```
% Plot the BER curves for all M values in logarithmic scale
figure;
for i = 1:length(M)
    semilogy(SNR, BER(i, :), '-o', 'DisplayName', sprintf('M = %d, binary', M(i)));
    hold on;
end
semilogy(SNR, BER(length(M) + 1, :), '-o', 'DisplayName', sprintf('M = %d, grey', M(end)));

% Finalize plot
xlabel('SNR (dB)');
ylabel('BER (logarithmic scale)');
title('BER in respect to SNR');
legend('show');
grid on;
```

Τρέξιμο του κώδικα εμφανίζει το εξής γράφημα:



Όπως φαίνεται τα 8-PAM έχουν περισσότερα σφάλματα γιατί τα σύμβολα πρέπει να αντιστοιχιστούν σε περισσότερες και στενότερες περιοχές, με αποτέλεσμα να είναι συχνότερο κάποιο σύμβολο λόγω θορύβου να σπρωχτεί σε γειτονική περιοχή. Το BER της κωδικοποίησης grey είναι μικρότερο γιατί το σφάλμα κατά πάσα πιθανότητα θα αντιστοιχίσει το σύμβολο σε γειτονική περιχή, και στην κωδικοποίηση αυτή οι γειτονικές περιοχές διαφέρουν κατά ένα μόνο bit, διατηρόντας το BER χαμηλό. Σε πραγματικές εφαρμογές αναμένουμε το BER να μειώνεται με την αύξηση του SNR. Αυτό όμως εδώ δε συμβαίνει γιατί το σήμα που περνάει από το κανάλι παίρνει τιμές με εύρος περίπου [-1000, 1000], ενώ ο θόρυβος παίρνει τιμές στη χειρότερη στο [-10, 10] με αποτέλεσμα η προσθήκη του να μην επηρεάζει το αποτέλεσμα. Στα πρώτα 2 σχήματα βλέπουμε το σήμα θορύβου για SNR = 0, 20 και στο τελευταίο σχήμα βλέπουμε το σήμα.



Όπως φαίνεται, ο θόρυβος είναι αμελητέος μπροστά στο σήμα στην περίπτωση του ελάχιστου αλλά και του μέγιστου SNR, εξού και το σταθερό BER.

2) Υπολογισμός BER ως προς SNR

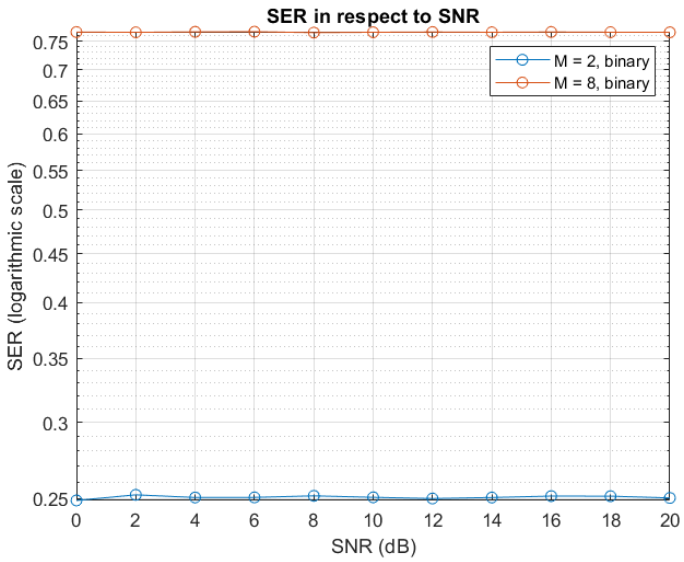
Δεδομένου πως έχει τρέξει ο κώδικας του προηγούμενου ερωτήματος, ο κώδικας του τρέχοντος ερωτήματος είναι ο παρακάτω.

```
% Question 3)

% Initialize SER matrix
SER = zeros(length(M), length(SNR));

% Compute SER for each M value using binary encoding
for i = 1:length(M)

    % Number of bits per symbol for current M
    bits_per_symbol = log2(M(i));

    % Pad the binary sequence with zeros if necessary
```

```
    padding_length = mod(-length(binary_sequence), bits_per_symbol);
    if padding_length > 0
        binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
    else
        binary_sequence_padded = binary_sequence;
    end

    % Group the binary sequence bits into symbols
    grouped_bits = reshape(binary_sequence_padded, bits_per_symbol, [])';

    % Loop through SNR values
    for j = 1:length(SNR)

        % Pass the binary sequence through the sender system
        bandpass_signal = transmitter(binary_sequence_padded, M(i), fc, Rs, 'binary');

        % Pass the bandpass signal through the channel
        noisy_signal = channel(bandpass_signal, SNR(j), M(i));

        % Pass the noisy signal through the receiver
        receiver_sequence = receiver(noisy_signal, M(i), fc, Rs, 'binary');

        % Group the receiver sequence bits into symbols
        grouped_received_bits = reshape(receiver_sequence, bits_per_symbol, [])';

        % Compute the number of symbol errors
        error_count = 0;
        for k = 1:size(grouped_bits, 1)  % Loop through all symbols
            if ~isequal(grouped_bits(k, :), grouped_received_bits(k, :))
                error_count = error_count + 1;
            end
        end

        % Compute the Symbol Error Rate (SER)
        SER(i, j) = error_count / size(grouped_bits, 1);  % Divide by number of symbols

    end
end

% Plot the SER curves for all M values in logarithmic scale
figure;
for i = 1:length(M)
    semilogy(SNR, SER(i, :), '-o', 'DisplayName', sprintf('M = %d, binary', M(i)));
    hold on;
end

% Finalize plot
xlabel('SNR (dB)');
ylabel('SER (logarithmic scale)');
title('SER in respect to SNR');
```

```
legend('show');
grid on;
```

Τρέξιμο του κώδικα εμφανίζει το εξής γράφημα:



Η δικαιολόγηση της συμπεριφοράς των καμπυλών είναι ίδια με αυτή του προηγούμενου ερωτήματος.

Παράρτημα: Κώδικας

Μέρος Α1

(Να εισαχθούν breakpoints στις γραμμές 51, 77, 103 και 116)

```
% Question i)

% Load the audio signal
[y, fs] = audioread('speech.wav');

% Transpose for correct dimensions
y = y';

% Initializing parameters
min_value = -1;
max_value = 1;
N = [2, 4, 8];

% Compute signal power
```

```
signal_power = mean(y.^2);

% Initialize MSE matrix
MSE = [];

figure
hold on

% Loop over each value of N
for i = 3:-1:1

    % Non Uniform Quantizer
    [xq, centers, D, lower_bound, upper_bound] = Lloyd_Max(y, N(i), min_value, max_value);
    quantized(i, :) = centers(xq);
    MSE(i, 1:length(D)) = D;
    for j = 1:length(D)
        % Compute the SQNR
        noise_power = D(j);
        SQNR(i, j) = 10 * log10(signal_power / noise_power);
    end

    % Plot the SQNR for the current N
    plot(SQNR(i, :), 'DisplayName', sprintf('N = %d bits', N(i)), 'LineWidth', 2);
end

% Customize the plot
xlabel('Number of Iterations');
ylabel('SQNR (dB)');
title('SQNR vs. Number of Iterations for Different N');
legend show;
grid on;
hold off;

% Question ii)

% Loop over each value of N
for i = 1:3

    % Uniform Quantizer
    [xq_u, centers_u] = my_quantizer(y, N(i), min_value, max_value);
    quantized_u(i, :) = centers_u(xq_u);

    % SQNR calculation
    noise_power_u = mean((y - quantized_u(i, :)).^2);
    signal_power_u = mean(y.^2);
    SQNR_u = 10 * log10(signal_power_u / noise_power_u);

    % MSE for later question
    MSE_u(i) = noise_power_u;
```

```
    row_sqnr = SQNR(i, :);
    non_zero_indices = find(row_sqnr ~= 0);
    % Get the last non-zero index
    last_non_zero_sqnr = row_sqnr(non_zero_indices(end));

    sprintf("N = %d bits. \n Uniform Quantizer: SQNR(dB) = %d \n" + ...
        " Non Uniform Quantizer SQNR(dB) = %d", N(i), SQNR_u, last_non_zero_sqnr)
end

% Question iii)

% For all values of N
for i = 1:3
    % Plot the signals side by side
    subplot(1, 3, 1);
    plot(y, 'r', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Original Signal');
    grid on;

    subplot(1, 3, 2);
    plot(quantized_u(i, :), 'g', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Uniform Quantizer');
    grid on;

    subplot(1, 3, 3);
    plot(quantized(i, :), 'b', 'LineWidth', 2);
    xlabel('Time');
    ylabel('Amplitude');
    title('Non Uniform Quantizer');
    grid on;

    sgtitle(sprintf('N = %d bits', N(i)));

    % Listen to the signals
    disp("Original Signal")
    sound(y, fs)
    pause(length(y) / fs)
    disp("Uniform Quantizer")
    sound(quantized_u(i, :))
    pause(length(quantized_u(i, :)) / fs)
    disp("Non Uniform Quantizer")
    sound(quantized(i, :))
    pause(length(quantized(i, :)) / fs)
end

% Question iv)
```

```
plot(N, MSE_u, 'r', 'LineWidth', 2);
hold on;
plot(N, MSE(:, end), 'b', 'LineWidth', 2);
hold off;
title('MSE of Uniform and Non Uniform Quantizers');
xlabel('N (Number of Bits)');
ylabel('MSE');
legend('Uniform', 'Non Uniform');
grid on;

% Uniform Quantizer
function [xq, centers] = my_quantizer(x, N, min_value, max_value)

    % Preprocess the signal to be within [min_value, max_value]
    x = max(min(x, max_value), min_value);

    % Calculate the number of quantization areas
    number_of_areas = 2^N;

    % Compute Delta (width of each area)
    Delta = (max_value - min_value) / number_of_areas;

    % Initialize arrays
    lower_bound = zeros(1, number_of_areas);
    upper_bound = zeros(1, number_of_areas);
    centers = zeros(1, number_of_areas);
    xq = zeros(1, length(x));

    % Calculate bounds and centers
    for i = 0:number_of_areas - 1
        lower_bound(number_of_areas - i) = min_value + i * Delta;
        upper_bound(number_of_areas - i) = min_value + (i + 1) * Delta;
        centers(number_of_areas - i) = min_value + (i + 0.5) * Delta;
    end

    % Quantize the signal
    for i = 1:length(x)
        for j = 1:number_of_areas
            if (x(i) >= lower_bound(j) && (x(i) < upper_bound(j) || j == 1))
                xq(i) = j;
                break;
            end
        end
    end
end

% Non Uniform Quantizer Using Lloyd-Max Algorihtm
function [xq, centers, D, lower_bound, upper_bound] = Lloyd_Max(x, N, min_value, max_value)
```

```
    % Tolerance for convergence
    tol = 1e-36;

    % Preprocess the signal to be within [min_value, max_value]
    x = max(min(x, max_value), min_value);

    % Calculate number of quantization areas
    number_of_areas = 2^N;

    % Compute Delta (width of each area)
    Delta = (max_value - min_value) / number_of_areas;

    % Initialize centers
    centers = zeros(1, number_of_areas);
    for i = 0:number_of_areas - 1
        centers(number_of_areas - i) = min_value + (i + 0.5) * Delta;
    end

    % Include extreme values to the centers temporarily
    centers = [min_value, centers, max_value];

    % Initialize vectors
    D = [];
    xq = zeros(1, length(x));
    lower_bound = zeros(1, number_of_areas);
    upper_bound = zeros(1, number_of_areas);

    iteration = 1;

    while true
        % Step 1: Calculate quantization bounds
        lower_bound(1) = centers(1);
        upper_bound(1) = (centers(2) + centers(3)) / 2;
        for i = 2:number_of_areas - 1
            lower_bound(i) = (centers(i) + centers(i+1)) / 2;
            upper_bound(i) = (centers(i+1) + centers(i+2)) / 2;
        end
        lower_bound(end) = (centers(end - 2) + centers(end - 1)) / 2;
        upper_bound(end) = centers(end);

        % Step 2: Quantize the signal
        for i = 1:length(x)
            [~, xq(i)] = min(abs(x(i) - centers(2:end - 1)));
        end

        % Step 3: Compute mean distortion
        quantized_values = centers(xq + 1);
        distortion = mean((x - quantized_values).^2);
        D = [D, distortion];
```

```
        % Step 4: Update centers
        for k = 1:number_of_areas
            points_in_area = x(x >= lower_bound(k) & x < upper_bound(k));
            if ~isempty(points_in_area)
                centers(k + 1) = mean(points_in_area);
            end
        end

        % Stop if the change in distortion is less than the tolerance
        if (iteration > 1 && abs(D(iteration) - D(iteration-1)) < tol)
            break;
        end

        % Increment iteration
        iteration = iteration + 1;
    end

    % Remove bounds from centers for final output
    centers = centers(2:end-1);

end
```

Μέρος Α2

(Να εισαχθούν breakpoints στις γραμμές 38 και 59)

```
% Question 2)

% Load the input signal (t)
load source.mat
t = t';

% length of the input
l = length(t);

% Initialize variables
min_value = -3.5;
max_value = 3.5;
p = [10, 1000];
N = [1, 2, 3];

% Main loop to plot the signal next to the prediction error
for i = 1:length(N)
    for j = 1:length(p)
        % Call the DPCM sender function
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);
```

```
        % Plot the input signal next to the prediction error
        figure
        plot(t)
        hold on
        plot(y)
        hold off
        title(['Input and Prediction Error signals (p=' num2str(p(j)) ', N=' num2str(N(i)) ')'])
        xlabel('Time Index')
        ylabel('Amplitude')
        legend('Input', 'Prediction Error')
    end
end

% Question 3)

% Initialize vectors
p = 5:10;
MSE = zeros(1, length(N));

% Main loop to plot the signal next to the prediction error
for j = 1:length(p)
    for i = 1:length(N)
        % Call the DPCM sender function
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);
        MSE(i) = mean((t - y).^2);
    end
    figure
    plot(MSE)
    title(['Mean Squared Error (MSE) (p=' num2str(p(j)) ')']);
    xlabel('Number of Bits (N)')
    ylabel('MSE')
    disp(a);
end

% Question 4)

% Initialize vectors
p = [5, 10];

for i = 1:length(N)
    for j = 1:length(p)
        [y, y_hat, a_hat, a] = dpcm_sender(t, p(j), N(i), min_value, max_value);
        y_hat_prime = dpcm_receiver(y_hat, a_hat, p);
        figure
        subplot(1, 2, 1)
        plot(t, 'r')
        xlabel('Time')
        ylabel('Amplitude')
        title('Input Signal')
        subplot(1, 2, 2)
```

```
        plot(y_hat_prime, 'b')
        xlabel('Time')
        ylabel('Amplitude')
        title('Reconstructed Signal')
        sgtitle(['p=' num2str(p(j)) ', N=' num2str(N(i))]);
    end
end

% Computes the predictor coefficients by solving the Yule-Walker equations
function a = predictor_coefficients(x, p)

    % Use the built-in aryule function to solve Yule-Walker equations
    % Requires Singal Proccessing Toolbox
    [a_full, ~] = aryule(x, p);

    % Remove the leading 1 to get only the AR coefficients
    a = a_full(2:end);

    % Reverse the sign as aryule returns -a
    a = -a;

end

% DPCM Sender function
function [y, y_hat, a_hat, a] = dpcm_sender(x, p, N, min_value, max_value)

    % Compute the length of the input
    l = length(x);

    % Error handling
    if p > l
        error('The predictor order cannot be greater than the signal length.');
    end

    % Initialize signal vectors
    y = zeros(1, l);
    y_hat = zeros(1, l);
    y_prime = zeros(1, l);
    y_hat_prime = zeros(1, l);

    % Compute the coefficients
    a = predictor_coefficients(x, p);

    % Quantize the coefficients to send them to the receiver
    [aq, centers] = my_quantizer(a, 8, -2, 2);
    a_hat = centers(aq);

    % Perform the first p iterations for hard-coded y_prime
    for n = 1:p
        y_prime(n) = x(n);
```

```
        y(n) = x(n) - y_prime(n);
        [yq, centers] = my_quantizer(y(n), N, min_value, max_value);
        y_hat(n) = centers(yq);
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

    % Perform the rest of the iterations
    for n = p + 1:l
        for i = 1:p
            y_prime(n) = y_prime(n) + a(i) * y_hat_prime(n - i);
        end
        y(n) = x(n) - y_prime(n);
        [yq, centers] = my_quantizer(y(n), N, min_value, max_value);
        y_hat(n) = centers(yq);
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

end

% DPCM Receiver function
function y_hat_prime = dpcm_receiver(y_hat, a_hat, p)

    % Compute the length of the input
    l = length(y_hat);

    % Error handling
    if p > l
        error('The predictor order cannot be greater than the signal length.');
    end

     % Initialize signal vectors
    y_prime = zeros(1, l);
    y_hat_prime = zeros(1, l);

    % Perform the first p iterations for hard-coded y_prime
    for n = 1:p
        y_prime(n) = y_hat(n);
        y_hat_prime(n) = y_prime(n);
    end

    % Perform the rest of the iterations
    for n = p + 1:l
        for i = 1:p
            y_prime(n) = y_prime(n) + a_hat(i) * y_hat_prime(n - i);
        end
        y_hat_prime(n) = y_prime(n) + y_hat(n);
    end

end
```

Μέρος Β

```
% Question 2)

% Create a binary sequence
binary_sequence = randi([0, 1], 1, 100000);

% Initialize the sender parameters
fc = 25e5;
Rs = 25e4;
M = [2, 8];
SNR = 0:2:20;

% Initialize BER matrix
BER = zeros(length(M) + 1, length(SNR));

% Compute BER for each M value using binary encoding
for i = 1:length(M)

    % Number of bits per symbol for current M
    bits_per_symbol = log2(M(i));

    % Pad the binary sequence with zeros if necessary
    padding_length = mod(-length(binary_sequence), bits_per_symbol);
    if padding_length > 0
        binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
    else
        binary_sequence_padded = binary_sequence;
    end

    % Loop through SNR values
    for j = 1:length(SNR)

        % Pass the binary sequence through the sender system
        bandpass_signal = transmitter(binary_sequence_padded, M(i), fc, Rs, 'binary');

        % Pass the bandpass signal through the channel
        noisy_signal = channel(bandpass_signal, SNR(j), M(i));

        % Pass the noisy signal through the receiver
        receiver_sequence = receiver(noisy_signal, M(i), fc, Rs, 'binary');

        % Compute the number of bit errors
        error_count = sum(binary_sequence_padded ~= receiver_sequence);

        % Compute the Bit Error Rate (BER)
```

```
        BER(i, j) = error_count / length(binary_sequence_padded);

    end
end

% Compute BER for M = 8 using grey encoding

% Number of bits per symbol for current M = 8
bits_per_symbol = log2(M(end));

% Pad the binary sequence with zeros if necessary
padding_length = mod(-length(binary_sequence), bits_per_symbol);
if padding_length > 0
    binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
else
    binary_sequence_padded = binary_sequence;
end

% Loop through SNR values
for j = 1:length(SNR)

    % Pass the binary sequence through the sender system
    bandpass_signal = transmitter(binary_sequence_padded, M(end), fc, Rs, 'grey');

    % Pass the bandpass signal through the channel
    noisy_signal = channel(bandpass_signal, SNR(j), M(end));

    % Pass the noisy signal through the receiver
    receiver_sequence = receiver(noisy_signal, M(end), fc, Rs, 'grey');

    % Compute the number of bit errors
    error_count = sum(binary_sequence_padded ~= receiver_sequence);

    % Compute the Bit Error Rate (BER)
    BER(end, j) = error_count / length(binary_sequence_padded);

end

% Plot the BER curves for all M values in logarithmic scale
figure;
for i = 1:length(M)
    semilogy(SNR, BER(i, :), '-o', 'DisplayName', sprintf('M = %d, binary', M(i)));
    hold on;
end
semilogy(SNR, BER(length(M) + 1, :), '-o', 'DisplayName', sprintf('M = %d, grey', M(end)));

% Finalize plot
xlabel('SNR (dB)');
ylabel('BER (logarithmic scale)');
title('BER in respect to SNR');
```

```
legend('show');
grid on;

% Question 3)

% Initialize SER matrix
SER = zeros(length(M), length(SNR));

% Compute SER for each M value using binary encoding
for i = 1:length(M)

    % Number of bits per symbol for current M
    bits_per_symbol = log2(M(i));

    % Pad the binary sequence with zeros if necessary
    padding_length = mod(-length(binary_sequence), bits_per_symbol);
    if padding_length > 0
        binary_sequence_padded = [binary_sequence, zeros(1, padding_length)];
    else
        binary_sequence_padded = binary_sequence;
    end

    % Group the binary sequence bits into symbols
    grouped_bits = reshape(binary_sequence_padded, bits_per_symbol, [])';

    % Loop through SNR values
    for j = 1:length(SNR)

        % Pass the binary sequence through the sender system
        bandpass_signal = transmitter(binary_sequence_padded, M(i), fc, Rs, 'binary');

        % Pass the bandpass signal through the channel
        noisy_signal = channel(bandpass_signal, SNR(j), M(i));

        % Pass the noisy signal through the receiver
        receiver_sequence = receiver(noisy_signal, M(i), fc, Rs, 'binary');

        % Group the receiver sequence bits into symbols
        grouped_received_bits = reshape(receiver_sequence, bits_per_symbol, [])';

        % Compute the number of symbol errors
        error_count = 0;
        for k = 1:size(grouped_bits, 1)  % Loop through all symbols
            if ~isequal(grouped_bits(k, :), grouped_received_bits(k, :))
                error_count = error_count + 1;
            end
        end

        % Compute the Symbol Error Rate (SER)
        SER(i, j) = error_count / size(grouped_bits, 1);  % Divide by number of symbols
```

```
    end
end

% Plot the SER curves for all M values in logarithmic scale
figure;
for i = 1:length(M)
    semilogy(SNR, SER(i, :), '-o', 'DisplayName', sprintf('M = %d, binary', M(i)));
    hold on;
end

% Finalize plot
xlabel('SNR (dB)');
ylabel('SER (logarithmic scale)');
title('SER in respect to SNR');
legend('show');
grid on;

% M-PAM receiver system
function receiver_sequence = receiver(noisy_signal, M, fc, Rs, encoding)

    % Compute basic parameters
    Tsymbol = 1 / Rs;
    fs = 4 * fc;
    Ts = 1 / fs;

    % Return the signal to the baseband using M-PAM demodulator
    baseband_signal = demodulator(noisy_signal, fc, Ts);

    % Perform matched filtering using M-PAM matched filter
    symbols = matched_filter(baseband_signal, Tsymbol, Ts);

    % Detect the received symbols using M-PAM detector
    indices = detector(symbols, M);

    % Map the symbols back to binary values using M-PAM demapper
    receiver_sequence = demapper(indices, M, encoding);

end

% AWGN Channel
function noisy_signal = channel(bandpass_signal, SNR, M)

    % Compute the noise variance σ^2
    variance = 2 / (log2(M) * 10^(SNR / 10));

    % Generate Gaussian noise
    noise = sqrt(variance) * randn(1, length(bandpass_signal));

    % Add the noise to the signal
```

```
    noisy_signal = bandpass_signal + noise;

end

% M-PAM transmitter system
function bandpass_signal = transmitter(binary_sequence, M, fc, Rs, encoding)

    % Compute basic parameters
    Tsymbol = 1 / Rs;
    fs = 4 * fc;
    Ts = 1 / fs;

    % Map the binary sequence to symbols using M-PAM mapper
    symbols = mapper(binary_sequence, M, encoding);

    % Shape the symbols using M-PAM pulse shaper
    baseband_signal = pulse_shaper(symbols, Tsymbol, Ts);

    % Take the signal to frequency fc using M-PAM modulator
    bandpass_signal = modulator(baseband_signal, fc, Ts);

end

% Demapper subsystem of the M-PAM receiver
function receiver_sequence = demapper(indices, M, encoding)

    % Number of bits per symbol
    bits_per_symbol = log2(M);

    % Check the encoding
    if strcmpi(encoding, "grey")

        % Convert grey to binary
        binary_indices = zeros(size(indices));
        binary_indices(:, 1) = indices(:, 1);
        for bit = 2:log2(M)
            binary_indices(:, bit) = xor(binary_indices(:, bit-1), indices(:, bit));
        end
        decimal_numbers = binary_indices;

    elseif strcmpi(encoding, "binary")

        % Perform no conversion
        decimal_numbers = indices;

    else
        error("Choose between 'grey' and 'binary' encoding.");
    end

    % Convert decimal numbers to binary sequence
```

```
    receiver_sequence = zeros(1, length(decimal_numbers) * bits_per_symbol);
    for i = 1:length(decimal_numbers)
        value = decimal_numbers(i);
        for bit = bits_per_symbol:-1:1
            receiver_sequence((i-1) * bits_per_symbol + (bits_per_symbol - bit + 1)) = bitget(v
        end
    end
end


% Mapper subsystem of the M-PAM receiver
function baseband_signal = demodulator(noisy_signal, fc, Ts)

    % Initialize time vector
    t = (0:length(noisy_signal)-1) * Ts;

    % Return the signal to the baseband
    baseband_signal = noisy_signal .* cos(2 * pi * fc * t);

end

% Detector subsystem of the M-PAM receiver
function indices = detector(symbols, M)

    % Compute the amplitude scaling factor A
    A = sqrt(3 / (M^2 - 1));

    % Map the amplitudes to the nearest M-PAM levels
    amplitudes = A * (-(M-1):2:(M-1));

    % Initialize a vector to store the indices of the nearest amplitudes
    indices = zeros(size(symbols));

    % Loop through each symbol in symbols received
    for i = 1:length(symbols)

        % Calculate the absolute difference between the current symbol and all amplitudes
        differences = abs(symbols(i) - amplitudes);

        % Find the index of the amplitude with the smallest difference
        [~, indices(i)] = min(differences);
    end

    % Adjust the indices to start from 0
    indices = indices - 1;

end

% Mapper subsystem of the M-PAM transmitter
function symbols = mapper(binary_sequence, M, encoding)
```

```
    % Check if M is a power of 2
    if mod(M, 2) ~= 0 || log2(M) ~= floor(log2(M))
        error('M must be a power of 2.');
    end

    % Number of bits per symbol
    bits_per_symbol = log2(M);

    % Compute amplitude scaling factor A (Mean Energy should be 1)
    A = sqrt(3 / (M^2 - 1));

    % Define M-PAM amplitudes based on A computed above
    amplitudes = A * (-(M-1):2:(M-1));

    % Divide the binary sequence into groups of bits of size "bits_per_symbol"
    bit_groups = reshape(binary_sequence, bits_per_symbol, [])';

    % Choose encoding
    if strcmpi(encoding, "grey")

        % Convert bit groups to decimal numbers
        decimal_numbers = bin2dec(num2str(bit_groups));

        % Perform Gray encoding (grey = number XOR number>>)
        indices = bitxor(decimal_numbers, floor(decimal_numbers / 2));

    elseif strcmpi(encoding, "binary")

        % Convert bit groups to decimal numbers
        indices = bin2dec(num2str(bit_groups));

    else
        % Error, unknown encoding
        error("Choose between 'grey' and 'binary' encoding.");
    end

    % Map the indices to the amplitudes
    symbols = amplitudes(indices + 1);

end

% Matched filter subsystem of the M-PAM receiver
function symbols = matched_filter(baseband_signal, Tsymbol, Ts)

    % Compute the rectangular pulse as round(Tsymbol / Ts) samples
    number_of_samples = round(Tsymbol / Ts);
    amplitude = sqrt(2 / Tsymbol);
    pulse = amplitude * ones(1, number_of_samples);
```

```
    % Perform matched filtering
    match_filtered_signal = conv(baseband_signal, pulse, 'same');

    % Recover the symbols (the corellation is higher at the middles of the intervals)
    symbol_indices = round(number_of_samples/2:number_of_samples:length(match_filtered_signal));
    symbols = match_filtered_signal(symbol_indices);

end

% Modulator subsystem of the M-PAM transmitter
function bandpass_signal = modulator(baseband_signal, fc, Ts)

    % Initialize time vector
    t = (0:length(baseband_signal)-1) * Ts;

    % Take the signal to the frequency fc
    bandpass_signal = baseband_signal .* cos(2 * pi * fc * t);

end

% Pulse shaper subsystem of the M-PAM transmitter
function baseband_signal = pulse_shaper(symbols, Tsymbol, Ts)

    % Compute the rectangular pulse as round(Tsymbol / Ts) samples
    number_of_samples = round(Tsymbol / Ts);
    amplitude = sqrt(2 / Tsymbol);
    pulse = amplitude * ones(1, number_of_samples);

    % Make the symbols last as long as the pulse (zero padding in between samples)
    symbols_upsampled = upsample(symbols, length(pulse));

    % Convolve (multiply each symbol and then add) with the pulse
    baseband_signal = conv(symbols_upsampled, pulse, 'same');

end
```

Κωνσταντίνος Αναστασόπουλος

1093320