ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Ονοματεπώνυμο: Βένος Αναστάσιος

ΑΜ: 1067536

Έτος: 7ο

Δομή Αναφοράς

Η αναφορά χωρίζεται σε 2 μέρη: την αναφορά και τον κώδικα. Στην αναφορά, πρώτα υπάρχει το θεωρητικό μέρος, όπου εξηγούνται μερικά πράγματα για την ερώτηση και στη συνέχεια υπάρχει η υλοποίηση, όπου εξηγείται ο κώδικας και επιδεικνύονται τα αποτελέσματα. Στο δεύτερο μέρος, υπάρχει ο κώδικας για κάθε ερώτημα. **Χρειάστηκε να εγκαταστήσω τα communication toolbox και deep learning toolbox για μερικές συναρτήσεις.** Για το κάθε ερώτημα είναι έτοιμος, δηλαδή, απλή αντιγραφή και εκτέλεση.

ΜΕΡΟΣ Α

Ερωτήματα Μέρους Α1

1.

a.

Θεωρητικά

Το Pulse Code Modulation (PCM) είναι ένας ψηφιακός τρόπος για τη μετάδοση αναλογικών δεδομένων. Μετατρέπει ένα αναλογικό σήμα σε ψηφιακή μορφή. Χρησιμοποιώντας PCM, είναι δυνατή η ψηφιοποίηση όλων των μορφών αναλογικών δεδομένων.

Το αναλογικό σήμα δειγματοληπτείται σε τακτά χρονικά διαστήματα. Ο ρυθμός δειγματοληψίας είναι πολλαπλάσιος της μέγιστης συχνότητας του αναλογικού σήματος. Το στιγμιαίο πλάτος του αναλογικού σήματος σε κάθε δειγματοληψία στρογγυλοποιείται στο πλησιέστερο από διάφορα συγκεκριμένα, προκαθορισμένα επίπεδα (κβάντιση). Ο αριθμός των επιπέδων είναι πάντα μια δύναμη του 2. Η έξοδος ενός PCM είναι μια σειρά δυαδικών αριθμών, καθένας από τους οποίους αντιπροσωπεύεται από κάποια δύναμη των 2 bits. Στο κύκλωμα επικοινωνίας, ο PCM μετατρέπει τους δυαδικούς αριθμούς σε παλμούς που έχουν τα ίδια κβαντικά επίπεδα με εκείνα του διαμορφωτή. Αυτοί οι παλμοί υποβάλλονται σε περαιτέρω επεξεργασία για την αποκατάσταση της αρχικής αναλογικής κυματομορφής.

Το PCM αποτελείται από 3 βασικά βήματα:

1. Δειγματοληψία
2. Κβαντισμός
3. Κωδικοποίηση

Τα θεμέλια του PCM βασίζονται στο θεώρημα δειγματοληψίας του Nyquist: *«Αν η συχνότητα δειγματοληψίας είναι τουλάχιστον 2 φορές μεγαλύτερη της μέγιστης συχνότητας του αρχικού σήματος, τότε το δείγμα περιέχει όλη τη βασική πληροφορία του αρχικού σήματος και η ανακατασκευή είναι δυνατή χωρίς απώλειες.»*

Ο κβαντισμός διακρίνεται σε 2 τύπους:

1. Ομοιόμορφος
2. Μη-Ομοιόμορφος

Ο σκοπός ενός PCM είναι να μεταδώσει τη τιμή ενός δείγματος τάσης σε μια συγκεκριμένη χρονική στιγμή. Το επόμενο βήμα είναι η αντιστοίχιση μιας δυαδικής ακολουθίας σε κάθε ένα από αυτά τα δείγματα τάσης. Για παράδειγμα, υποθέτουμε ότι ένα σύστημα έχει τιμές τάσης από -1V μέχρι +1V. Ανάμεσα σε αυτές τις τιμές, μπορεί να υπάρχουν άλλες άπειρες τιμές.

Για αυτό το λόγο, χωρίζουμε αυτό το διάστημα σε έναν διακριτό αριθμό, τον οποίο ονομάζουμε επίπεδα.

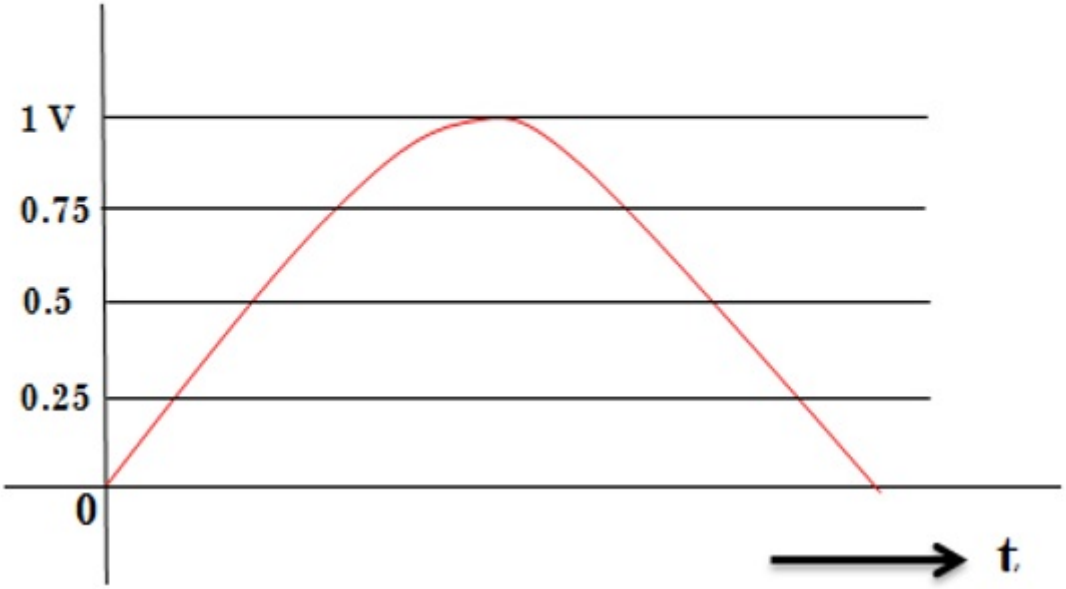
Τα επίπεδα βρίσκονται με τον τύπο: Δ = max_value – min_value/2^N, όπου ο αριθμητής είναι το ελάχιστο και μέγιστο άκρο του σήματος και N είναι τα bits της κβάντισης. Το Δ είναι το βήμα. Για να βρούμε τα επίπεδα, θα ξεκινήσουμε από την ελάχιστη τιμή και με βήμα Δ θα φτάσουμε στη μέγιστη. Έτσι, το διάστημα θα χωριστεί σε επίπεδα. Στη συνέχεια, η κάθε τιμή της δειγματοληψίας θα αντιστοιχηθεί στο κοντινότερο επίπεδο. Έτσι λοιπόν, έχουμε τον κβαντισμό.

Σε αυτή τη διαδικασία όμως, υπάρχει και το σφάλμα κβαντοποίησης. Για παράδειγμα, μια τιμή δειγματοληψίας 0.37 μπορεί να αντιστοιχηθεί στο πιο κοντινό επίπεδο 0.4. Έτσι, υπάρχει 0.03 σφάλμα. Για αυτό τον λόγο, όσα περισσότερα επίπεδα υπάρχουν, τόσο λιγότερο σφάλμα θα έχουμε.

Οι τιμές των επιπέδων πρέπει να αντιστοιχηθούν σε μια δυαδική τιμή. Εφόσον δουλεύουμε με δυαδικό σύστημα, επιλέγουμε το συνολικό αριθμό των επιπέδων να είναι δυνάμεις του 2 (πχ 2, 4, 8, 16 κλπ). Αυτό διευκολύνει τη δυαδική κωδικοποίηση. Αν διαλέξουμε 4 επίπεδα, θα ήταν τα 00, 01, 10, 11 αντίστοιχα. Αυτός είναι 2 bit κώδικας. Παρατηρούμε λοιπόν, ότι δεδομένου ενός κώδικα n bits, υπάρχουν 2^n επίπεδα.

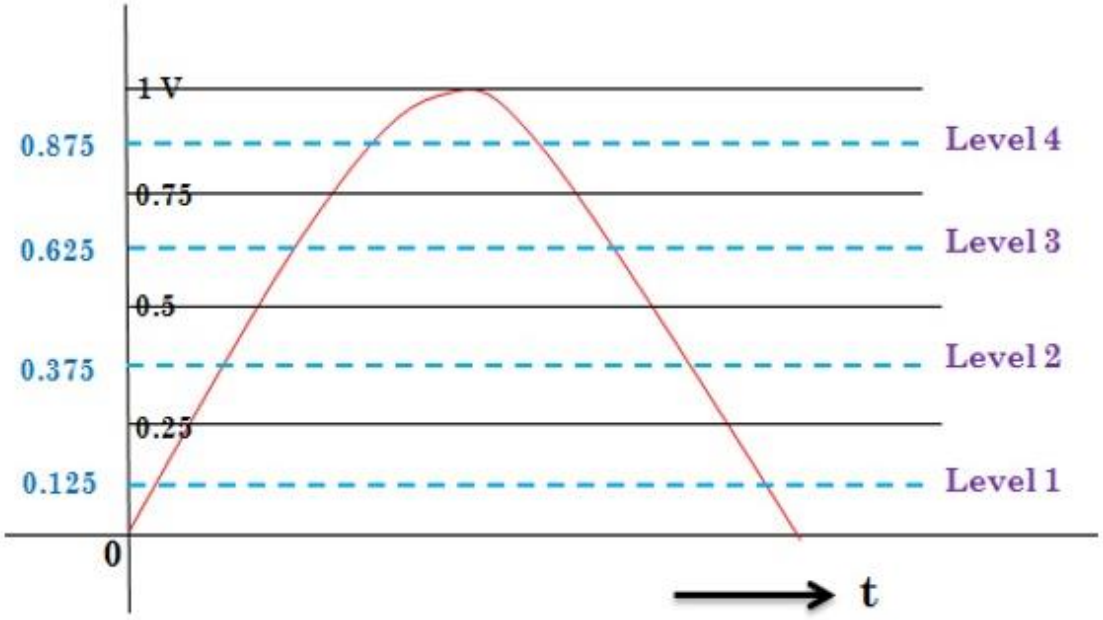
Παρακάτω, βλέπουμε εικόνες από την διαδικασία:

Step 1: Step size $S = \frac{V_H - V_L}{2^n} = \frac{1-0}{4} = \frac{1}{4} = 0.25$

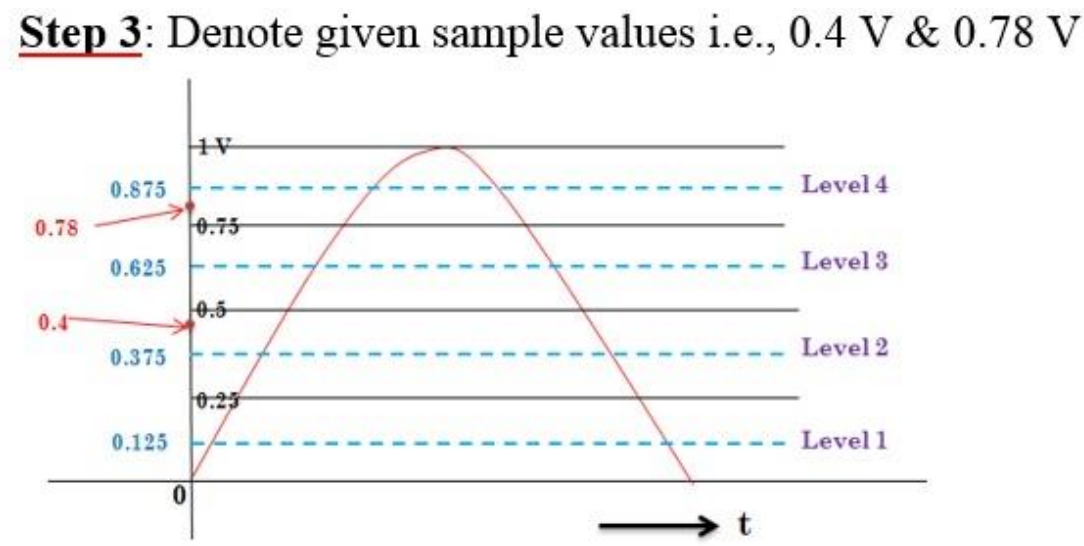


Σχήμα 1: Εύρεση βήματος Δ [1]

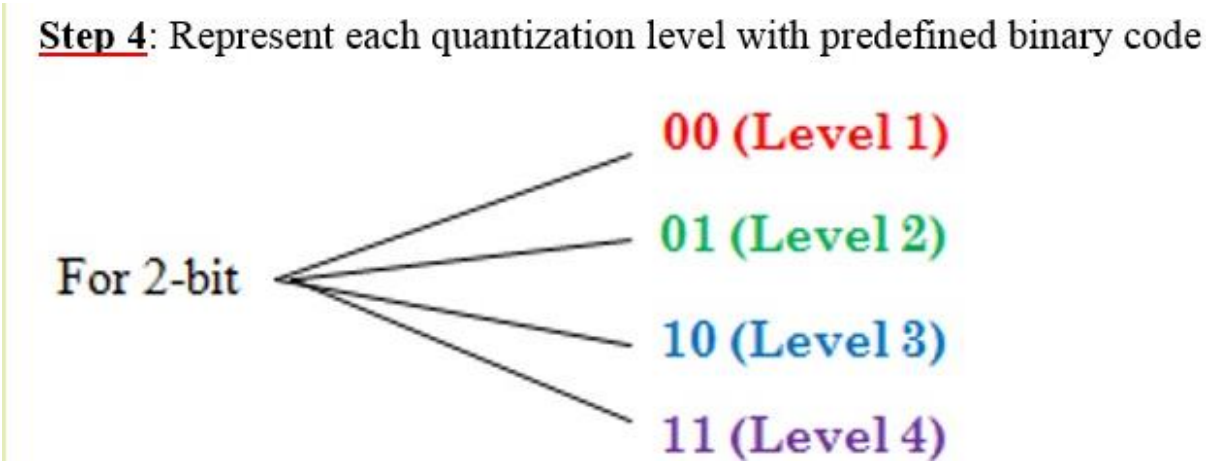
Step 2: Draw mid-lines, which represents quantization levels



Σχήμα 2: Εύρεση επιπέδων [1]



Σχήμα 3: Αντιστοίχιση δειγμάτων στα επίπεδα [1]



Σχήμα 4: Αντιστοίχιση επιπέδων στην δυαδική του τιμή [1]

Υλοποίηση

Σε αυτό το ερώτημα, υλοποιώ έναν ομοιόμορφο κβαντιστή για ένα αναλογικό σήμα συνεχούς χρόνου, ακολουθώντας τα 4 βήματα της εργαστηριακής άσκησης.

Δημιουργώ τη συνάρτηση “function [xq, centers] = my_quantizer(x, N, min_value, max_value)”, στην οποία υλοποιούνται τα 4 βήματα.

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

Αρχικά, η συνάρτηση διαβεβαιώνει ότι η τιμή της μεταβλητής max_value να είναι μεγαλύτερη της τιμής της μεταβλητής min_value και η τιμή του N να είναι θετικός ακέραιος.

```
% Διασφάλιση ότι min_value < max_value
assert(min_value < max_value, 'min_value must be smaller than max_value');

% Διασφάλιση ότι N > 0
assert(N > 0 && floor(N) == N, 'N must be a positive integer');
```

Στη συνέχεια, η μεταβλητή x_confined διαβεβαιώνει ότι οι τιμές του σήματος x θα βρίσκονται στο διάστημα [min_value, max_value] και όσες είναι εκτός, θα λαμβάνουν την πλησιέστερη ακραία αποδεκτή τιμή.

```
% Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
% max_value]
x_confined = max(min(x, max_value), min_value);
```

Στο επόμενο βήμα, υπολογίζονται οι παράμετροι της κβάντισης, δηλαδή το L (αριθμός επιπέδων), το Δ (βήμα) και ορίζονται τα κέντρα κάθε επιπέδου.

```
% Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
% περιοχής
L = 2^N;                 % Αριθμός επιπέδων κβαντισμού
delta = (max_value - min_value) / L;
centers = min_value + (0:L-1) * delta;
```

Έπειτα, το xq_indices ορίζει σε ποια περιοχή ανήκει το κάθε δείγμα, προσθέτοντας το e για την αποφυγή θεμάτων στρογγυλοποίησης.

```
% Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
epsilon = 1e-12; % Μικρό e για αποφυγή σφάλματα στρογγυλοποίησης
xq_indices = round((x_confined - min_value + epsilon) / delta);
xq_indices = max(min(xq_indices, L-1), 0);
```

Τέλος, το διάνυσμα xq χρησιμοποιείται ως δείκτης στο διάνυσμα centers για να πάρουμε το κβαντισμένο σήμα.

```
% Βήμα 4: xq ως δείκτης στο διάνυσμα centers
xq = centers(xq_indices + 1);
xq = reshape(xq, size(x));

end
```

Εκτός συνάρτησης, ορίζω τις παραμέτρους εισόδου, που είναι το αναλογικό σήμα x μαζί με τις απαραίτητες παραμέτρους του (συχνότητα F, συχνότητα δειγματοληψίας Fs, με την οποία υπολογίζω τη περίοδο δειγματοληψίας Ts=1/Fs και το βήμα t για τα δείγματα), το N που είναι ο αριθμός των bits για την κβάντιση και την ελάχιστη και μέγιστη επιτρεπτή τιμή που μπορεί να λάβει το σήμα (min-max value).

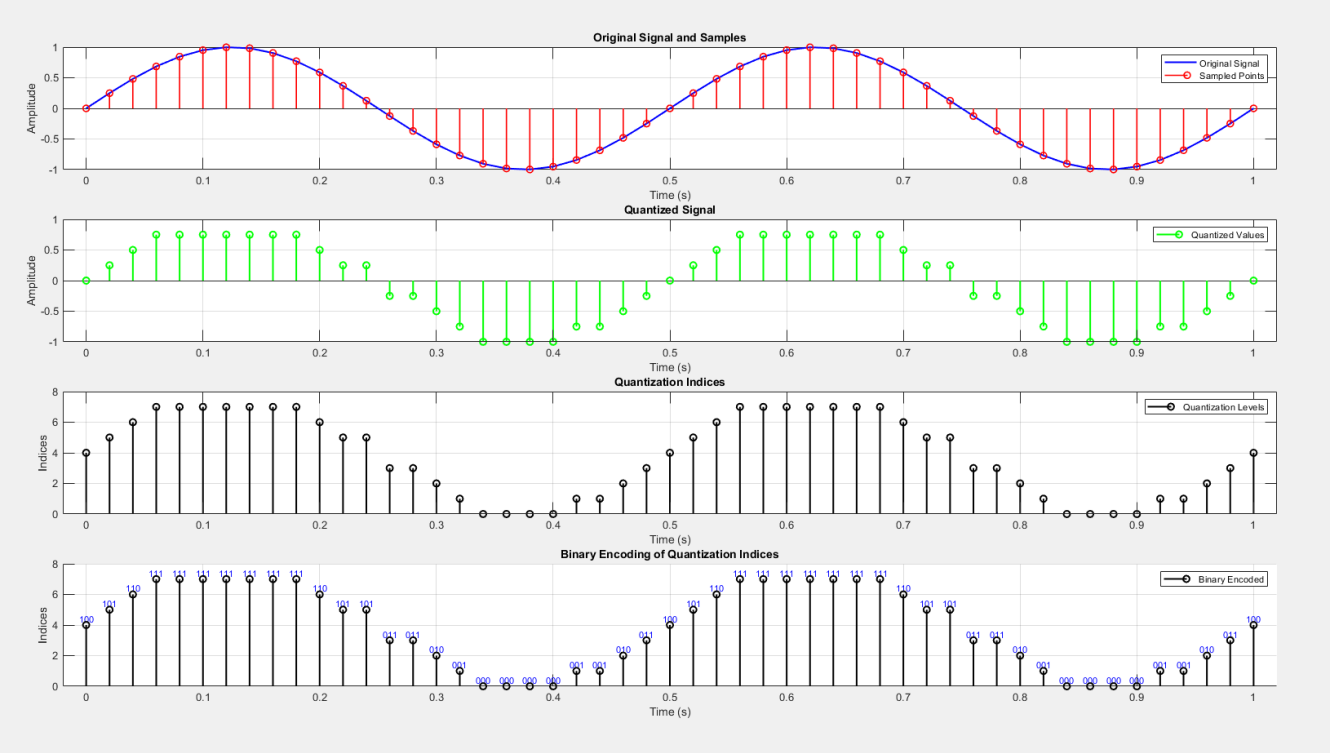
```
% Αρχικοποίηση Παραμέτρων Σήματος
F = 4;                  % Συχνότητα αναλογικού σήματος (Hz)
Fs = 50;                % Συχνότητα δειγματοληψίας (Hz)
Ts = 1/Fs;              % Περίοδος δειγματοληψίας
t = 0:Ts:1;             % Βήμα δειγματοληψίας
x = sin(2*pi*F*t);      % Αναλογικό σήμα
```

Καλείται η συνάρτηση και μας δίνει τις εξόδους xq (κβαντισμένο σήμα) και centers (κέντρα των επιπέδων).

```
% Κλήση Συνάρτησης
[xq, centers] = my_quantizer(x, N, min_value, max_value);
```

Τέλος, δημιουργώ 4 σχήματα για να αναπαραστήσω το αρχικό σήμα x με τα δείγματα του, το κβαντισμένο σήμα, τους κβαντισμένους δείκτες και την δυαδική κωδικοποίηση, η οποία προέκυψε με τη συνάρτηση dec2bin από τους δείκτες xq_indices.

```
% Plot
figure;

% Αρχικό σήμα και δείγματα
subplot(4, 1, 1);
plot(t, x, 'b', 'LineWidth', 1.5); hold on;
stem(t, x, 'r', 'LineWidth', 1.2); grid on;
title('Original Signal and Samples');
xlabel('Time (s)'); ylabel('Amplitude');
legend('Original Signal', 'Sampled Points');

% Κβαντισμένο σήμα
subplot(4, 1, 2);
stem(t, xq, 'g', 'LineWidth', 1.5); grid on;
title('Quantized Signal');
xlabel('Time (s)'); ylabel('Amplitude');
legend('Quantized Values');

% Δείκτες κβαντισμού
subplot(4, 1, 3);
stem(t, xq_indices, 'k', 'LineWidth', 1.5); grid on;
title('Quantization Indices');
xlabel('Time (s)'); ylabel('Indices');
legend('Quantization Levels');

% Δυαδική Κωδικοποίηση
binary_encoded = dec2bin(xq_indices, N);
subplot(4, 1, 4);
hold on;
for i = 1:length(t)
    text(t(i), xq_indices(i), binary_encoded(i, :), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', ...
        'FontSize', 8, 'Color', 'blue');
end
stem(t, xq_indices, 'k', 'LineWidth', 1.5); grid on;
title('Binary Encoding of Quantization Indices');
xlabel('Time (s)'); ylabel('Indices');
legend('Binary Encoded');
```



Σχήμα 5: PCM με ομοιόμορφο κβαντιστή

b.

Σε έναν μη-ομοιόμορφο κβαντιστή, το βήμα Δ δεν είναι σταθερό. Χρησιμοποιείται μικρότερο βήμα σε περιοχή όπου το πλάτος του σήματος είναι πιο πιθανό να βρίσκεται και μεγαλύτερο βήμα σε περιοχή με μικρότερη πιθανότητα. Αυτό γίνεται για να μειωθεί το σφάλμα.

Ο αλγόριθμος Lloyd-Max κάνει αυτό ακριβώς. Σε περιοχές με μεγαλύτερη πιθανότητα δημιουργεί περισσότερα επίπεδα, άρα μικρότερα intervals και το ανάποδο για μικρότερες πιθανότητες.

Ο αλγόριθμος λειτουργεί με επαναλήψεις. Αρχικά, χωρίζει το διάστημα του σήματος σε ίσα intervals και δίνει μια τιμή στο καθένα, όπως ακριβώς και ο ομοιόμορφος κβαντιστής. Τοποθετεί όρια στα κέντρα των intervals κι έτσι δημιουργούνται ζώνες. Στη συνέχεια κβαντίζει τη τιμή του δείγματος, αναλόγως σε ποια ζώνη ανήκει. Έπειτα, για κάθε ζώνη, ενημερώνει το κέντρο της βάσει τις τιμές που ήδη έχει. Υπολογίζει το μέσο όρο των τιμών της και αυτός θα είναι το νέο κέντρο. Όλη αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου: $|Di – Di\text{-}1| < ε$, όπου Di είναι η μέση

παραμόρφωση της i επανάληψης. Ουσιαστικά, το ε είναι ένα threshold, και όταν ικανοποιηθεί η παραπάνω συνθήκη, περισσότερες επαναλήψεις δε θα μειώσουν αρκετά το σφάλμα κβάντισης. Επίσης, είναι ένα όριο για να αποτρέψει τον αλγόριθμο να τρέχει για πάντα. Όσο πιο μικρή τιμή έχει, τόσες περισσότερες επαναλήψεις θα γίνουν, αλλά θα πετύχει και μικρότερο σφάλμα.

Υλοποίηση

Για το ερώτημα αυτό, δημιουργώ τη συνάρτηση function [xq, centers, D] = my_lloyd_max(x, N, min_value, max_value) και ακολουθώ τα βήματα που αναγράφονται στην εργαστηριακή άσκηση.

```
% Lloyd-Max Αλγόριθμος
function [xq, centers, D] = Lloyd_max(x, N, min_value, max_value)
```

Η έξοδος της συνάρτησης είναι το xq που είναι ίδιο με αυτό του ομοιόμορφου κβαντιστή, το centers που είναι τα συνεχώς ανανεωμένα κέντρα και το D που είναι ένα διάνυσμα που αποθηκεύει τη παραμόρφωση κάθε επανάληψης.

Ορίζονται πάλι οι παράμετροι (επίπεδα L, βήμα Δ και τα κέντρα), αλλά και τα όρια boundaries μεταξύ των επιπέδων, που βεβαιώνουν ότι όλες οι τιμές σε ένα διάστημα θα αντιστοιχηθούν σε ένα κέντρο.

```
% Αρχικοποίηση ομοιόμορφων παραμέτρων
L = 2^N;
delta = (max_value - min_value) / L;
centers = linspace(min_value + delta / 2, max_value - delta / 2, L);
boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];

D = []; % Διάνυσμα που κρατάει τις παραμορφώσεις
```

Στο πρώτο βήμα της συνάρτησης, αντιστοιχίζεται κάθε τιμή του σήματος x με τη τιμή του κέντρου του επιπέδου που ανήκει. Τα boundaries καθορίζουν τα Intervals που ανήκουν οι τιμές.

```
% Επαναλήψεις
while true
    % Βήμα 1: Υπολογισμός ορίων ζωνών κβαντισμού
    xq = zeros(size(x));
    for i = 1:length(x)
        for j = 1:length(boundaries) - 1
            if x(i) >= boundaries(j) && x(i) < boundaries(j + 1)
                xq(i) = centers(j);
                break;
            end
        end
    end
```

Στο δεύτερο βήμα, υπολογίζεται η παραμόρφωση μεταξύ του αρχικού σήματος x και του κβαντισμένου xq. Η τιμή της παραμόρφωσης αποθηκεύεται στο διάνυσμα D.

```
% Βήμα 2: Υπολογισμός παραμόρφωσης
distortion = mean((x - xq).^2);
D = [D, distortion];
```

Στο τρίτο βήμα, ενημερώνονται τα επίπεδα κβαντισμού. Για όλες τις τιμές του σήματος x που βρίσκονται σε ένα κέντρο center, βρίσκεται ο μέσος όρος τους και αυτό θα είναι το νέο κέντρο. Αν δεν υπάρχουν τιμές, το κέντρο δεν αλλάζει.

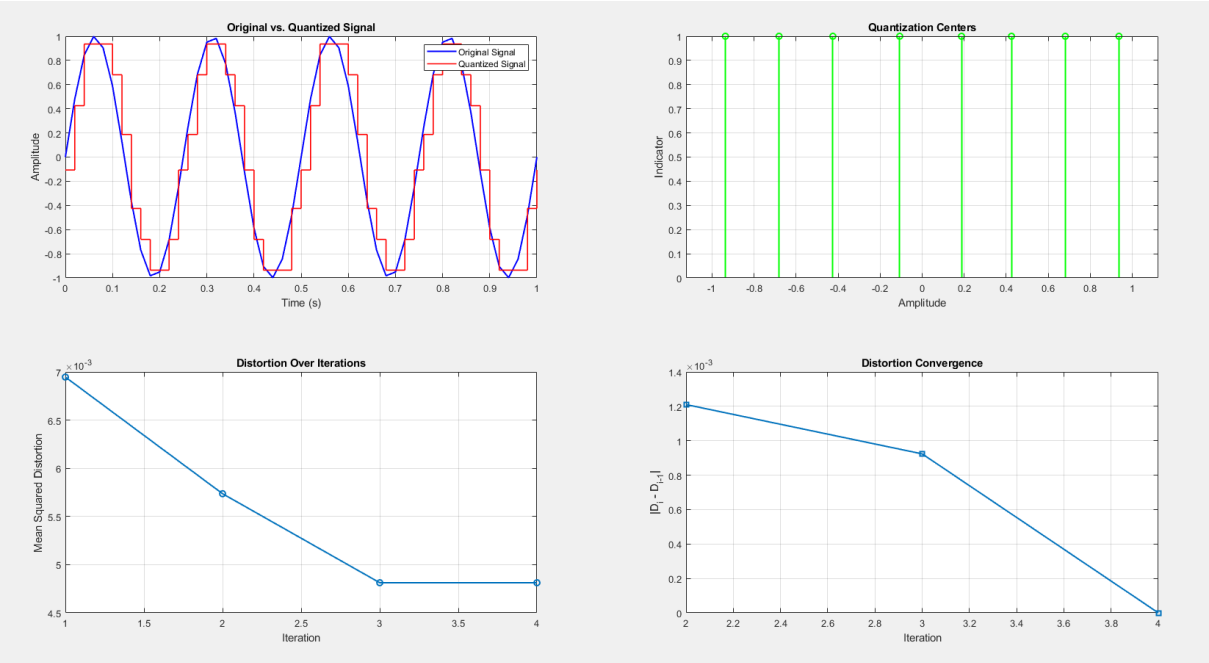
```
% Βήμα 3: Ενημέρωση επιπέδων κβαντισμού
new_centers = zeros(1, L);
counts = zeros(1, L);
for i = 1:length(x)
    for j = 1:L
        if xq(i) == centers(j)
            new_centers(j) = new_centers(j) + x(i);
            counts(j) = counts(j) + 1;
            break;
        end
    end
end

for j = 1:L
    if counts(j) > 0
        new_centers(j) = new_centers(j) / counts(j);
    else
        new_centers(j) = centers(j);
    end
end
```

Τέλος, υπάρχει ο έλεγχος για σύγκλιση. Αν η παραμόρφωση μεταξύ δύο διαδοχικών επαναλήψεων είναι μικρότερη του threshold ε, τότε σταματάει η λούπα. Αλλιώς, ενημερώνονται τα κέντρα.

```
        % Σύγκλιση για να σταματήσει η λούπα
        if length(D) > 1 && abs(D(end) - D(end-1)) < 1e-6
            break;
        end

        centers = new_centers;
        boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];
    end
end
```

Κάνω plot 4 σχήματα. Το πρώτο δείχνει το αρχικό σήμα και το κβαντισμένο, το δεύτερο δείχνει τα τελικά κέντρα και το τρίτο δείχνει τη μέση παραμόρφωση σε κάθε επανάληψη. Όπως παρατηρούμε, η μέση παραμόρφωση μειώνεται σε κάθε επανάληψη, μέχρι να φτάσει σε ένα σημείο όπου σταθεροποιείται και σταματάνε οι επαναλήψεις του αλγορίθμου.

```
% Plot
figure;

% Plot 1: Αρχικό και Κβαντισμένο Σήμα
subplot(3, 1, 1);
plot(t, x, 'b', 'LineWidth', 1.5); hold on;
stairs(t, xq, 'r', 'LineWidth', 1.2);
title('Original vs. Quantized Signal');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Original Signal', 'Quantized Signal');
grid on;

% Plot 2: Κέντρα Κβαντισμού
subplot(3, 1, 2);
stem(centers, ones(size(centers)), 'g', 'LineWidth', 1.5);
title('Quantization Centers');
xlabel('Amplitude');
ylabel('Indicator');
grid on;

% Plot 3: Παραμόρφωση σε κάθε επανάληψη
subplot(3, 1, 3);
plot(1:length(D), D, '-o', 'LineWidth', 1.5);
title('Distortion Over Iterations');
xlabel('Iteration');
ylabel('Mean Squared Distortion');
grid on;
```



Σχήμα 6: PCM με τον αλγόριθμο Lloyd-Max

2.

i.

Θεωρητικά

Ο ήχος αποτελείται από κυματομορφές, που αποτελούνται από την παρεμβολή κυμάτων διαφορετικών συχνοτήτων και πλάτους. Για την αναπαράσταση αυτών των κυματομορφών μέσα σε ψηφιακά μέσα, οι κυματομορφές πρέπει να δειγματοληπτούνται σε ρυθμούς που μπορούν (τουλάχιστον) να αναπαραστήσουν ήχους της υψηλότερης συχνότητας που θέλετε να αναπαραγάγετε, και πρέπει επίσης να αποθηκεύουν αρκετό βάθος bit για να αναπαραστήσουν το κατάλληλο πλάτος (ένταση και απαλότητα) των κυματομορφών σε όλο το δείγμα ήχου.

Η ικανότητα μιας συσκευής επεξεργασίας ήχου να αναδημιουργεί συχνότητες είναι γνωστή ως απόκριση συχνότητας και η ικανότητά της να δημιουργεί την κατάλληλη ένταση και απαλότητα είναι γνωστή ως δυναμικό εύρος. Μαζί αυτοί οι όροι αναφέρονται συχνά ως πιστότητα μιας συσκευής ήχου. Μια κωδικοποίηση, στην απλούστερη μορφή της, είναι ένα μέσο με το οποίο μπορεί να ανακατασκευαστεί ο ήχος χρησιμοποιώντας αυτές τις δύο βασικές αρχές, καθώς και να είναι σε θέση να αποθηκεύει και να μεταφέρει αποτελεσματικά τα δεδομένα αυτά.

Ο ήχος υπάρχει ως αναλογική κυματομορφή. Ένα τμήμα ψηφιακού ήχου προσεγγίζει αυτό το αναλογικό κύμα με τη δειγματοληψία του πλάτους αυτού του αναλογικού κύματος με αρκετά γρήγορο ρυθμό ώστε να μιμείται τις εγγενείς συχνότητες του κύματος. Ο ρυθμός δειγματοληψίας ενός τμήματος ψηφιακού ήχου καθορίζει τον αριθμό των δειγμάτων που λαμβάνονται από το υλικό της πηγής του ήχου (ανά δευτερόλεπτο)- ένας υψηλός ρυθμός δειγματοληψίας αυξάνει την ικανότητα του ψηφιακού ήχου να αναπαριστά πιστά τις υψηλές συχνότητες.

Μια από τις πιο δημοφιλείς τεχνικές ψηφιακού ήχου είναι γνωστή ως διαμόρφωση παλμικού κώδικα (ή PCM). Ο ήχος δειγματοληπτείται σε καθορισμένα χρονικά διαστήματα και το πλάτος του δειγματοληπτούμενου

κύματος σε εκείνο το σημείο αποθηκεύεται ως ψηφιακή τιμή χρησιμοποιώντας το βάθος bit του δείγματος.

Το SQNR (Signal to Quantization Noise Ratio) είναι ένα μέτρο που χρησιμοποιείται στη ψηφιακή επεξεργασία σημάτων για να αξιολογήσει τη ποιότητα του κβαντισμένου σήματος σε σχέση με το αρχικό σήμα. Δείχνει την αναλογία της ισχύς του σήματος με τον θόρυβο κβαντισμού.

Ο τύπος υπολογισμού είναι ο εξής: SQNR (db) = 10log(10) (Ισχύς Σήματος / Παραμόρφωση).

Όσο πιο υψηλό είναι, τόσο καλύτερη είναι η ποιότητα του σήματος και τόσο λιγότερο σφάλμα κβαντισμού υπάρχει. Αυτό μπορούμε να το επιτύχουμε με το να δημιουργήσουμε περισσότερα επίπεδα κβαντισμού N.

Υλοποίηση

Σε αυτό το ερώτημα, κωδικοποιώ τη πηγή ήχου με τον αλγόριθμο Lloyd-Max που υλοποίησα παραπάνω.

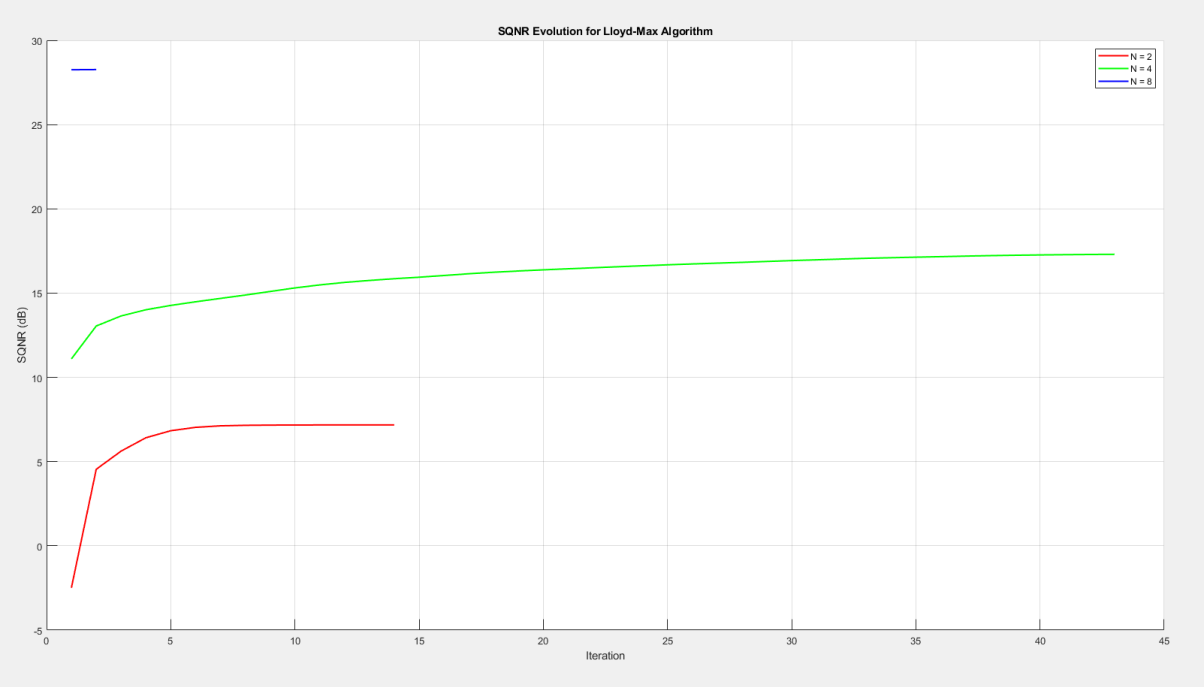
Αρχικά, φορτώνουμε το σήμα ήχου “speech.wav” και το κανονικοποιούμε στις τιμές [-1,1]. Έπειτα, ορίζουμε τις τιμές min_value, max_value, το threshold ε και τα 3 διαφορετικά N (2, 4, 8).

```
% Φόρτωση του αρχείου ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y));

% Παράμετροι
min_value = -1;
max_value = 1;
epsilon = 1e-6; % Threshold σύγκλισης
bits = [2, 4, 8]; % Bit-depth
colors = ['r', 'g', 'b']; % Χρώματα για το Plot
```

Στη συνέχεια, για κάθε N, καλούμε τη συνάρτηση Lloyd-Max, υπολογίζεται το SQNR και τέλος γίνεται plot το SQNR για κάθε επανάληψη.

```
% Αποθήκευση τιμών SQNR
SQNR_all = cell(length(bits), 1);

figure;
hold on;

for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση Συνάρτησης
    [xq, centers, D] = Lloyd_max(y, N, min_value, max_value);

    % Υπολογισμός SQNR σε κάθε επανάληψη
    SQNR = 10 * log10(mean(y.^2) ./ D);

    % Αποθήκευση τιμών SQNR για plot
    SQNR_all{idx} = SQNR;

    % Plot SQNR
    plot(1:length(SQNR), SQNR, 'Color', colors(idx), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('N = %d', N));
end

% Plot
xlabel('Iteration');
ylabel('SQNR (dB)');
title('SQNR Evolution for Lloyd-Max Algorithm');
legend show;
grid on;
hold off;
```

Όπως παρατηρούμε, το SQNR βελτιώνεται όσο περνάνε οι επαναλήψεις. Αυτό είναι φυσιολογικό, εφόσον ο αλγόριθμος ανανεώνει συνέχεια τα κέντρα των επιπέδων, έτσι ώστε να μειώνεται η παραμόρφωση. Μετά από ένα σημείο, η γραμμή γίνεται σταθερή, διότι επέρχεται η σύγκλιση και ο αλγόριθμος έχει βελτιστοποιήσει τα επίπεδα σε σημείο όπου περισσότερες επαναλήψεις επιφέρουν ελάχιστη βελτίωση. Όσο το N αυξάνεται (περισσότερα επίπεδα κβαντισμού), τόσο καλύτερο είναι το SQNR, επειδή το σφάλμα κβάντισης μειώνεται, εφόσον το σήμα παρίσταται πιο εύστοχα.



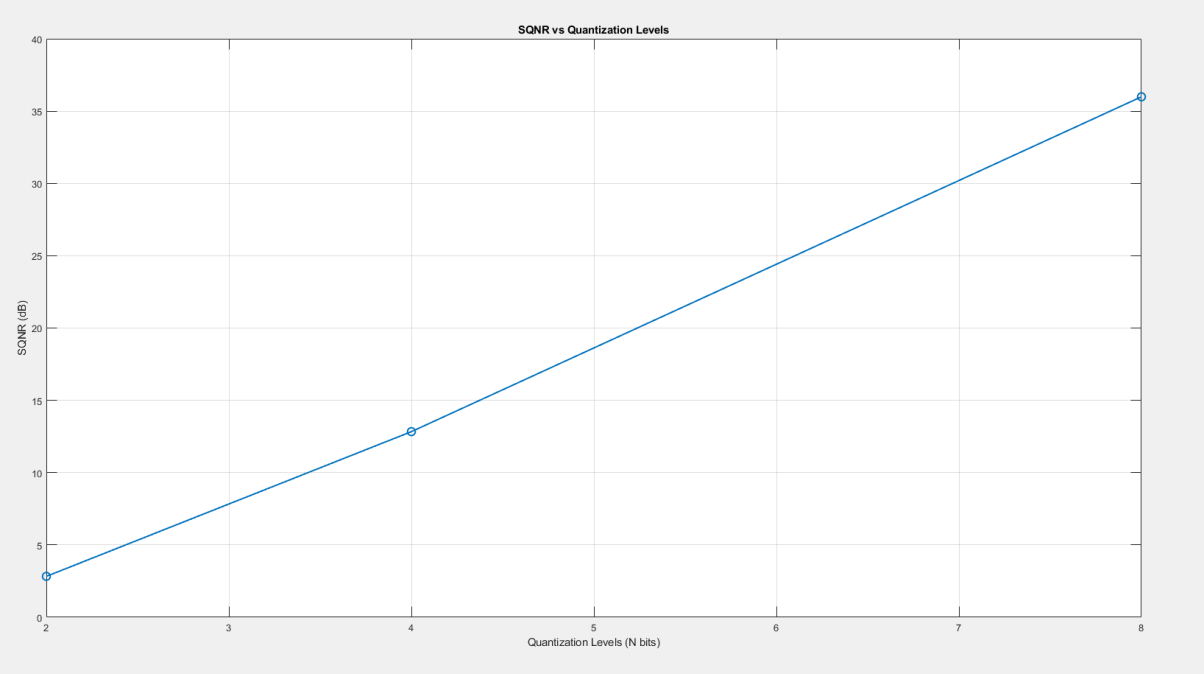
Σχήμα 7: SQNR για κάθε N (Lloyd-max algorithm)

ii.

Υλοποίηση

```
% Φόρτωση αρχείου ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y)); % Κανονικοποίηση σήματος στο διάστημα [-1, 1]

% Παράμετροι
min_value = -1;
max_value = 1;
bits = [2, 4, 8];

% Αρχικοποίηση πίνακα για αποθήκευση τιμών SQNR για κάθε N
SQNR_results = zeros(length(bits), 1);

% Επαναλήψεις για κάθε N
for b = 1:length(bits)
    N = bits(b);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq, ~] = my_quantizer(y, N, min_value, max_value);

    % Υπολογισμός SQNR
    signal_power = mean(y(:).^2);
    noise_power = mean((y(:) - xq(:)).^2);
    SQNR_results(b) = 10 * log10(signal_power / noise_power);
end

% Plot για τα διαφορετικά N
figure;
plot(bits, SQNR_results, '-o', 'LineWidth', 1.5, 'MarkerSize', 8);
xlabel('Quantization Levels (N bits)');
ylabel('SQNR (dB)');
title('SQNR vs Quantization Levels');
grid on;
```

Σε αυτό το ερώτημα, χρησιμοποιώ τον ομοιόμορφο κβαντιστή που υλοποίησα στο προηγούμενο ερώτημα για να υπολογίσω το SQNR για κάθε N (2, 4, 8). Η μόνη διαφορά είναι ότι για κάθε N καλείται η συνάρτηση ομοιόμορφου κβαντισμού και υπολογίζεται το SQNR, το οποίο αποθηκεύεται στο διάνυσμα SQNR_results. Δεν έχουμε επαναλήψεις, οπότε το SQNR αναμένουμε να είναι σταθερό σε κάθε N και όσο μεγαλύτερο N, τόσο καλύτερο SQNR (για τους λόγους που είπαμε παραπάνω).



Σχήμα 8: SQNR για κάθε N (ομοιόμορφος κβαντιστής)

iii.

Θεωρητικά

Η κβάντιση ενός ηχητικού σήματος με ομοιόμορφο κβαντιστή, δεν αναμένουμε να είναι πολύ αποδοτική, εφόσον το σήμα δεν έχει σταθερό πλάτος σε όλες του τις συχνότητες. Έτσι, θα οδηγηθούμε σε μεγαλύτερο σφάλμα κβάντισης και ο ήχος θα ακούγεται με πολύ θόρυβο. Όσον αφορά τον μη-ομοιόμορφο κβαντιστή, θα είναι μια πιο αποτελεσματική διαδικασία, διότι η κβάντιση θα γίνει βάσει της κατανομής του σήματος. Ο ήχος αναμένουμε να ακούγεται πολύ πιο καθαρά.

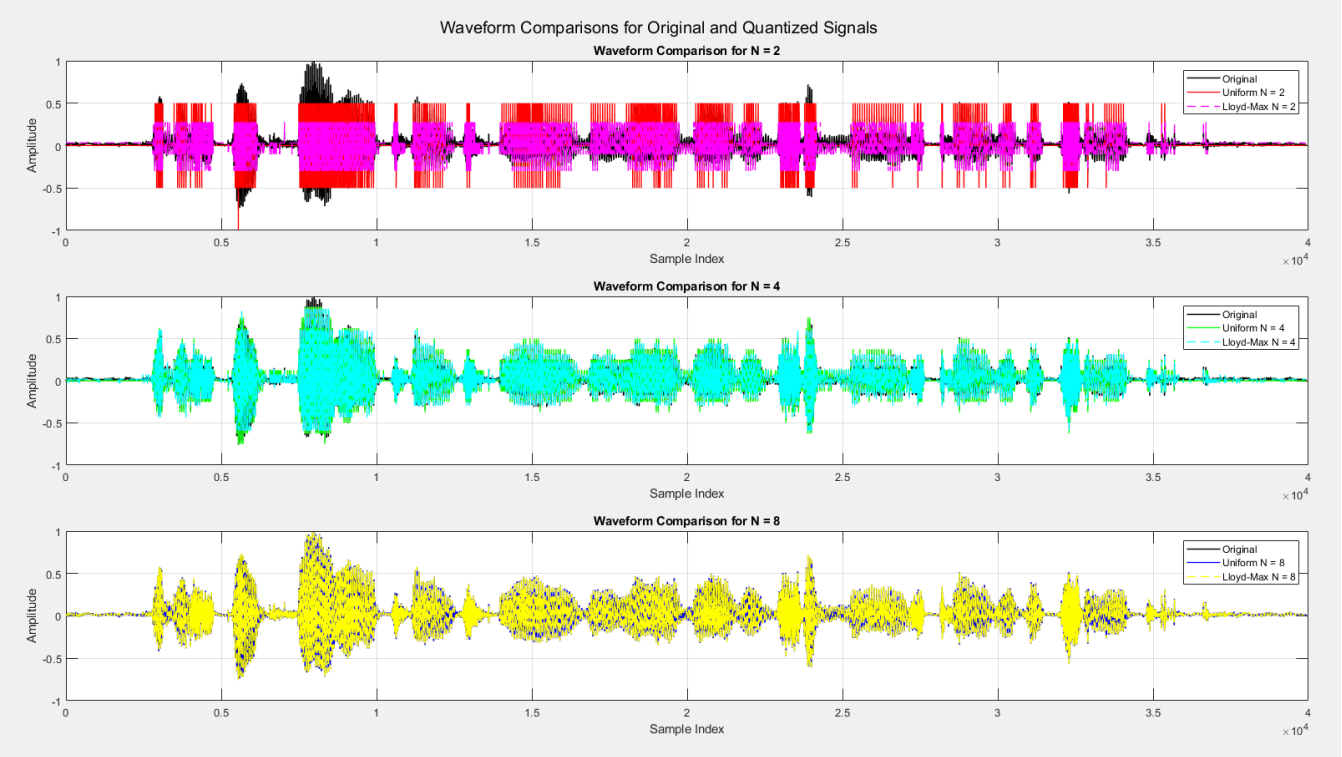
Υλοποίηση

Σε αυτό το ερώτημα, χρησιμοποίησα τον ομοιόμορφο και μη-ομοιόμορφο κβαντιστή που υλοποίησα στα παραπάνω ερωτήματα. Δεν κάνω τίποτα διαφορετικό από τα προηγούμενα ερωτήματα, απλά για κάθε N χρησιμοποιώ τους δύο κβαντιστές και απεικονίζω τα αποτελέσματα σε γραφήματα, όπως ζητείται στο ερώτημα.

Όσο αυξάνεται το N, η ποιότητα του ήχου βελτιώνεται. Αυτό συμβαίνει διότι, υπάρχουν πολλά επίπεδα, οπότε το σφάλμα κβαντισμού μειώνεται σε

μεγάλο βαθμό και ο ήχος γίνεται σχεδόν ίδιος με τον αρχικό. Σε χαμηλά N (N=2), η ποιότητα του ήχου είναι αρκετά κακή, καθώς ο ήχος αναπαράγεται πολύ παραμορφωμένος.

```
% Λούπα για τα N
for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq_uniform, ~] = my_quantizer(y, N, min_value, max_value);

    % Κλήση συνάρτησης Lloyd-Max
    [xq_lloyd, ~, ~] = Lloyd_max(y, N, min_value, max_value);

    % Αρχικός ήχος
    fprintf('Playing original signal...\n');
    sound(y, fs);
    pause(length(y) / fs + 1); % Αναμονή μέχρι να τελειώσει ο ήχος

    fprintf('Playing uniform quantized signal for N = %d...\n', N);
    sound(xq_uniform, fs);
    pause(length(xq_uniform) / fs + 1);

    fprintf('Playing Lloyd-Max quantized signal for N = %d...\n', N);
    sound(xq_lloyd, fs);
    pause(length(xq_lloyd) / fs + 1);

    % Plot κυματομορφών
    subplot(length(bits), 1, idx);
    plot(y, 'k', 'LineWidth', 1.2, 'DisplayName', 'Original');
    hold on;
    plot(xq_uniform, [colors(idx) line_styles{1}], 'LineWidth', 1, ...
         'DisplayName', sprintf('Uniform N = %d', N));
    plot(xq_lloyd, [lloyd_colors(idx) line_styles{2}], 'LineWidth', 1, ...
         'DisplayName', sprintf('Lloyd-Max N = %d', N));
    title(sprintf('Waveform Comparison for N = %d', N));
    xlabel('Sample Index');
    ylabel('Amplitude');
    legend;
    grid on;
end

hold off;
sgtitle('Waveform Comparisons for Original and Quantized Signals');
```



Σχήμα 9: Αρχικό και κωδικοποιημένο σήμα για κάθε N

iv.

Θεωρητικά

Το Μέσο Τετραγωνικό Σφάλμα (MSE) είναι μία μετρική για τον υπολογισμό της ακρίβειας ενός μοντέλου. Μετράει τη μέση τετραγωνική διαφορά μεταξύ των προβλεπόμενων τιμών και των πραγματικών στο σύνολο των δεδομένων.

Ο τύπος για τον υπολογισμό του είναι ο εξής:

$$\text{Mean Squared Error (MSE)} = n1i=1\sum n(Yi-Y\hat{}i)2$$

Τι ακριβώς σημαίνει όμως; Ένα χαμηλό MSE υποδεικνύει ότι η πρόβλεψη είναι πιο κοντά στις πραγματικές τιμές, οπότε υπάρχει μεγαλύτερη ακρίβεια. Αντιθέτως, ένα μεγάλο MSE δείχνει ότι η πρόβλεψη απέχει πολύ από τις πραγματικές τιμές.

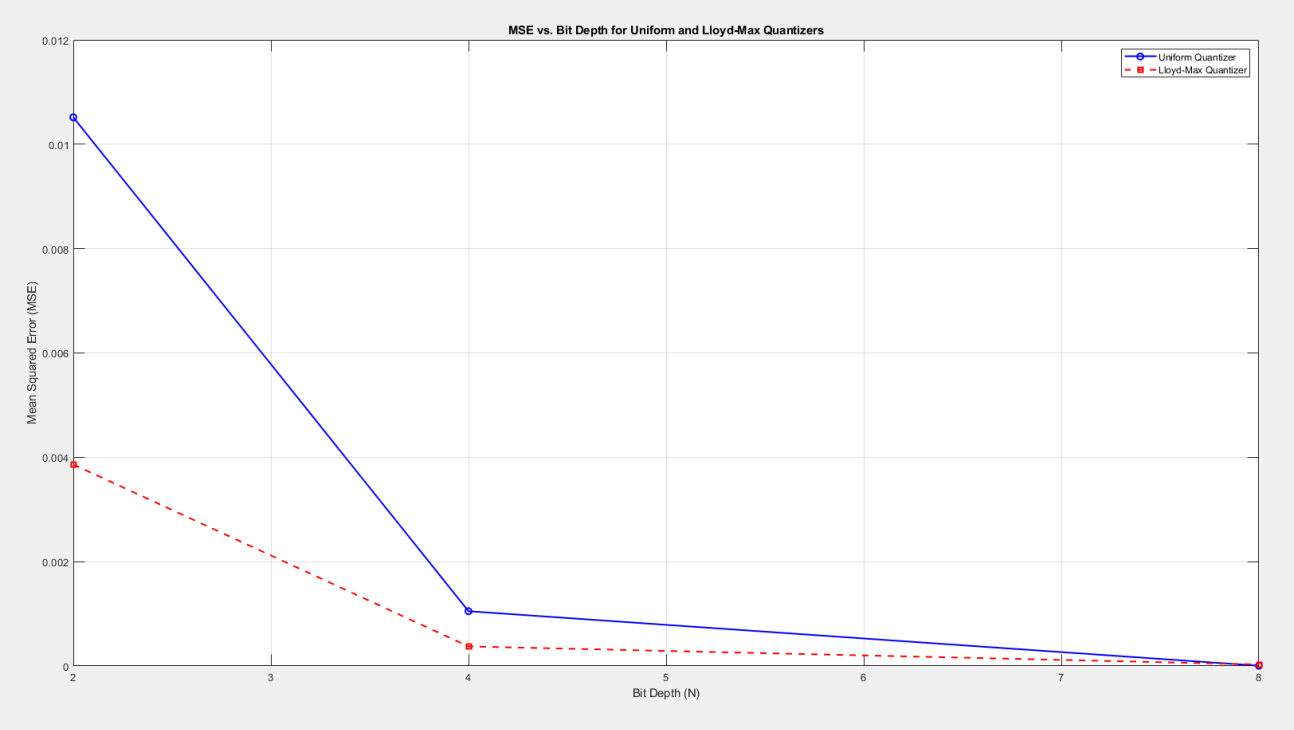
Υλοποίηση

Χρησιμοποίησα τους δύο κβαντιστές που έχω υλοποιήσει και υπολόγισα το MSE. Στη συνέχεια το απεικόνισα σε ένα γράφημα.

```
% Φόρτωση ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y));

% Παράμετροι
min_value = -1;
max_value = 1;
bits = [2, 4, 8];

% Αρχικοποίηση πινάκων MSE για κάθε συνάρτηση
mse_uniform = zeros(size(bits));
mse_lloyd = zeros(size(bits));

% Λούπα για κάθε N
for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq_uniform, ~] = my_quantizer(y, N, min_value, max_value);
    mse_uniform(idx) = mean((y - xq_uniform).^2); % Υπολογισμός MSE

    % Κλήση συνάρτησης Lloyd-Max
    [xq_lloyd, ~, ~] = Lloyd_max(y, N, min_value, max_value);
    mse_lloyd(idx) = mean((y - xq_lloyd).^2); % Υπολογισμός MSE
end

% Plot MSE
figure;
plot(bits, mse_uniform, '-o', 'Color', 'b', 'LineWidth', 1.5, ...
     'DisplayName', 'Uniform Quantizer');
hold on;
plot(bits, mse_lloyd, '--s', 'Color', 'r', 'LineWidth', 1.5, ...
     'DisplayName', 'Lloyd-Max Quantizer');

xlabel('Bit Depth (N)');
ylabel('Mean Squared Error (MSE)');
title('MSE vs. Bit Depth for Uniform and Lloyd-Max Quantizers');
legend('Location', 'northeast');
grid on;
hold off;
```

Όπως παρατηρούμε, το MSE μειώνεται όσο αυξάνεται το N. Αυτό δείχνει ότι όσο περισσότερα είναι τα bit depths, τόσο μικρότερο σφάλμα κβαντισμού υπάρχει. Επίσης, με τον μη-ομοιόμορφο κβαντιστή, το MSE είναι πάντα μικρότερο, διότι εκμεταλλεύεται καλύτερα τα intervals. Η διαφορά πάντως φαίνεται στο μικρό N (2), όπου ο αλγόριθμος Lloyd-max

είναι κατά πολύ πιο αποδοτικός. Όσο όμως το N μεγαλώνει, η διαφορά είναι αμελητέα. Αυτό συμβαίνει, διότι υπάρχουν αρκετά επίπεδα, οπότε επιτρέπει και στους δύο κβαντιστές να κβαντίσουν το σήμα πολύ αποδοτικά, με ελάχιστο σφάλμα.



Σχήμα 10: MSE για ομοιόμορφο και μη-ομοιόμορφο κβαντιστή

Ερωτήματα - Μέρους Α2

1.

Θεωρητικά

Μέχρι στιγμής, μιλήσαμε για το PCM. Τώρα θα δούμε τη μέθοδο του DPCM. Η κύρια διαφορά είναι ότι η DPCM κβαντίζει ένα συγκεκριμένο δείγμα και την αναμενόμενη τιμή, γι' αυτό το λόγο χαρακτηρίζεται ως διαφορική PCM.

Πως λειτουργεί όμως; Το DPCM ξεκινά με την εκτίμηση του τρέχοντος δείγματος από το προηγούμενο δείγμα της σειράς. Στη συνέχεια προσδιορίζει τη διαφορά ή το σφάλμα μεταξύ του τρέχοντος δείγματος και της προβλεπόμενης τιμής. Αυτή η διαφορά είναι που μεταδίδεται και κωδικοποιείται. Η διαδικασία αποκωδικοποίησης περιλαμβάνει την

ανακατασκευή του αρχικού σήματος μέσω μιας τεχνικής πρόσθεσης της μεταδιδόμενης διαφοράς στο τελευταίο δείγμα που επιτεύχθηκε.

Υλοποίηση

Χρησιμοποίησα τον ομοιόμορφο κβαντιστή που υλοποίησα στο προηγούμενο μέρος της εργαστηριακής άσκησης, με μία μικρή αλλαγή.

```
function yq = my_quantizer(y, N, min_value, max_value)

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be less than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    y_confined = max(min(y, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12;
    yq_index = round((y_confined - min_value + epsilon) / delta);
    yq_index = max(min(yq_index, L-1), 0); % Clamp to valid index range

    % Βήμα 4: Χαρτογράφηση του δείκτη στην αντίστοιχη κβαντισμένη τιμή
    yq = centers(yq_index + 1);

end
```

Εκτός συνάρτησης, αρχικοποιώ τις παραμέτρους. Η μόνη καινούρια μεταβλητή εδώ είναι το p, όπου συμβολίζει τον αριθμό των δειγμάτων για πρόβλεψη. Το N, min_value και max_value τα έχουμε ξανά δει. Στη συνέχεια φορτώνουμε τα δείγματα.

```
% Αρχικοποίηση Παραμέτρων
p = 8;
N = 3;
min_value = -3.5;
max_value = 3.5;

% Φόρτωση δειγμάτων
data = load('source.mat');
variables = fieldnames(data);
x = data.(variables{1});

x_length = length(x);
```

Στο βήμα 1, υπολογίζουμε τον πίνακα αυτοσυσχέτισης R και το διάνυσμα αυτοσυσχέτισης r, όπως υποδεικνύονται στην εργαστηριακή άσκηση, οι οποίοι χρησιμοποιούνται για τις εξισώσεις Yule-Walker για την εύρεση των συντελεστών πρόβλεψης.

Για τον πίνακα αυτοσυσχέτισης, υπολογίζονται οι συσχετίσεις ανά ζεύγη μεταξύ τιμών σήματος σε διαφορετικές χρονικές υστερήσεις.

Για το διάνυσμα αυτοσυσχέτισης, για κάθε υστέρηση i, υπολογίζει το μέσο γινόμενο του σήματος και της καθυστερημένης εκδοχής του.

```
% Βήμα 1: Υπολογισμός πίνακα αυτοσυσχέτισης R και διάνυσμα αυτοσυσχέτισης r
R = zeros(p, p);
r = zeros(p, 1);

for i = 1:p
    sum_r = 0;
    for n = p+1:x_length
       sum_r = sum_r + x(n) * x(n-i);
    end
    r(i) = sum_r / (x_length - p);

    for j = 1:p
        sum_R = 0;
        for n = p+1:x_length
           sum_R = sum_R + x(n-j) * x(n-i);
        end
        R(i, j) = sum_R / (x_length - p);
    end
end
```

Στο βήμα 2, έχουμε την ισότητα Yule – Walker, η οποία επιλύεται για τον υπολογισμό των συντελεστών πρόβλεψης a. Οι συντελεστές μετά κβαντίζονται χρησιμοποιώντας τη συνάρτηση.

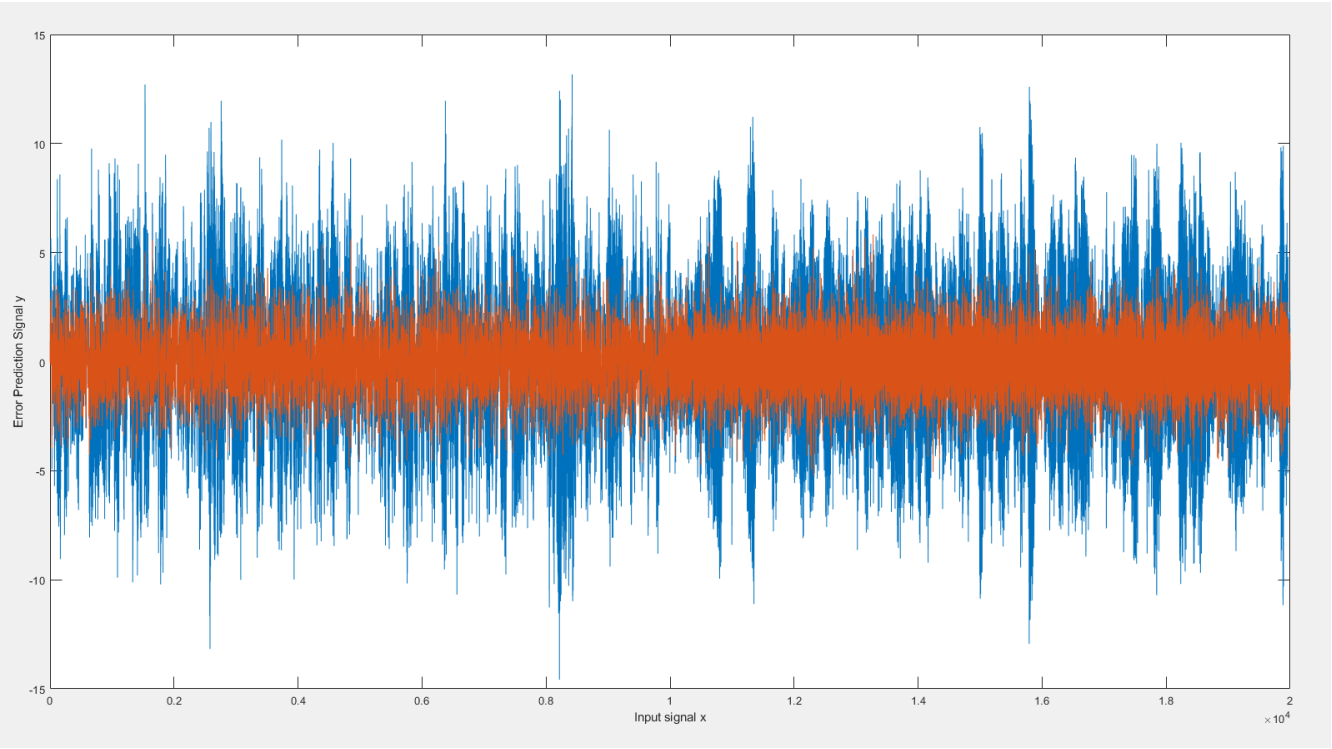
```
% Βήμα 2: Υπολογισμός συντελεστπών και κβαντισμός τους
a = R \ r; % Yule-Walker
for i = 1:p
    a(i) = my_quantizer(a(i), 8, -2, 2);
end
```

Στο βήμα 3 έχομε τη κωδικοποίηση DPCM. Αυτό το κομμάτι προβλέπει το σήμα, υπολογίζει το σφάλμα πρόβλεψης και κβαντίζει το σφάλμα.

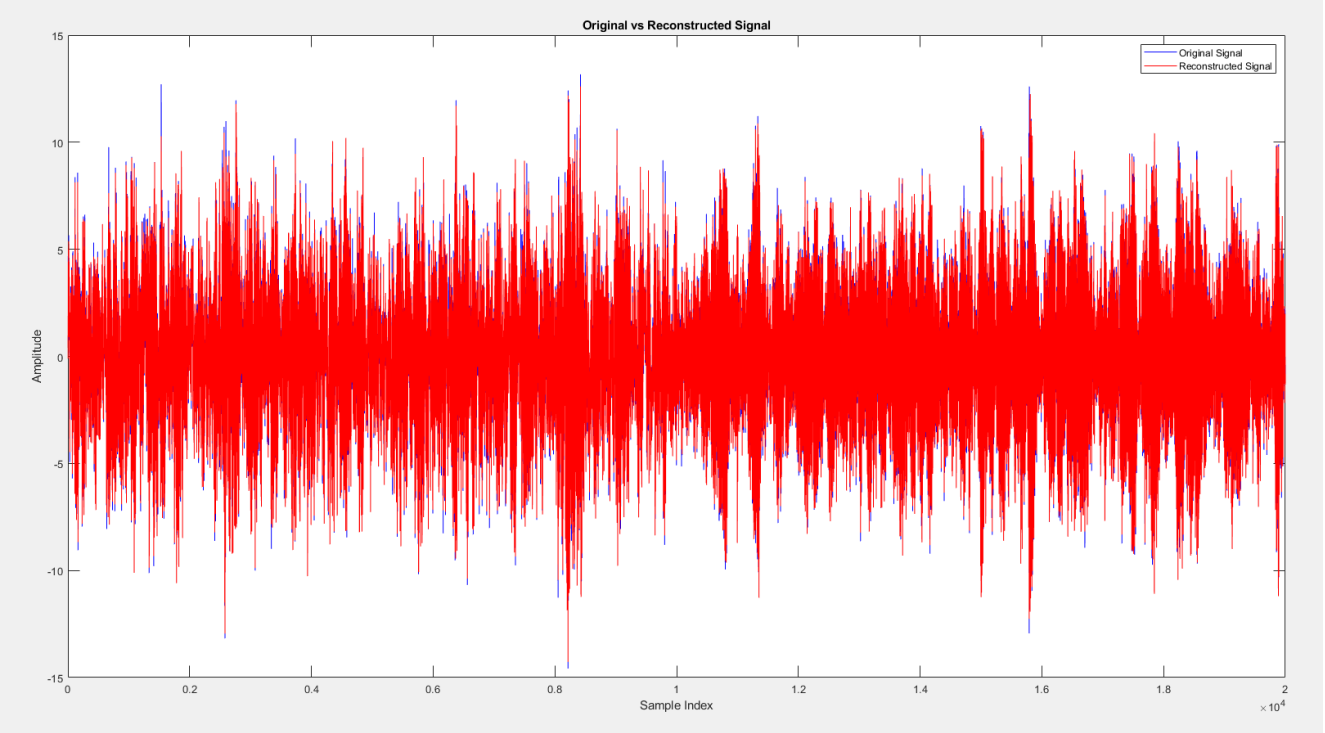
```
% Βήμα 3: DPCM Κωδικοποίηση
memory = zeros(p, 1);
y = zeros(x_length, 1);
y_hat = zeros(x_length, 1);
y_toned = 0;

for i = 1:x_length
    y(i) = x(i) - y_toned; % Πρόβλεψη σφάλματος
    y_hat(i) = my_quantizer(y(i), N, min_value, max_value); % Κβαντισμός error
    y_hat_toned = y_hat(i) + y_toned; % Προσθήκη σφάλμα κβαντισμού στη πρόβλεψη
    memory = [y_hat_toned; memory(1:p-1)];
    y_toned = a' * memory;
end
```

Στο βήμα 4 έχουμε την αποκωδικοποίηση DPCM. Εδώ ανακατασκευάζεται το αρχικό σήμα από το κβαντισμένο σφάλμα.

```
% Βήμα 4: DPCM Αποκωδικοποίηση
memory = zeros(p, 1);
y_toned = 0;
y_final = zeros(x_length, 1);

for i = 1:x_length
    y_final(i) = y_hat(i) + y_toned; % Ανακατασκευή
    memory = [y_final(i); memory(1:p-1)];
    y_toned = a' * memory;
end
```



Σχήμα 11: Αρχικό σήμα (μπλε) και πρόβλεψη σφάλματος (πορτοκαλί)



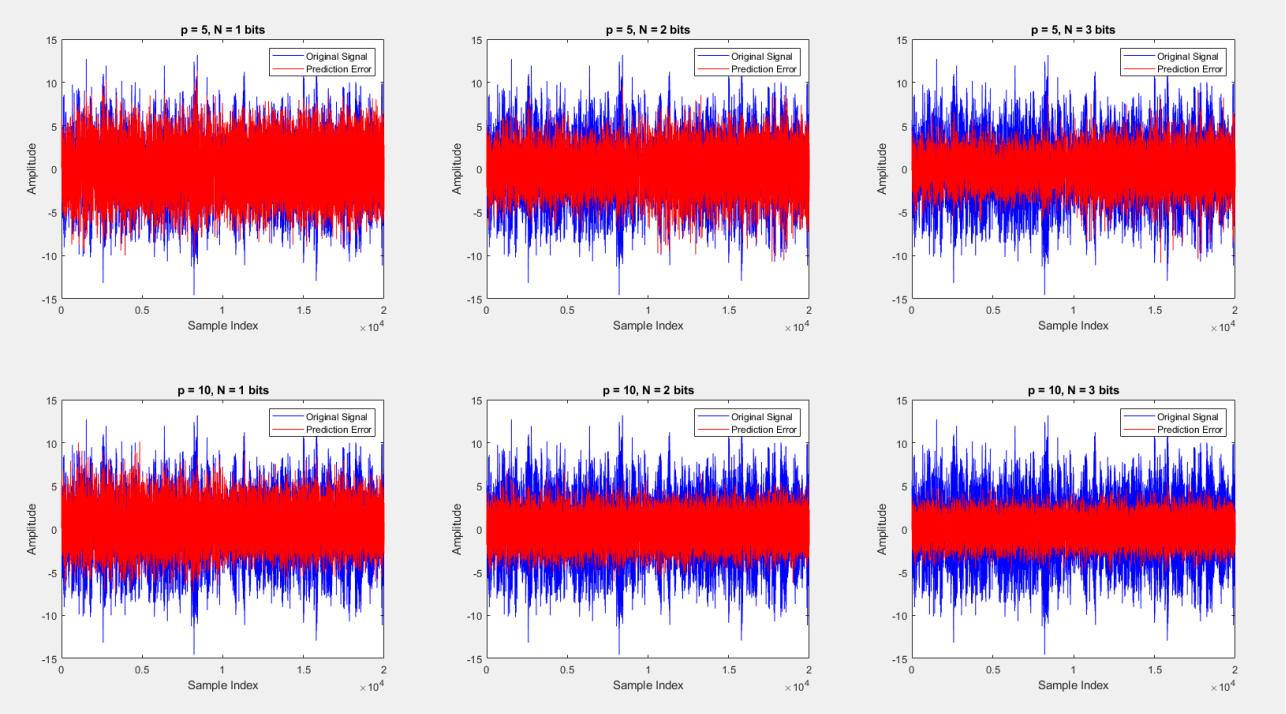
Σχήμα 12: Αρχικό σήμα (μπλε) και ανακατασκευασμένο (κόκκινο)

2.

Υλοποίηση

Για το ερώτημα αυτό, επέλεξα τις τιμές p=5 και p=10. Ο κώδικας είναι ίδιος, με τη μόνη διαφορά ότι πλέον έχουμε 2 τιμές p και 3 τιμές N. Οι λούπες επαναλαμβάνονται για κάθε p και N, οπότε θα έχουμε 6 γραφήματα σύνολο.

```
% Λούπα για όλα τα p και N
figure;
for p_idx = 1:length(p_values)
    p = p_values(p_idx);

    R = zeros(p, p);
    r = zeros(p, 1);
    for i = 1:p
        sum_r = 0;
        for n = p+1:x_length
            sum_r = sum_r + x(n) * x(n-i);
        end
        r(i) = sum_r / (x_length - p);

        for j = 1:p
            sum_R = 0;
            for n = p+1:x_length
                sum_R = sum_R + x(n-j) * x(n-i);
            end
            R(i, j) = sum_R / (x_length - p);
        end
    end

    a = R \ r;
    for i = 1:p
        a(i) = my_quantizer(a(i), 8, -2, 2);
    end

    for N_idx = 1:length(N_values)
        N = N_values(N_idx);

        % Αρχικοποίηση παραμέτρων για DPCM
        memory = zeros(p, 1);
        y = zeros(x_length, 1);
        y_hat = zeros(x_length, 1);
        y_toned = 0;

        for i = 1:x_length
            y(i) = x(i) - y_toned;
            y_hat(i) = my_quantizer(y(i), N, min_value, max_value);
            y_hat_toned = y_hat(i) + y_toned;
            memory = [y_hat_toned; memory(1:p-1)];
            y_toned = a' * memory;
        end

        % Plot αρχικού σήματος και πρόβλεψης σφάλματος
        subplot(length(p_values), length(N_values), (p_idx-1)*length(N_values) + N_idx);
        plot(1:x_length, x, 'b', 1:x_length, y, 'r');
        title(sprintf('p = %d, N = %d bits', p, N));
        legend('Original Signal', 'Prediction Error');
        xlabel('Sample Index');
        ylabel('Amplitude');
    end
end
```



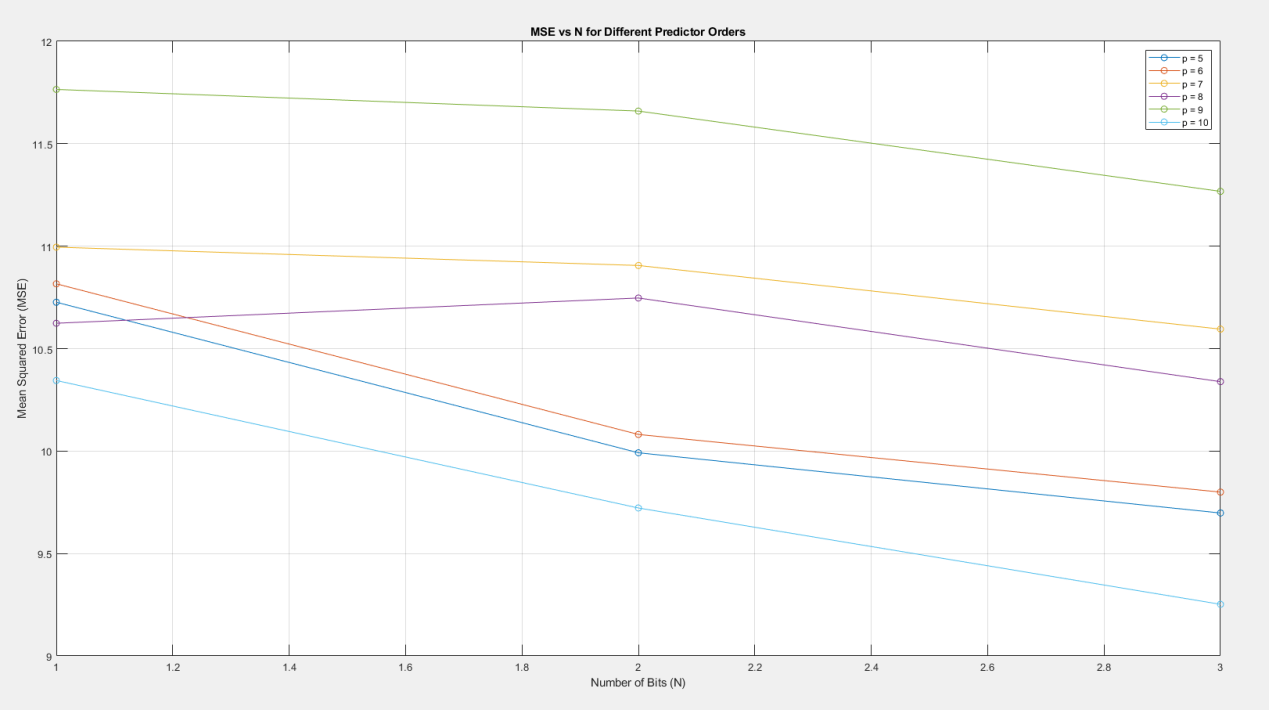
Σχήμα 13: Αρχικό σήμα και σφάλμα πρόβλεψης για p=5/10 και N=1/2/3

Όπως παρατηρούμε στο παραπάνω σχήμα, όσο το N αυξάνεται (πάνω σειρά), τόσο περισσότερο μειώνεται το σφάλμα πρόβλεψης (κόκκινο), υποδηλώνοντας καλύτερη ακρίβεια κβάντισης. Αυτό γιατί θα έχουμε περισσότερες περιοχές κβάντισης και το σήμα θα αντιπροσωπεύεται καλύτερα με χαμηλότερο σφάλμα κβάντισης.

Όσον αφορά το p, καθώς το αυξήσαμε στο 10, υπάρχει μικρή βελτίωση. Αυτό δείχνει ότι αν αυξήσουμε το p, θα βοηθήσει το μοντέλο να κάνει καλύτερες προβλέψεις. Αυτό συμβαίνει, διότι περισσότερη πληροφορία θα αποθηκευτεί στη μνήμη, οπότε θα πραγματοποιηθεί πιο αποδοτική πρόβλεψη.

3.

Υλοποίηση



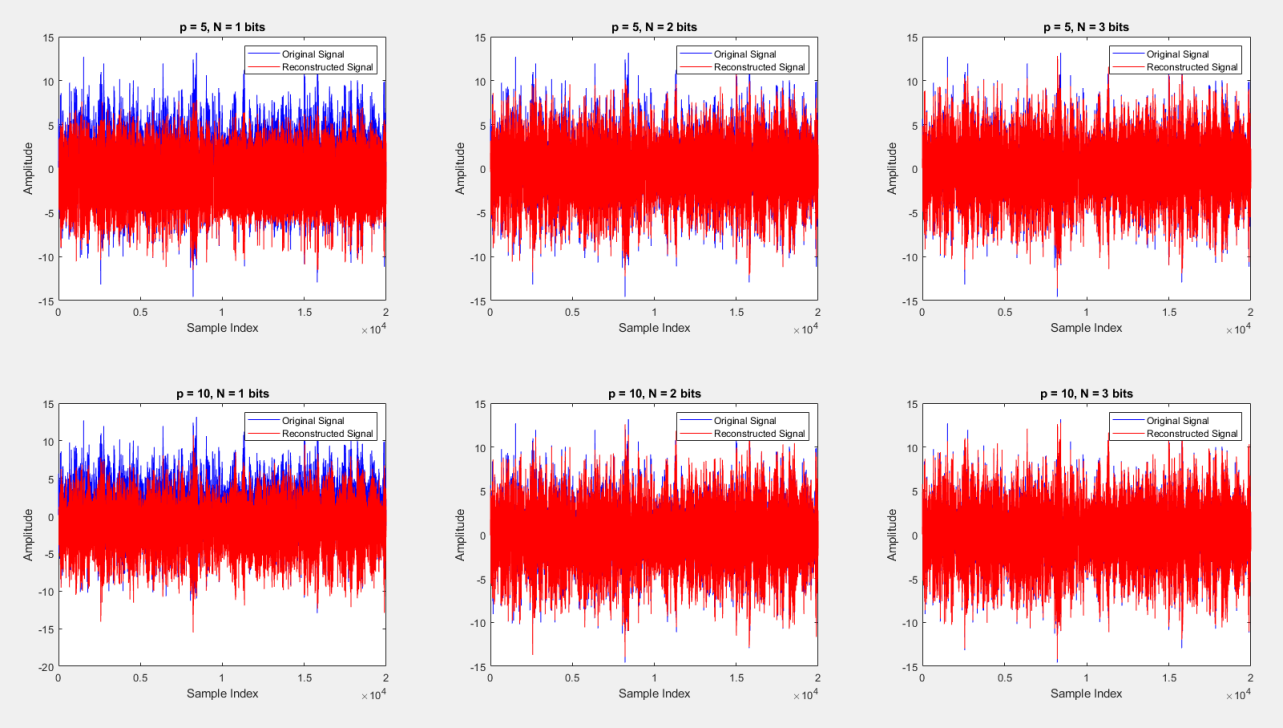
Σχήμα 14: MSE για κάθε N (1/2/3) και για κάθε p (5:10)

Όπως παρατηρούμε στο σχήμα, όσο αυξάνεται το N, το MSE μειώνεται, διότι υπάρχει μεγαλύτερη ακρίβεια στη κβάντιση, οπότε και μικρότερο σφάλμα. Τα μοντέλα με μεγαλύτερο p, συνήθως ξεκινάνε με χαμηλότερο MSE και συγκλίνουν περίπου στο ίδιο N.

Στην αρχή, σε χαμηλά N, παρατηρούμε πως όσο μεγαλύτερο είναι το p, τόσο καλύτερα αποδίδει το μοντέλο, αλλά όσο αυξάνεται το N, οι διαφορά στο MSE είναι μικρή.

4.

Υλοποίηση



Σχήμα 15: Αρχικό και ανακατασκευασμένο σήμα για N (1/2/3) και p (5/10)

Στο παραπάνω σχήμα, βλέπουμε το αρχικό σήμα (μπλε) και το ανακατασκευασμένο (κόκκινο) για N=1,2,3 και p=5,10.

Παρατηρούμε πως όσο αυξάνεται το N, τόσο μειώνεται το σφάλμα κβάντισης και τόσο καλύτερα αναπαρίσταται το αρχικό σήμα. Σε χαμηλά N, βλέπουμε ότι υπάρχει αρκετός θόρυβος, κάνοντας την ανακατασκευή lossy.

Όσον αφορά το p, παρατηρούμε μικρή βελτίωση καθώς γίνεται μεγαλύτερο.

Συμπεραίνοντας, μια ισορροπία πρέπει να βρεθεί στην αντιστοιχία του N και του p, για να έχουμε τα καλύτερα δυνατά αποτελέσματα, αλλά και για λόγους απλοποίησης.

ΜΕΡΟΣ Β

Ερωτήματα – Μέρος Β

1.

Θεωρητικά

Η διαμόρφωση παλμικού πλάτους (PAM) είναι μια βασική τεχνική διαμόρφωσης που χρησιμοποιείται στις ψηφιακές επικοινωνίες για τη μετάδοση αναλογικών δεδομένων και είναι ένας από τους πιο διαδεδομένους τύπους αναλογικής-ψηφιακής μετατροπής. Η διαδικασία της είναι απλή, όπου το πλάτος μιας ακολουθίας παλμών αλλάζει με το στιγμιαίο πλάτος του αναλογικού σήματος μηνύματος. Το αναλογικό σήμα που πρόκειται να διαμορφωθεί δειγματοληπτείται από μια ακολουθία παλμών που διαμορφώνεται κατά πλάτος στο φέρον για να παραχθούν οι διαμορφωμένοι κατά πλάτος παλμοί.

Το αναλογικό σήμα δειγματοληπτείται σε τακτά χρονικά διαστήματα, ώστε να είναι δυνατή η μεταβολή του πλάτους των παλμών που πρέπει να παραχθούν από τον φορέα. Οι δειγματοληπτούμενες τιμές κβαντίζονται σε συγκεκριμένο αριθμό επιπέδων κβαντισμού ή διακριτών επιπέδων, οπότε η διαδικασία επαναλαμβάνεται.

Ένα σύστημα PAM λειτουργεί με την αντιστοίχιση ομάδων bits σε διακριτά επίπεδα πλάτους, τη διαμόρφωσή τους σε ένα φέρον και τη μετάδοση του διαμορφωμένου σήματος μέσω ενός καναλιού προσθετικού λευκού γκαουσιανού θορύβου (AWGN). Ο δέκτης αποδιαμορφώνει και ανιχνεύει τα μεταδιδόμενα σύμβολα, υπολογίζει τα σφάλματα και αξιολογεί την απόδοση του συστήματος.

Υλοποίηση

Για την υλοποίηση αυτού του ερωτήματος, ακολούθησα τα βήματα της εργαστηριακής άσκησης. Συγκεκριμένα:

- Ορισμός όλων των απαραίτητων παραμέτρων που ζητούνται

```
%% Παράμετροι
M = 8; % Τάξη M-PAM (μπορεί επίσης να δοκιμαστεί με M=2)
log2M = log2(M); % Αριθμός bits ανά σύμβολο
Lb = 100000; % Συνολικός αριθμός bits
Lb = floor(Lb / log2M) * log2M; % Προσαρμογή ώστε να είναι διαιρετός με log2(M)
Es = 1; % Ενέργεια ανά σύμβολοl
Rs = 250e3; % Ρυθμός συμβόλων (250 Ksymbols/sec)
Ts = 1 / Rs; % Διάρκεια συμβόλου
fc = 2.5e6; % Συχνότητα φέρουσας (2.5 MHz)
Tc = 1 / fc; % Περίοδος φέρουσας
Tsamp = Tc / 4; % Περίοδος δειγματοληψίας (4 δείγματα ανά περίοδο φέρουσας)
Ns = round(Ts / Tsamp); % Αριθμός δειγμάτων ανά σύμβολο
SNR_dB_values = 0:2:20; % Εύρος SNR (dB)
```

- Δημιουργία μιας τυχαίας δυαδικής ακολουθίας

```
%% Δημιουργία Δυαδικής Ακολουθίας Εισόδου
bits = randi([0 1], 1, Lb); % Τυχαία δυαδική ακολουθία
```

- Μετατροπή της σε σύμβολα

```
%% Mapping: Bits σε σύμβολα
symbols = reshape(bits, log2M, []); % Ομαδοποίηση bits σε log2(M) ανά σύμβολο
decSymbols = bi2de(symbols.', 'left-msb'); % Μετατροπή σε δεκαδικά σύμβολα
Am = (2 * (0:M-1) - (M - 1)) * sqrt(Es / (2 * (M - 1)^2)); % Επίπεδα πλάτους PAM
```

- Πολλαπλασιασμός της με έναν ορθογώνιο παλμό

```
%% Διαμόρφωση παλμού
gT = ones(1, Ns); % Ορθογώνιος Παλμός
gT = gT / sqrt(sum(gT.^2)); % Κανονικοποίηση στη μονάδα ενέργειας

s_t = reshape(repmat(Am(decSymbols + 1), Ns, 1), 1, []); % Σήμα PAM με διαμόρφωση παλμού
```

- Διαμόρφωση του σήματος

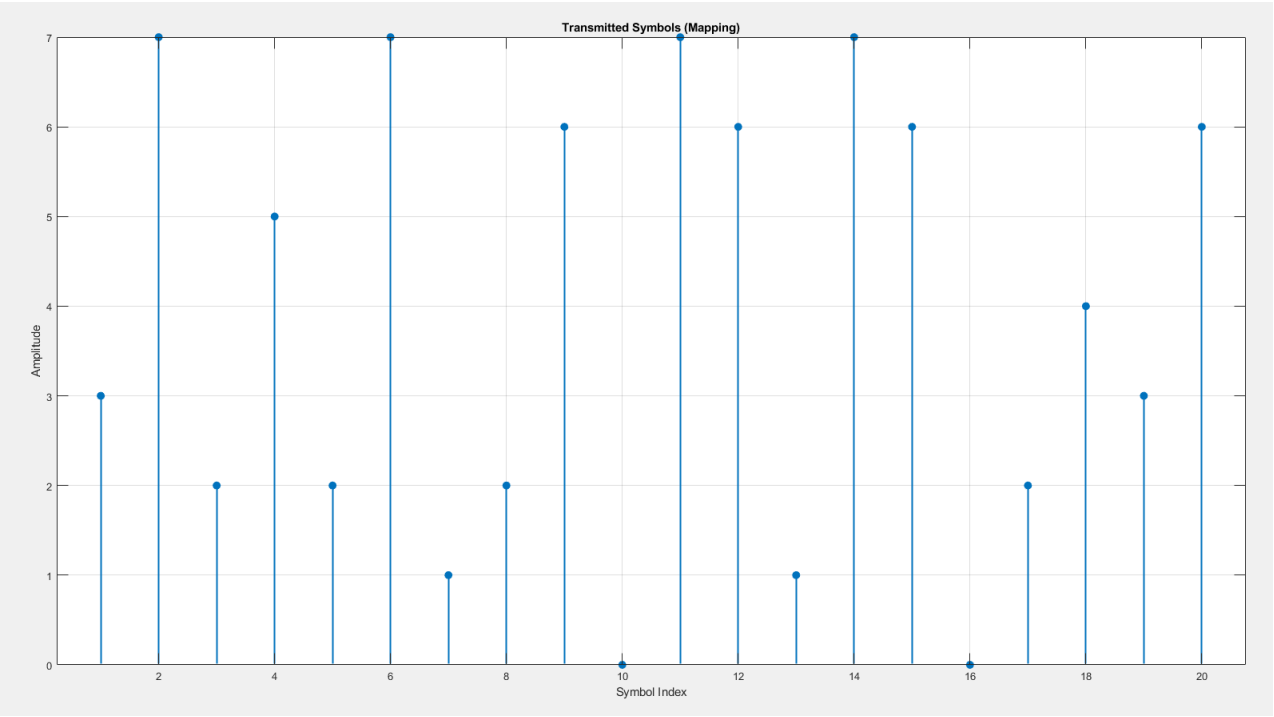
```
%% Διαμόρφωση Σήματος
num_samples = length(s_t); % Συνολικός αριθμός δειγμάτων στο s_tt
t_carrier = linspace(0, (num_samples-1)*Tsamp, num_samples); % Χρονικό διάνυσμα για τη φέρουσα
carrier = cos(2 * pi * fc * t_carrier); % Κυματομορφή φέρουσας
transmitted_signal = s_t .* carrier; % Διαμορφωμένο σήμα
```

Στη συνέχεια, το σήμα περνάει μέσα από ένα AWGN κανάλι και εν τέλει πραγματοποιείται η αποδιαμόρφωση και εντοπίζονται τα σύμβολα στον φωρατή.

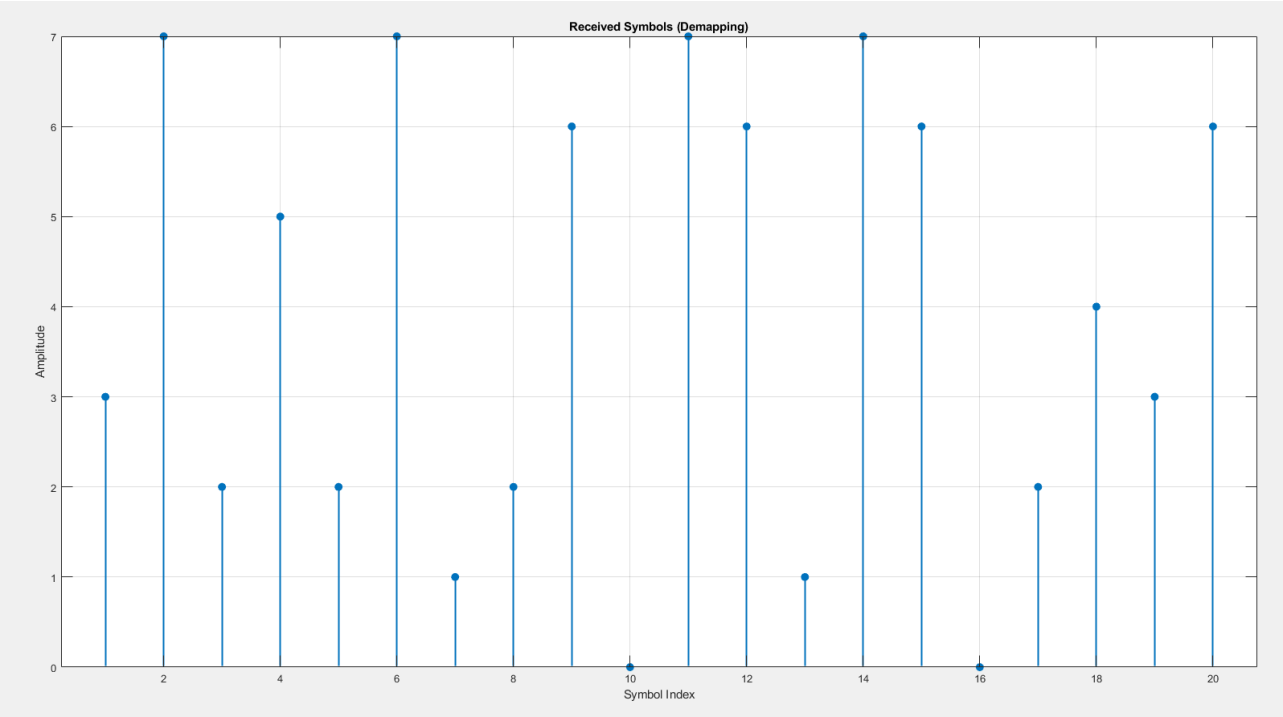
```
%% Οπτικοποίηση Μεταδιδόμενων Συμβόλων (Αντιστοίχιση)
figure;
stem(1:20, decSymbols(1:20), 'filled', 'LineWidth', 1.5);
title('Transmitted Symbols (Mapping)');
xlabel('Symbol Index');
ylabel('Amplitude');
grid on;
```

Τέλος, υπολογίζονται το BER και SER.

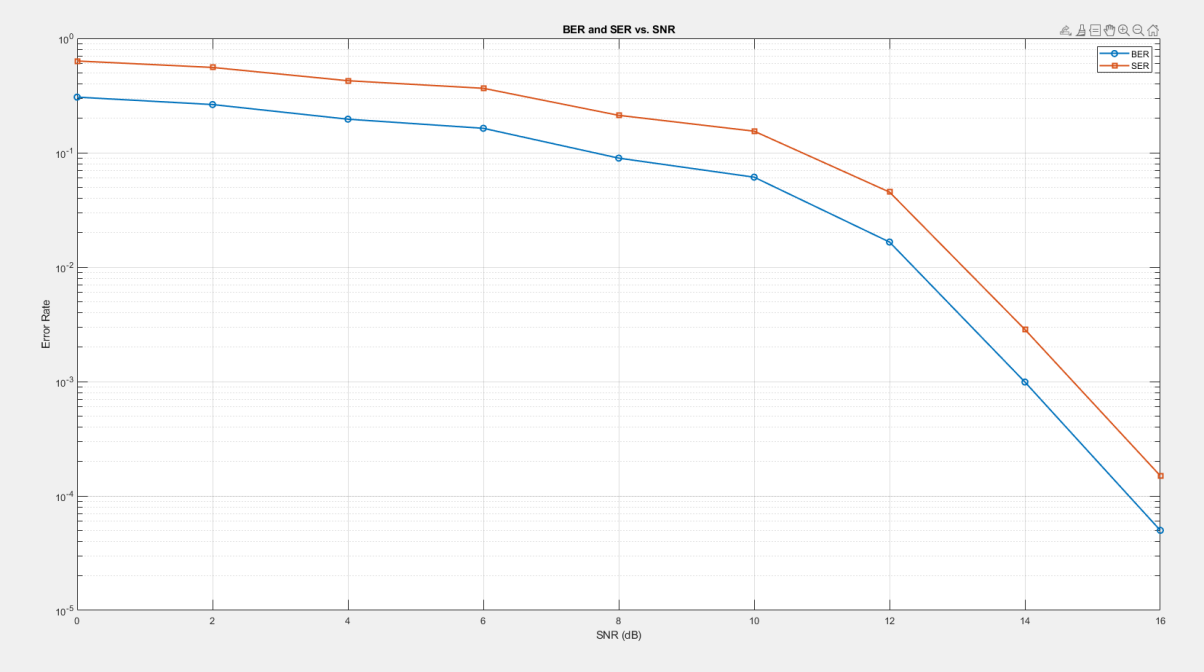
```
for idx = 1:length(SNR_dB_values)
    SNR_dB = SNR_dB_values(idx);
    SNR_linear = 10^(SNR_dB / 10);

    % Προσθήκη AWGN
    No = Es / (SNR_linear * log2M); % Ισχύς φασματικής πυκνότητας θορύβου
    sigma = sqrt(No / 2); % Τυπική απόκλιση θορύβου
    noise = sigma * randn(size(transmitted_signal));
    received_signal = transmitted_signal + noise;

    % Αποδιαμόρφωση
    demodulated_signal = received_signal .* carrier; % Πολλαπλασιασμός με φέρουσα

    % Ολοκλήρωση
    integrated_signal = zeros(1, length(decSymbols)); % Αρχικοποίηση ολοκληρωμένου σήματος
    for i = 1:length(decSymbols)
        start_idx = (i-1)*Ns + 1;
        end_idx = i*Ns;
        integrated_signal(i) = sum(demodulated_signal(start_idx:end_idx) .* gT);
    end

    % Κλιμάκωση ολοκληρωμένου σήματος
    integrated_signal = integrated_signal / max(abs(integrated_signal)) * max(abs(Am));

    % Ανίχνευση Συμβόλων
    detected_symbols = zeros(size(integrated_signal));
    for i = 1:length(integrated_signal)
        [~, detected_symbols(i)] = min(abs(integrated_signal(i) - Am)); % Ανίχνευση πλησιέστερου πλάτους
    end
    detected_symbols = detected_symbols - 1; % Προσαρμογή στο εύρος [0, M-1]

    % Οπτικοποίηση Ληφθέντων Συμβόλων (Αντιστοίχιση) για SNR = 20 dB
    if SNR_dB == 20
        figure;
        stem(1:20, detected_symbols(1:20), 'filled', 'LineWidth', 1.5);
        title('Ληφθέντα Σύμβολα (Αντιστοίχιση)');
        xlabel('Δείκτης Συμβόλου');
        ylabel('Πλάτος');
        grid on;
    end

    % Αντιστοίχιση: Σύμβολα σε Bits
    detected_bits = de2bi(detected_symbols, log2M, 'left-msb')'; % Μετατροπή συμβόλων σε bits
    detected_bits = detected_bits(:)'; % Επίπεδη αναπαράσταση σε 1D πίνακα

    % Υπολογισμός BER
    if ~isempty(detected_bits)
        minLength = min(length(bits), length(detected_bits)); % Ευθυγράμμιση μηκών
        BER_values(idx) = sum(bits(1:minLength) ~= detected_bits(1:minLength)) / Lb; % Κανονικοποίηση με τον συνολικό αριθμό bits
    else
        BER_values(idx) = NaN; % Ανάθεση NaN αν η ανίχνευση αποτύχει
        fprintf('Η ανίχνευση απέτυχε για SNR = %d dB\n', SNR_dB);
    end

    % Υπολογισμός SER
    decSymbols = decSymbols(:)'; % Διασφάλιση ότι είναι διάνυσμα γραμμής
    detected_symbols = detected_symbols(:)'; % Διασφάλιση ότι είναι διάνυσμα γραμμής
    num_symbol_errors = sum(detected_symbols ~= decSymbols); % Μέτρηση λαθών συμβόλων
    SER_values(idx) = num_symbol_errors / length(decSymbols); % Αναλογία λανθασμένων συμβόλων
end
```



Σχήμα 16: Αντιστοίχιση bits σε σύμβολα



Σχήμα 17: Αντιστοίχιση συμβόλων σε bits



Σχήμα 18: Μετρήσεις SER – BER

2.

Θεωρητικά

Σε αυτό το ερώτημα, ζητείται να υλοποιηθεί κώδικας για τη μέτρηση πιθανότητας σφάλματος και να σχεδιαστεί η καμπύλη BER.

Τι είναι όμως το BER? Είναι μια μετρική που δείχνει πόσο συχνά ένα πακέτο ή άλλα δεδομένα πρέπει να ξανά μεταδοθούν λόγω κάποιου σφάλματος. Στις ψηφιακές τηλεπικοινωνίες, ο αριθμός των bit errors είναι ο αριθμός των ληφθέντων bits που δε μεταδόθηκαν σωστά (έχουν αλλάξει), λόγω κάποιου θορύβου, παρεμβολή, παραμόρφωση και άλλα.

Το BER είναι ο λόγος των λανθασμένων bits προς των συνολικό αριθμό bits που μεταδίδονται σε ένα χρονικό διάστημα. Για παράδειγμα:

110001011 (μεταδίδονται)

010101001 (λαμβάνονται)

Έχουμε 3 λανθασμένα bits. Άρα, ο λόγος 3/9, σημαίνει ότι έχουμε BER = 33.3%.

Γιατί χρησιμοποιούμε BER? Στα ψηφιακή τηλεπικοινωνιακά συστήματα, η αξιολόγηση της απόδοσης και της αξιοπιστίας είναι πολύ σημαντική. Το BER

μετράει την απόδοση στο πιο σημαντικό επίπεδο, στα bits. Δείχνει πόσο καλά τος σύστημα διατηρεί αναλλοίωτη τη δυαδική πληροφορία που μεταδίδεται μέσα από ένα κανάλι. Επίσης, αξιολογεί την ακρίβεια του συστήματος, συνυπολογίζοντας και τα αρνητικά, όπως θόρυβος, παρεμβολή και άλλα.

Υλοποίηση

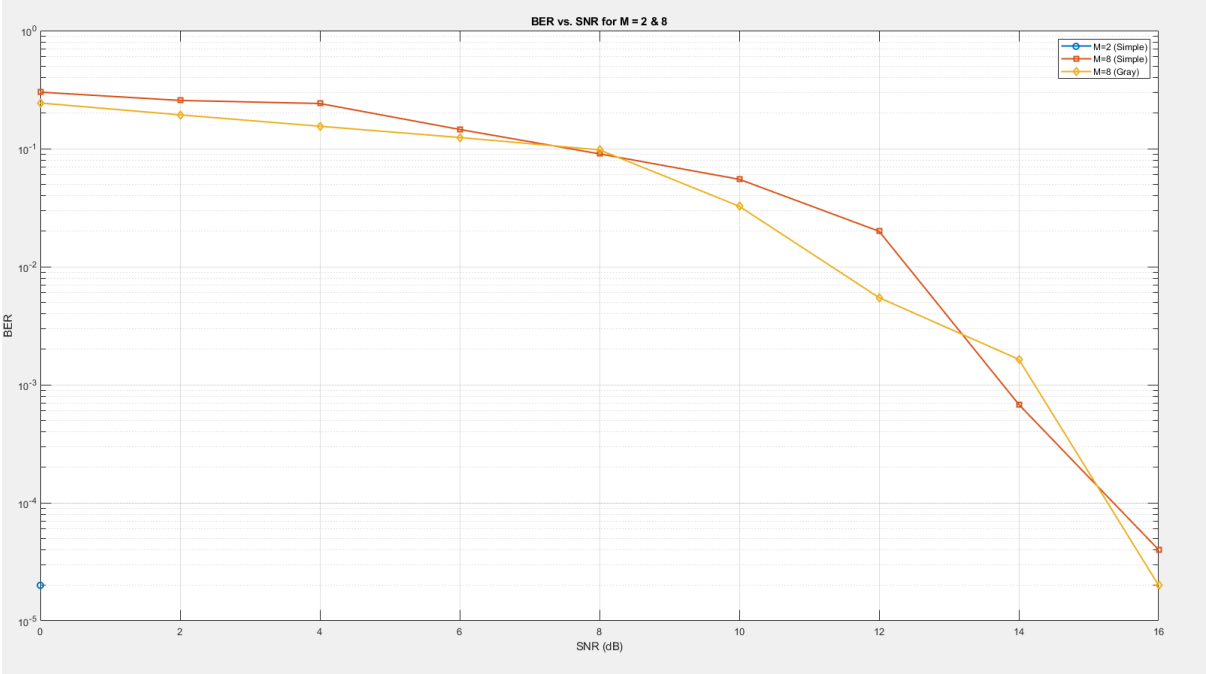
Για την υλοποίηση του ερωτήματος, ξεκίνησα με την αρχικοποίηση των απαραίτητων παραμέτρων.

```
%% Παράμετροι
M_values = [2, 8]; % Επίπεδα M-PAM
log2M_values = log2(M_values); % Αριθμός bits ανά σύμβολο για κάθε M
Lb = 100000; % Συνολικός αριθμός bits
Es = 1; % Αριθμός bits ανά σύμβολο για κάθε M
Rs = 250e3; % Ρυθμός συμβόλων (250 Ksymbols/sec)
Ts = 1 / Rs; % Διάρκεια συμβόλου
fc = 2.5e6; % Συχνότητα φέρουσας (2.5 MHz)
Tc = 1 / fc; % Περίοδος φέρουσας
Tsamp = Tc / 4; % Περίοδος δειγματοληψίας (4 δείγματα ανά περίοδο φέρουσας
Ns = round(Ts / Tsamp); % Δείγματα ανά σύμβολο
SNR_dB_values = 0:2:20; % Εύρος SNR (dB)
```

Στη συνέχεια, όρισα μητρώα για την αποθήκευση απλού mapping και κωδικοποίησης Gray mapping.

```
%% Αρχικοποίηση αποθήκευσης BER
BER_simple = zeros(length(SNR_dB_values), length(M_values));
BER_gray = zeros(length(SNR_dB_values), 1); % Μόνο για M = 8 με κωδικοποίηση Gray
```

Έπειτα, υπάρχει η λούπα για κάθε M. Υπολογίζει τον αριθμό των bits για επεξεργασία και παράγει μια τυχαία δυαδική ακολουθία.



Σχήμα 19: BER για M=2, M=8 (απλή) και M=8 (Gray)

Τα αποτελέσματα υποδεικνύουν ότι για M=2 επιτυγχάνεται το χαμηλότερο BER σε όλες τις τιμές SNR, επιδεικνύοντας ανώτερη ανθεκτικότητα στο θόρυβο σε σύγκριση με το M=8. Για M=8, η κωδικοποίηση Gray υπερτερεί της απλής χαρτογράφησης, ιδίως σε χαμηλότερες τιμές SNR, καθώς ελαχιστοποιεί τα σφάλματα bit εξασφαλίζοντας ότι τα γειτονικά σημεία αστερισμού διαφέρουν μόνο κατά ένα bit. Καθώς αυξάνεται το SNR, το BER μειώνεται για όλα τα σχήματα, με το M=2 να φτάνει σε σχεδόν μηδενικό BER σε πολύ χαμηλότερα επίπεδα SNR. Αντίθετα, η διαμόρφωση υψηλότερης τάξης (M=8) εμφανίζει συμβιβασμό μεταξύ του αυξημένου ρυθμού δεδομένων και της μειωμένης ανθεκτικότητας στο θόρυβο, που αντικατοπτρίζεται στο υψηλότερο BER.

3.

Θεωρητικά

Το SER από την άλλη, είναι επίσης μια μετρική που δείχνει πόσα σύμβολα λήφθηκαν λανθασμένα. Είναι ο λόγος των λανθασμένων ληφθέντων συμβόλων προς τον συνολικό αριθμό των μεταδιδόμενων συμβόλων.

Ένα λανθασμένο σύμβολο μπορεί να περιέχει πολλαπλά λανθασμένα bits, εφόσον ένα σύμβολο περιέχει bits.

Ο λόγος που χρησιμοποιείται το SER είναι κυρίως για την ανάλυση συστημάτων διαμόρφωσης. Επικεντρώνοντας στα σύμβολα, παρά στα bits, το SER τονίζει τα λάθη που συμβαίνουν λόγω modulation του συστήματος.

Η σχέση που συνδέει το BER – SER είναι η εξής:

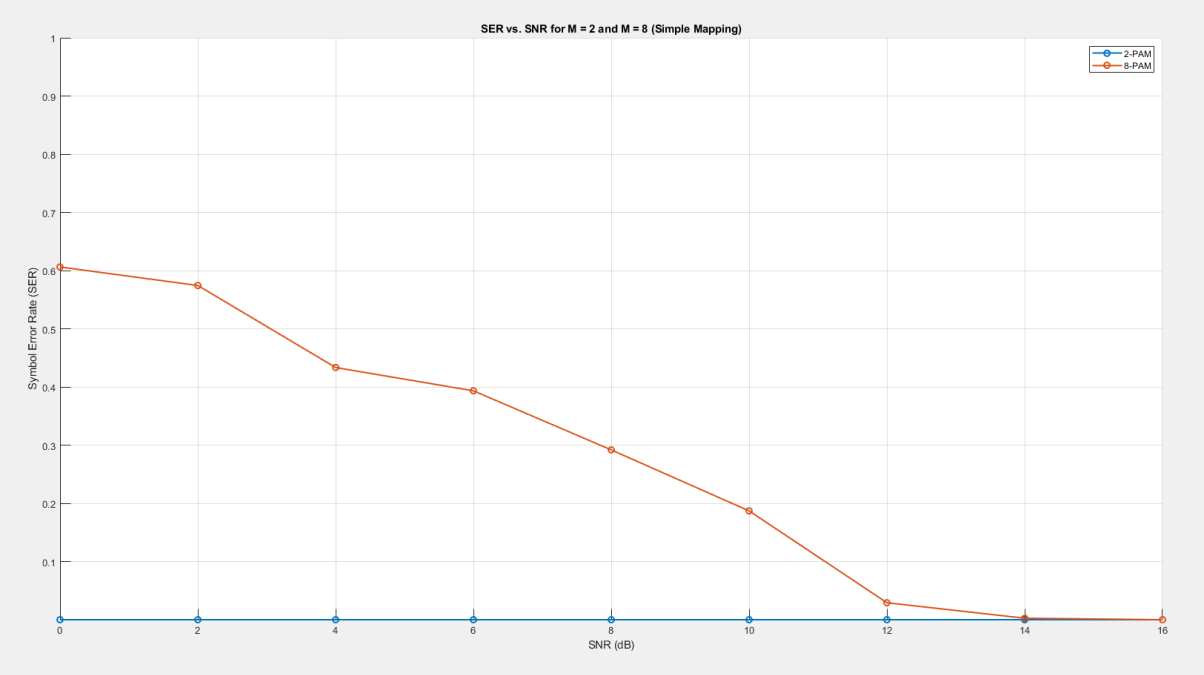
$$BER \approx SER \,/\, \log2(M)$$

Συνοψίζοντας:

BER: Χρήσιμο για την αξιολόγηση της συνολικής αξιοπιστίας του συστήματος, συμπεριλαμβάνοντας τον θόρυβο, την διαμόρφωση και την κωδικοποίηση

SER: Επικεντρώνεται στην ακρίβεια του modulation, και συγκεκριμένα στην ανίχνευση συμβόλων

Υλοποίηση



Σχήμα 20: SER για M=2 και M=8 (απλή)

Το 2-PAM έχει ανώτερη απόδοση σφάλματος σε όλες τις τιμές SNR λόγω του απλούστερου αστερισμού του με λιγότερα σύμβολα, τα οποία απέχουν περισσότερο μεταξύ τους. Αντίθετα, το 8-PAM υποφέρει από υψηλότερο

SER, ιδίως σε χαμηλό SNR, λόγω του πυκνότερου αστερισμού του και της μειωμένης απόστασης συμβόλων, καθιστώντας το πιο επιρρεπές σε σφάλματα που προκαλούνται από θόρυβο. Ωστόσο, σε υψηλά επίπεδα SNR, η απόδοση σφάλματος του 8-PAM βελτιώνεται σημαντικά.

ΚΩΔΙΚΑΣ

Ερωτήματα Μέρους Α1

1.

a.

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

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be smaller than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    x_confined = max(min(x, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;                % Αριθμός επιπέδων κβαντισμού
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12; % Μικρό ε για αποφυγή σφάλματα στρογγυλοποίησης
    xq_indices = round((x_confined - min_value + epsilon) / delta);
    xq_indices = max(min(xq_indices, L-1), 0);

    % Βήμα 4: xq ως δείκτης στο διάνυσμα centers
    xq = centers(xq_indices + 1);
    xq = reshape(xq, size(x));

end

% Αρχικοποίηση Παραμέτρων Σήματος
F = 4;                  % Συχνότητα αναλογικού σήματος (Hz)
Fs = 50;                % Συχνότητα δειγματοληψίας (Hz)
Ts = 1/Fs;              % Περίοδος δειγματοληψίας
t = 0:Ts:1;             % Βήμα δειγματοληψίας
x = sin(2*pi*F*t);      % Αναλογικό σήμα

% Παράμετροι Ομοιόμορφου Κβαντισμού
N = 3;                  % Αριθμός bits
min_value = -1;         % Ελάχιστη αποδεκτή υιμή του σήματος εισόδου
max_value = 1;          % Μέγιστη αποδεκτή υιμή του σήματος εισόδου

% Κλήση Συνάρτησης
[xq, centers] = my_quantizer(x, N, min_value, max_value);
```

```
% Υπολογισμός Δεικτών Κβαντισμού
delta = (max_value - min_value) / (2^N);  % Βήμα κβαντισμού
xq_indices = round((xq - min_value) / delta);  % Ανάκτηση δεικτών κβαντισμού
xq_indices = max(min(xq_indices, 2^N - 1), 0); % Περιορισμός στις επιτρεπτές τιμές

% Plot
figure;

% Αρχικό σήμα και δείγματα
subplot(4, 1, 1);
plot(t, x, 'b', 'LineWidth', 1.5); hold on;
stem(t, x, 'r', 'LineWidth', 1.2); grid on;
title('Original Signal and Samples');
xlabel('Time (s)'); ylabel('Amplitude');
legend('Original Signal', 'Sampled Points');

% Κβαντισμένο σήμα
subplot(4, 1, 2);
stem(t, xq, 'g', 'LineWidth', 1.5); grid on;
title('Quantized Signal');
xlabel('Time (s)'); ylabel('Amplitude');
legend('Quantized Values');

% Δείκτες κβαντισμού
subplot(4, 1, 3);
stem(t, xq_indices, 'k', 'LineWidth', 1.5); grid on;
title('Quantization Indices');
xlabel('Time (s)'); ylabel('Indices');
legend('Quantization Levels');

% Δυαδική Κωδικοποίηση
binary_encoded = dec2bin(xq_indices, N);
subplot(4, 1, 4);
hold on;
for i = 1:length(t)
    text(t(i), xq_indices(i), binary_encoded(i, :), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', ...
        'FontSize', 8, 'Color', 'blue');
end
stem(t, xq_indices, 'k', 'LineWidth', 1.5); grid on;
title('Binary Encoding of Quantization Indices');
xlabel('Time (s)'); ylabel('Indices');
legend('Binary Encoded');
```

b.

```
% Lloyd-Max Αλγόριθμος
function [xq, centers, D] = Lloyd_max(x, N, min_value, max_value)

    % Αρχικοποίηση ομοιόμορφων παραμέτρων
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = linspace(min_value + delta / 2, max_value - delta / 2, L);
    boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];

    D = []; % Διάνυσμα που κρατάει τις παραμορφώσεις
    convergence_rates = []; % Διάνυσμα που κρατάει τις διαφορές παραμορφώσεων

    % Επαναλήψεις
    while true
        % Βήμα 1: Υπολογισμός ορίων ζωνών κβαντισμού
        xq = zeros(size(x));
        for i = 1:length(x)
            for j = 1:length(boundaries) - 1
```

```
                if x(i) >= boundaries(j) && x(i) < boundaries(j + 1)
                    xq(i) = centers(j);
                    break;
                end
            end
        end

        % Βήμα 2: Υπολογισμός παραμόρφωσης
        distortion = mean((x - xq).^2);
        D = [D, distortion];

        % Υπολογισμός διαφοράς παραμόρφωσης (για τη σύγκλιση)
        if length(D) > 1
            convergence_rates = [convergence_rates, abs(D(end) - D(end-1))];
        end

        % Βήμα 3: Ενημέρωση επιπέδων κβαντισμού
        new_centers = zeros(1, L);
        counts = zeros(1, L);
        for i = 1:length(x)
            for j = 1:L
                if xq(i) == centers(j)
                    new_centers(j) = new_centers(j) + x(i);
                    counts(j) = counts(j) + 1;
                    break;
                end
            end
        end

        for j = 1:L
            if counts(j) > 0
                new_centers(j) = new_centers(j) / counts(j);
            else
                new_centers(j) = centers(j);
            end
        end

        % Σύγκλιση για να σταματήσει η λούπα
        if length(D) > 1 && abs(D(end) - D(end-1)) < 1e-6
            break;
        end

        centers = new_centers;
        boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];
    end
end


% Αρχικοποίηση Παραμέτρων Σήματος
F = 4;                  % Συχνότητα αναλογικού σήματος (Hz)
Fs = 50;                % Συχνότητα δειγματοληψίας (Hz)
Ts = 1/Fs;              % Περίοδος δειγματοληψίας
t = 0:Ts:1;             % Βήμα δειγματοληψίας
x = sin(2*pi*F*t);      % Αναλογικό σήμα

% Παράμετροι Ομοιόμορφου Κβαντισμού
N = 3;                  % Αριθμός bits
min_value = -1;         % Ελάχιστη αποδεκτή υιμή του σήματος εισόδου
max_value = 1;          % Μέγιστη αποδεκτή υιφή του σήματος εισόδου

% Κλήση Συνάρτησης
[xq, centers, D] = Lloyd_max(x, N, min_value, max_value);

% Plot
figure;

% Corrected Plot Section

% Plot 1: Αρχικό και Κβαντισμένο Σήμα
subplot(2, 2, 1);
```

```
plot(t, x, 'b', 'LineWidth', 1.5); hold on;
stairs(t, xq, 'r', 'LineWidth', 1.2);
title('Original vs. Quantized Signal');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Original Signal', 'Quantized Signal');
grid on;

% Plot 2: Κέντρα Κβαντισμού
subplot(2, 2, 2);
stem(centers, ones(size(centers)), 'g', 'LineWidth', 1.5);
title('Quantization Centers');
xlabel('Amplitude');
ylabel('Indicator');
grid on;

% Plot 3: Παραμόρφωση σε κάθε επανάληψη
subplot(2, 2, 3);
plot(1:length(D), D, '-o', 'LineWidth', 1.5);
title('Distortion Over Iterations');
xlabel('Iteration');
ylabel('Mean Squared Distortion');
grid on;

% Plot 4: Σύγκλιση Παραμόρφωσης
% Calculate convergence rates directly here
convergence_rates = abs(diff(D));
subplot(2, 2, 4);
plot(2:length(D), convergence_rates, '-s', 'LineWidth', 1.5);
title('Distortion Convergence');
xlabel('Iteration');
ylabel('|D_{i} - D_{i-1}|');
grid on;
```

2.

i.

```
% Lloyd-Max Αλγόριθμος
function [xq, centers, D] = Lloyd_max(x, N, min_value, max_value)

    % Αρχικοποίηση ομοιόμορφων παραμέτρων
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = linspace(min_value + delta / 2, max_value - delta / 2, L);
    boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];

    D = []; % Διάνυσμα που κρατάει τις παραμορφώσεις

    while true
        % Βήμα 1: Υπολογισμός ορίων ζωνών κβαντισμού
        xq = zeros(size(x));
        for i = 1:length(x)
            for j = 1:length(boundaries) - 1
                if x(i) >= boundaries(j) && x(i) < boundaries(j + 1)
                    xq(i) = centers(j);
                    break;
                end
            end
```

```
        end

        % Βήμα 2: Υπολογισμός παραμόρφωσης
        distortion = mean((x - xq).^2);
        D = [D, distortion];

        % Βήμα 3: Ενημέρωση επιπέδων κβαντισμού
        new_centers = zeros(1, L);
        counts = zeros(1, L);
        for i = 1:length(x)
            for j = 1:L
                if xq(i) == centers(j)
                    new_centers(j) = new_centers(j) + x(i);
                    counts(j) = counts(j) + 1;
                    break;
                end
            end
        end

        for j = 1:L
            if counts(j) > 0
                new_centers(j) = new_centers(j) / counts(j);
            else
                new_centers(j) = centers(j);
            end
        end

        % Σύγκλιση για να σταματήσει η λούπα
        if length(D) > 1 && abs(D(end) - D(end-1)) < 1e-6
            break;
        end

        centers = new_centers;
        boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];
    end
end

clear; clc;

% Φόρτωση του αρχείου ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y));

% Παράμετροι
min_value = -1;
max_value = 1;
epsilon = 1e-6; % Threshold σύγκλισης
bits = [2, 4, 8]; % Bit-depth
colors = ['r', 'g', 'b']; % Χρώματα για το Plot

% Αποθήκευση τιμών SQNR
SQNR_all = cell(length(bits), 1);

figure;
hold on;

for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση Συνάρτησης
    [xq, centers, D] = Lloyd_max(y, N, min_value, max_value);

    % Υπολογισμός SQNR σε κάθε επανάληψη
    SQNR = 10 * log10(mean(y.^2) ./ D);

    % Αποθήκευση τιμών SQNR για plot
    SQNR_all{idx} = SQNR;

    % Plot SQNR
    plot(1:length(SQNR), SQNR, 'Color', colors(idx), 'LineWidth', 1.5, ...
```

```
        'DisplayName', sprintf('N = %d', N));
end

% Plot
xlabel('Iteration');
ylabel('SQNR (dB)');
title('SQNR Evolution for Lloyd-Max Algorithm');
legend show;
grid on;
hold off;
```

ii.

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

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be smaller than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    x_confined = max(min(x, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;                  % Αριθμός επιπέδων κβαντισμού
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12; % Μικρό ε για αποφυγή σφάλματα στρογγυλοποίησης
    xq_indices = round((x_confined - min_value + epsilon) / delta);
    xq_indices = max(min(xq_indices, L-1), 0);

    % Βήμα 4: xq ως δείκτης στο διάνυσμα centers
    xq = centers(xq_indices + 1);
    xq = reshape(xq, size(x));

end

% Φόρτωση αρχείου ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y)); % Κανονικοποίηση σήματος στο διάστημα [-1, 1]

% Παράμετροι
min_value = -1;
max_value = 1;
bits = [2, 4, 8];

% Αρχικοποίηση πίνακα για αποθήκευση τιμών SQNR για κάθε N
SQNR_results = zeros(length(bits), 1);

% Επαναλήψεις για κάθε N
for b = 1:length(bits)
    N = bits(b);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq, ~] = my_quantizer(y, N, min_value, max_value);

    % Υπολογισμός SQNR
    signal_power = mean(y(:).^2);
    noise_power = mean((y(:) - xq(:)).^2);
    SQNR_results(b) = 10 * log10(signal_power / noise_power);
```

```
end

% Plot για τα διαφορετικά N
figure;
plot(bits, SQNR_results, '-o', 'LineWidth', 1.5, 'MarkerSize', 8);
xlabel('Quantization Levels (N bits)');
ylabel('SQNR (dB)');
title('SQNR vs Quantization Levels');
grid on;
```

iii.

```
% Lloyd-Max Αλγόριθμος
function [xq, centers, D] = Lloyd_max(x, N, min_value, max_value)

    % Αρχικοποίηση ομοιόμορφων παραμέτρων
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = linspace(min_value + delta / 2, max_value - delta / 2, L);
    boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];

    D = []; % Διάνυσμα που κρατάει τις παραμορφώσεις

    while true
        % Βήμα 1: Υπολογισμός ορίων ζωνών κβαντισμού
        xq = zeros(size(x));
        for i = 1:length(x)
            for j = 1:length(boundaries) - 1
                if x(i) >= boundaries(j) && x(i) < boundaries(j + 1)
                    xq(i) = centers(j);
                    break;
                end
            end
        end

        % Βήμα 2: Υπολογισμός παραμόρφωσης
        distortion = mean((x - xq).^2);
        D = [D, distortion];

        % Βήμα 3: Ενημέρωση επιπέδων κβαντισμού
        new_centers = zeros(1, L);
        counts = zeros(1, L);
        for i = 1:length(x)
            for j = 1:L
                if xq(i) == centers(j)
                    new_centers(j) = new_centers(j) + x(i);
                    counts(j) = counts(j) + 1;
                    break;
                end
            end
        end

        for j = 1:L
            if counts(j) > 0
                new_centers(j) = new_centers(j) / counts(j);
            else
                new_centers(j) = centers(j);
            end
        end

        % Σύγκλιση για να σταματήσει η λούπα
        if length(D) > 1 && abs(D(end) - D(end-1)) < 1e-6
            break;
        end
```

```
        centers = new_centers;
        boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];
    end
end

% Ομοιόμορφος Κβαντιστής
function [xq, centers] = my_quantizer(x, N, min_value, max_value)

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'To min_value prepei na einai mikrotero tou max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'To N prepei na einai 8etikos akeraios');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    x_confined = max(min(x, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;                % Αριθμός επιπέδων κβαντισμού
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12; % Small tolerance to avoid numerical issues
    xq_indices = round((x_confined - min_value + epsilon) / delta);
    xq_indices = max(min(xq_indices, L-1), 0);

    % Βήμα 4: xq ως δείκτης στο διάνυσμα centers
    xq = centers(xq_indices + 1);
    xq = reshape(xq, size(x));

end

clear; clc;

% Φόρτωση ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y));

% Παράμετροι
min_value = -1;
max_value = 1;
bits = [2, 4, 8];

% Επιλογές για το plot
line_styles = {'-', '--', ':'};
colors = ['r', 'g', 'b'];
lloyd_colors = ['m', 'c', 'y'];

figure;
hold on;

% Λούπα για τα N
for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq_uniform, ~] = my_quantizer(y, N, min_value, max_value);

    % Κλήση συνάρτησης Lloyd-Max
    [xq_lloyd, ~, ~] = Lloyd_max(y, N, min_value, max_value);

    % Αρχικός ήχος
    fprintf('Playing original signal...\n');
    sound(y, fs);
    pause(length(y) / fs + 1); % Αναμονή μέχρι να τελειώσει ο ήχος

    fprintf('Playing uniform quantized signal for N = %d...\n', N);
```

```
    sound(xq_uniform, fs);
    pause(length(xq_uniform) / fs + 1);

    fprintf('Playing Lloyd-Max quantized signal for N = %d...\n', N);
    sound(xq_lloyd, fs);
    pause(length(xq_lloyd) / fs + 1);

    % Plot κυματομορφών
    subplot(length(bits), 1, idx);
    plot(y, 'k', 'LineWidth', 1.2, 'DisplayName', 'Original');
    hold on;
    plot(xq_uniform, [colors(idx) line_styles{1}], 'LineWidth', 1, ...
         'DisplayName', sprintf('Uniform N = %d', N));
    plot(xq_lloyd, [lloyd_colors(idx) line_styles{2}], 'LineWidth', 1, ...
         'DisplayName', sprintf('Lloyd-Max N = %d', N));
    title(sprintf('Waveform Comparison for N = %d', N));
    xlabel('Sample Index');
    ylabel('Amplitude');
    legend;
    grid on;
end

hold off;
sgtitle('Waveform Comparisons for Original and Quantized Signals');
```

iv.

```
clear; clc;

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

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be smaller than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    x_confined = max(min(x, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;                % Αριθμός επιπέδων κβαντισμού
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12; % Μικρό e για αποφυγή σφάλματα στρογγυλοποίησης
    xq_indices = round((x_confined - min_value + epsilon) / delta);
    xq_indices = max(min(xq_indices, L-1), 0);

    % Βήμα 4: xq ως δείκτης στο διάνυσμα centers
    xq = centers(xq_indices + 1);
    xq = reshape(xq, size(x));

end

% Lloyd-Max Αλγόριθμος
function [xq, centers, D] = Lloyd_max(x, N, min_value, max_value)

    % Αρχικοποίηση ομοιόμορφων παραμέτρων
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = linspace(min_value + delta / 2, max_value - delta / 2, L);
```

```
    boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];

    D = []; % Διάνυσμα που κρατάει τις παραμορφώσεις

    while true
        % Βήμα 1: Υπολογισμός ορίων ζωνών κβαντισμού
        xq = zeros(size(x));
        for i = 1:length(x)
            for j = 1:length(boundaries) - 1
                if x(i) >= boundaries(j) && x(i) < boundaries(j + 1)
                    xq(i) = centers(j);
                    break;
                end
            end
        end

        % Βήμα 2: Υπολογισμός παραμόρφωσης
        distortion = mean((x - xq).^2);
        D = [D, distortion];

        % Βήμα 3: Ενημέρωση επιπέδων κβαντισμού
        new_centers = zeros(1, L);
        counts = zeros(1, L);
        for i = 1:length(x)
            for j = 1:L
                if xq(i) == centers(j)
                    new_centers(j) = new_centers(j) + x(i);
                    counts(j) = counts(j) + 1;
                    break;
                end
            end
        end

        for j = 1:L
            if counts(j) > 0
                new_centers(j) = new_centers(j) / counts(j);
            else
                new_centers(j) = centers(j);
            end
        end

        % Σύγκλιση για να σταματήσει η λούπα
        if length(D) > 1 && abs(D(end) - D(end-1)) < 1e-6
            break;
        end

        centers = new_centers;
        boundaries = [min_value, (centers(1:end-1) + centers(2:end)) / 2, max_value];
    end
end

% Φόρτωση ήχου
[y, fs] = audioread('speech.wav');
y = y / max(abs(y));

% Παράμετροι
min_value = -1;
max_value = 1;
bits = [2, 4, 8];

% Αρχικοποίηση πινάκων MSE για κάθε συνάρτηση
mse_uniform = zeros(size(bits));
mse_lloyd = zeros(size(bits));

% Λούπα για κάθε N
for idx = 1:length(bits)
    N = bits(idx);

    % Κλήση συνάρτησης ομοιόμορφου κβαντισμού
    [xq_uniform, ~] = my_quantizer(y, N, min_value, max_value);
```

```
    mse_uniform(idx) = mean((y - xq_uniform).^2); % Υπολογισμός MSE

    % Κλήση συνάρτησης Lloyd-Max
    [xq_lloyd, ~, ~] = Lloyd_max(y, N, min_value, max_value);
    mse_lloyd(idx) = mean((y - xq_lloyd).^2); % Υπολογισμός MSE
end

% Plot MSE
figure;
plot(bits, mse_uniform, '-o', 'Color', 'b', 'LineWidth', 1.5, ...
     'DisplayName', 'Uniform Quantizer');
hold on;
plot(bits, mse_lloyd, '--s', 'Color', 'r', 'LineWidth', 1.5, ...
     'DisplayName', 'Lloyd-Max Quantizer');

xlabel('Bit Depth (N)');
ylabel('Mean Squared Error (MSE)');
title('MSE vs. Bit Depth for Uniform and Lloyd-Max Quantizers');
legend('Location', 'northeast');
grid on;
hold off;
```

Ερωτήματα - Μέρους Α2

1.

```
function yq = my_quantizer(y, N, min_value, max_value)

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be less than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    y_confined = max(min(y, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12;
    yq_index = round((y_confined - min_value + epsilon) / delta);
    yq_index = max(min(yq_index, L-1), 0); % Clamp to valid index range

    % Βήμα 4: Χαρτογράφηση του δείκτη στην αντίστοιχη κβαντισμένη τιμή
    yq = centers(yq_index + 1);

end

close all;

% Αρχικοποίηση Παραμέτρων
p = 8;
N = 3;
min_value = -3.5;
max_value = 3.5;

% Φόρτωση δειγμάτων
data = load('source.mat');
```

```
variables = fieldnames(data);
x = data.(variables{1});

x_length = length(x);

% Βήμα 1: Υπολογισμός πίνακα αυτοσυσχέτισης R και διάνυσμα αυτοσυσχέτισης r
R = zeros(p, p);
r = zeros(p, 1);

for i = 1:p
    sum_r = 0;
    for n = p+1:x_length
       sum_r = sum_r + x(n) * x(n-i);
    end
    r(i) = sum_r / (x_length - p);

    for j = 1:p
        sum_R = 0;
        for n = p+1:x_length
           sum_R = sum_R + x(n-j) * x(n-i);
        end
        R(i, j) = sum_R / (x_length - p);
    end
end

% Βήμα 2: Υπολογισμός συντελεστών και κβαντισμός τους
a = R \ r; % Yule-Walker
for i = 1:p
    a(i) = my_quantizer(a(i), 8, -2, 2);
end

% Βήμα 3: DPCM Κωδικοποίηση
memory = zeros(p, 1);
y = zeros(x_length, 1);
y_hat = zeros(x_length, 1);
y_toned = 0;

for i = 1:x_length
    y(i) = x(i) - y_toned; % Πρόβλεψη σφάλματος
    y_hat(i) = my_quantizer(y(i), N, min_value, max_value); % Κβαντισμός error
    y_hat_toned = y_hat(i) + y_toned; % Προσθήκη σφάλμα κβαντισμού στη πρόβλεψη
    memory = [y_hat_toned; memory(1:p-1)];
    y_toned = a' * memory;
end

% Βήμα 4: DPCM Αποκωδικοποίηση
memory = zeros(p, 1);
y_toned = 0;
y_final = zeros(x_length, 1);

for i = 1:x_length
    y_final(i) = y_hat(i) + y_toned; % Ανακατασκευή
    memory = [y_final(i); memory(1:p-1)];
    y_toned = a' * memory;
end

% Plot αρχικού και ανακατασκευασμένου σήματος
figure;
plot(1:x_length, x, 'b', 1:x_length, y_final, 'r');
legend('Original Signal', 'Reconstructed Signal');
title('Original vs Reconstructed Signal');
xlabel('Sample Index');
ylabel('Amplitude');

% Plot πρόβλεψη σφάλματος
figure;
plot(1:x_length, y, 'k');
title('Prediction Error');
xlabel('Sample Index');
ylabel('Amplitude');
```

```
plot(x) % Αρχικό σήμα
hold
plot(y) % Σήμα πρόβλεψης σφάλματος
xlabel('Input signal x')
ylabel('Error Prediction Signal y')
```

2.

```
function yq = my_quantizer(y, N, min_value, max_value)

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be less than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    y_confined = max(min(y, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12;
    yq_index = round((y_confined - min_value + epsilon) / delta);
    yq_index = max(min(yq_index, L-1), 0); % Clamp to valid index range

    % Βήμα 4: Χαρτογράφηση του δείκτη στην αντίστοιχη κβαντισμένη τιμή
    yq = centers(yq_index + 1);

end

close all;

% Παράμετροι
p_values = [5, 10]; % Δύο τιμέσ για το p
N_values = [1, 2, 3];
min_value = -3.5;
max_value = 3.5;

% Φόρτωση δειγμάτων
data = load('source.mat');
variables = fieldnames(data);
x = data.(variables{1});

x_length = length(x);

% Λούπα για όλα τα p και N
figure;
for p_idx = 1:length(p_values)
    p = p_values(p_idx);

    R = zeros(p, p);
    r = zeros(p, 1);
    for i = 1:p
        sum_r = 0;
        for n = p+1:x_length
            sum_r = sum_r + x(n) * x(n-i);
```

```
        end
        r(i) = sum_r / (x_length - p);

        for j = 1:p
            sum_R = 0;
            for n = p+1:x_length
                sum_R = sum_R + x(n-j) * x(n-i);
            end
            R(i, j) = sum_R / (x_length - p);
        end
    end

    a = R \ r;
    for i = 1:p
        a(i) = my_quantizer(a(i), 8, -2, 2);
    end

    for N_idx = 1:length(N_values)
        N = N_values(N_idx);

        % Αρχικοποίηση παραμέτρων για DPCM
        memory = zeros(p, 1);
        y = zeros(x_length, 1);
        y_hat = zeros(x_length, 1);
        y_toned = 0;

        for i = 1:x_length
            y(i) = x(i) - y_toned;
            y_hat(i) = my_quantizer(y(i), N, min_value, max_value);
            y_hat_toned = y_hat(i) + y_toned;
            memory = [y_hat_toned; memory(1:p-1)];
            y_toned = a' * memory;
        end

        % Plot αρχικού σήματος και πρόβλεψης σφάλματος
        subplot(length(p_values), length(N_values), (p_idx-1)*length(N_values) + N_idx);
        plot(1:x_length, x, 'b', 1:x_length, y, 'r');
        title(sprintf('p = %d, N = %d bits', p, N));
        legend('Original Signal', 'Prediction Error');
        xlabel('Sample Index');
        ylabel('Amplitude');
    end
end
```

3.

```
function yq = my_quantizer(y, N, min_value, max_value)
% Διασφάλιση ότι min_value < max_value
assert(min_value < max_value, 'min_value must be less than max_value');
% Διασφάλιση ότι N > 0
assert(N > 0 && floor(N) == N, 'N must be a positive integer');
% Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
% max_value]
y_confined = max(min(y, max_value), min_value);
% Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
% περιοχής
L = 2^N;
delta = (max_value - min_value) / L;
centers = min_value + (0:L-1) * delta;
% Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
epsilon = 1e-12;
yq_index = round((y_confined - min_value + epsilon) / delta);
yq_index = max(min(yq_index, L-1), 0); % Clamp to valid index range
% Βήμα 4: Χαρτογράφηση του δείκτη στην αντίστοιχη κβαντισμένη τιμή
yq = centers(yq_index + 1);
```

```
end
% Define parameters
p_values = 5:10; % Predictor orders (from 5 to 10)
N_values = [1, 2, 3]; % Bits for quantization
min_value = -3.5;
max_value = 3.5;
% Load input data
data = load('source.mat');
variables = fieldnames(data);
x = data.(variables{1});
x_length = length(x);
% Initialize results
mse_results = zeros(length(p_values), length(N_values));
coefficients = cell(length(p_values), 1);
% Loop through p and N values
for p_idx = 1:length(p_values)
p = p_values(p_idx);
% Calculate R and r for this p
R = zeros(p, p);
r = zeros(p, 1);
for i = 1:p
sum_r = 0;
for n = p+1:x_length
sum_r = sum_r + x(n) * x(n-i);
end
r(i) = sum_r / (x_length - p);
for j = 1:p
sum_R = 0;
for n = p+1:x_length
sum_R = sum_R + x(n-j) * x(n-i);
end
R(i, j) = sum_R / (x_length - p);
end
end
% Solve for a and quantize coefficients
a = R \ r; % Solve Yule-Walker equations
% Apply threshold to remove insignificant coefficients
threshold = 0.1 * max(abs(a)); % 10% of the maximum coefficient
a(abs(a) < threshold) = 0;
% Quantize coefficients
for i = 1:p
a(i) = my_quantizer(a(i), 10, -2, 2); % Quantize to 10 bits
end
coefficients{p_idx} = a; % Store quantized coefficients
% Evaluate for each N
for N_idx = 1:length(N_values)
N = N_values(N_idx);
% Initialize variables for DPCM
memory = zeros(p, 1);
y = zeros(x_length, 1);
y_hat = zeros(x_length, 1);
y_toned = 0;
% DPCM Encoding
for i = 1:x_length
y(i) = x(i) - y_toned; % Prediction error
y_hat(i) = my_quantizer(y(i), N, min_value, max_value); % Quantize error
y_hat_toned = y_hat(i) + y_toned; % Add quantized error to prediction
memory = [y_hat_toned; memory(1:p-1)]; % Update memory
y_toned = a' * memory; % Update prediction
end
% Calculate MSE
mse_results(p_idx, N_idx) = mean((x - y_hat).^2);
end
end
% Plot MSE vs N for each p
figure;
for p_idx = 1:length(p_values)
plot(N_values, mse_results(p_idx, :), '-o', 'DisplayName', sprintf('p =%d',p_values(p_idx)));
hold on;
end
```

```
grid on;
title('MSE vs N for Different Predictor Orders');
xlabel('Number of Bits (N)');
ylabel('Mean Squared Error (MSE)');
legend;
% Display quantized predictor coefficients
disp('Quantized Predictor Coefficients:');
for p_idx = 1:length(p_values)
fprintf('p = %d: ', p_values(p_idx));
disp(coefficients{p_idx}');
end
```

4.

```
function yq = my_quantizer(y, N, min_value, max_value)

    % Διασφάλιση ότι min_value < max_value
    assert(min_value < max_value, 'min_value must be less than max_value');

    % Διασφάλιση ότι N > 0
    assert(N > 0 && floor(N) == N, 'N must be a positive integer');

    % Βήμα 1: Περιορισμός δυναμικής περιοχής του σήματος στις τιμές [min_value,
    % max_value]
    y_confined = max(min(y, max_value), min_value);

    % Βήμα 2: Υπολογισμός του βήματος κβαντισμού Δ και των κεντρών κάθε
    % περιοχής
    L = 2^N;
    delta = (max_value - min_value) / L;
    centers = min_value + (0:L-1) * delta;

    % Βήμα 3: Εύρεση περιοχής που ανήκει το κάθε δείγμα
    epsilon = 1e-12;
    yq_index = round((y_confined - min_value + epsilon) / delta);
    yq_index = max(min(yq_index, L-1), 0); % Clamp to valid index range

    % Βήμα 4: Χαρτογράφηση του δείκτη στην αντίστοιχη κβαντισμένη τιμή
    yq = centers(yq_index + 1);

end

% Define parameters
p_values = [5, 10]; % Predictor orders
N_values = [1, 2, 3]; % Bits for quantization
min_value = -3.5;
max_value = 3.5;

% Load input data
data = load('source.mat');
variables = fieldnames(data);
x = data.(variables{1});

x_length = length(x);

% Loop through p and N values
figure;
plot_idx = 1; % Subplot index
for p_idx = 1:length(p_values)
    p = p_values(p_idx);

    % Calculate R and r for this p
    R = zeros(p, p);
    r = zeros(p, 1);
    for i = 1:p
```

```
        sum_r = 0;
        for n = p+1:x_length
            sum_r = sum_r + x(n) * x(n-i);
        end
        r(i) = sum_r / (x_length - p);

        for j = 1:p
            sum_R = 0;
            for n = p+1:x_length
                sum_R = sum_R + x(n-j) * x(n-i);
            end
            R(i, j) = sum_R / (x_length - p);
        end
    end

    % Solve for a and quantize coefficients
    a = R \ r;
    for i = 1:p
        a(i) = my_quantizer(a(i), 8, -2, 2);
    end

    for N_idx = 1:length(N_values)
        N = N_values(N_idx);

        % Initialize variables for DPCM
        memory = zeros(p, 1);
        y_hat = zeros(x_length, 1);
        y_toned = 0;

        % DPCM Encoding
        for i = 1:x_length
            y(i) = x(i) - y_toned; % Prediction error
            y_hat(i) = my_quantizer(y(i), N, min_value, max_value); % Quantize error
            y_hat_toned = y_hat(i) + y_toned; % Add quantized error to prediction
            memory = [y_hat_toned; memory(1:p-1)]; % Update memory
            y_toned = a' * memory; % Update prediction
        end

        % DPCM Decoding
        memory = zeros(p, 1);
        y_toned = 0;
        y_final = zeros(x_length, 1);
        for i = 1:x_length
            y_final(i) = y_hat(i) + y_toned; % Reconstruct signal
            memory = [y_final(i); memory(1:p-1)];
            y_toned = a' * memory; % Update prediction
        end

        % Plot original and reconstructed signal
        subplot(length(p_values), length(N_values), plot_idx);
        plot(1:x_length, x, 'b', 1:x_length, y_final, 'r');
        title(sprintf('p = %d, N = %d bits', p, N));
        legend('Original Signal', 'Reconstructed Signal');
        xlabel('Sample Index');
        ylabel('Amplitude');
        plot_idx = plot_idx + 1;
    end
end
```

Ερωτήματα – Μέρος Β

1.

```
clc;
```

```
clear;
close all;

%% Παράμετροι
M = 8; % Τάξη M-PAM (μπορεί επίσης να δοκιμαστεί με M=2)
log2M = log2(M); % Αριθμός bits ανά σύμβολο
Lb = 100000; % Συνολικός αριθμός bits
Lb = floor(Lb / log2M) * log2M; % Προσαρμογή ώστε να είναι διαιρετός με log2(M)
Es = 1; % Ενέργεια ανά σύμβολοl
Rs = 250e3; % Ρυθμός συμβόλων (250 Ksymbols/sec)
Ts = 1 / Rs; % Διάρκεια συμβόλου
fc = 2.5e6; % Συχνότητα φέρουσας (2.5 MHz)
Tc = 1 / fc; % Περίοδος φέρουσας
Tsamp = Tc / 4; % Περίοδος δειγματοληψίας (4 δείγματα ανά περίοδο φέρουσας)
Ns = round(Ts / Tsamp); % Αριθμός δειγμάτων ανά σύμβολο
SNR_dB_values = 0:2:20; % Εύρος SNR (dB)

%% Δημιουργία Δυαδικής Ακολουθίας Εισόδου
bits = randi([0 1], 1, Lb); % Τυχαία δυαδική ακολουθία

%% Mapping: Bits σε σύμβολα
symbols = reshape(bits, log2M, []); % Ομαδοποίηση bits σε log2(M) ανά σύμβολο
decSymbols = bi2de(symbols.', 'left-msb'); % Μετατροπή σε δεκαδικά σύμβολα
Am = (2 * (0:M-1) - (M - 1)) * sqrt(Es / (2 * (M - 1)^2)); % Επίπεδα πλάτους PAM

%% Διαμόρφωση παλμού
gT = ones(1, Ns); % Ορθογώνιος Παλμός
gT = gT / sqrt(sum(gT.^2)); % Κανονικοποίηση στη μονάδα ενέργειας

s_t = reshape(repmat(Am(decSymbols + 1), Ns, 1), 1, []); % Σήμα PAM με διαμόρφωση παλμού

%% Διαμόρφωση Σήματος
num_samples = length(s_t); % Συνολικός αριθμός δειγμάτων στο s_tt
t_carrier = linspace(0, (num_samples-1)*Tsamp, num_samples); % Χρονικό διάνυσμα για τη φέρουσα
carrier = cos(2 * pi * fc * t_carrier); % Κυματομορφή φέρουσας
transmitted_signal = s_t .* carrier; % Διαμορφωμένο σήμα

%% Οπτικοποίηση Μεταδιδόμενων Συμβόλων (Αντιστοίχιση)
figure;
stem(1:20, decSymbols(1:20), 'filled', 'LineWidth', 1.5);
title('Transmitted Symbols (Mapping)');
xlabel('Symbol Index');
ylabel('Amplitude');
grid on;

%% Αρχικοποίηση BER και SER
BER_values = zeros(size(SNR_dB_values)); % Αποθήκευση BER
SER_values = zeros(size(SNR_dB_values)); % Αποθήκευση SER

%% Υπολογισμός BER και SER
for idx = 1:length(SNR_dB_values)
    SNR_dB = SNR_dB_values(idx);
    SNR_linear = 10^(SNR_dB / 10);

    % Προσθήκη AWGN
    No = Es / (SNR_linear * log2M); % Ισχύς φασματικής πυκνότητας θορύβου
    sigma = sqrt(No / 2); % Τυπική απόκλιση θορύβου
    noise = sigma * randn(size(transmitted_signal));
    received_signal = transmitted_signal + noise;

    % Αποδιαμόρφωση
    demodulated_signal = received_signal .* carrier; % Πολλαπλασιασμός με φέρουσα

    % Ολοκλήρωση
    integrated_signal = zeros(1, length(decSymbols)); % Αρχικοποίηση ολοκληρωμένου σήματος
    for i = 1:length(decSymbols)
        start_idx = (i-1)*Ns + 1;
        end_idx = i*Ns;
        integrated_signal(i) = sum(demodulated_signal(start_idx:end_idx) .* gT);
    end
```

```
    % Κλιμάκωση ολοκληρωμένου σήματος
    integrated_signal = integrated_signal / max(abs(integrated_signal)) * max(abs(Am));

    % Ανίχνευση Συμβόλων
    detected_symbols = zeros(size(integrated_signal));
    for i = 1:length(integrated_signal)
        [~, detected_symbols(i)] = min(abs(integrated_signal(i) - Am));
% Ανίχνευση πλησιέστερου πλάτους
    end
    detected_symbols = detected_symbols - 1; % Προσαρμογή στο εύρος [0, M-1]

    % Οπτικοποίηση Ληφθέντων Συμβόλων (Αντιστοίχιση) για SNR = 20 dB
    if SNR_dB == 20
        figure;
        stem(1:20, detected_symbols(1:20), 'filled', 'LineWidth', 1.5);
        title('Ληφθέντα Σύμβολα (Αντιστοίχιση)');
        xlabel('Δείκτης Συμβόλου');
        ylabel('Πλάτος');
        grid on;
    end

    % Αντιστοίχιση: Σύμβολα σε Bits
    detected_bits = de2bi(detected_symbols, log2M, 'left-msb')'; % Μετατροπή συμβόλων σε bits
    detected_bits = detected_bits(:)'; % Επίπεδη αναπαράσταση σε 1D πίνακα

    % Υπολογισμός BER
    if ~isempty(detected_bits)
        minLength = min(length(bits), length(detected_bits)); % Ευθυγράμμιση μηκών
        BER_values(idx) = sum(bits(1:minLength) ~= detected_bits(1:minLength)) / Lb;
% Κανονικοποίηση με τον συνολικό αριθμό bits
    else
        BER_values(idx) = NaN; % Ανάθεση NaN αν η ανίχνευση αποτύχει
        fprintf('Η ανίχνευση απέτυχε για SNR = %d dB\n', SNR_dB);
    end

    % Υπολογισμός SER
    decSymbols = decSymbols(:)'; % Διασφάλιση ότι είναι διάνυσμα γραμμής
    detected_symbols = detected_symbols(:)'; % Διασφάλιση ότι είναι διάνυσμα γραμμής
    num_symbol_errors = sum(detected_symbols ~= decSymbols); % Μέτρηση λαθών συμβόλων
    SER_values(idx) = num_symbol_errors / length(decSymbols); % Αναλογία λανθασμένων συμβόλων
end

%% Debugging
disp('First 10 transmitted symbols:');
disp(decSymbols(1:10));

disp('First 10 detected symbols (SNR = 20 dB):');
disp(detected_symbols(1:10));

fprintf('Amplitude levels: [%f, %f]\n', min(Am), max(Am));
fprintf('Integrated signal range: [%f, %f]\n', min(integrated_signal), max(integrated_signal));

figure;
semilogy(SNR_dB_values, BER_values, '-o', 'LineWidth', 1.5);
hold on;
semilogy(SNR_dB_values, SER_values, '-s', 'LineWidth', 1.5);
title('BER and SER vs. SNR');
xlabel('SNR (dB)');
ylabel('Error Rate');
legend('BER', 'SER');
grid on;
```

2.

```
clc;
clear;
close all;

%% Παράμετροι
M_values = [2, 8]; % Επίπεδα M-PAM
log2M_values = log2(M_values); % Αριθμός bits ανά σύμβολο για κάθε M
Lb = 100000; % Συνολικός αριθμός bits
Es = 1; % Αριθμός bits ανά σύμβολο για κάθε M
Rs = 250e3; % Ρυθμός συμβόλων (250 Ksymbols/sec)
Ts = 1 / Rs; % Διάρκεια συμβόλου
fc = 2.5e6; % Συχνότητα φέρουσας (2.5 MHz)
Tc = 1 / fc; % Περίοδος φέρουσας
Tsamp = Tc / 4; % Περίοδος δειγματοληψίας (4 δείγματα ανά περίοδο φέρουσας
Ns = round(Ts / Tsamp); % Δείγματα ανά σύμβολο
SNR_dB_values = 0:2:20; % Εύρος SNR (dB)

%% Αρχικοποίηση αποθήκευσης BER
BER_simple = zeros(length(SNR_dB_values), length(M_values));
BER_gray = zeros(length(SNR_dB_values), 1); % Μόνο για M = 8 με κωδικοποίηση Gray

%% Προσομοίωση για κάθε M
for m_idx = 1:length(M_values)
    M = M_values(m_idx);
    log2M = log2M_values(m_idx);
    Lb_adj = floor(Lb / log2M) * log2M; % Προσαρμογή αριθμού bits
    bits = randi([0 1], 1, Lb_adj); % Τυχαία δυαδική ακολουθία

    % Αντιστοίχιση Συμβόλων (Απλή)
    symbols = reshape(bits, log2M, []); % Ομαδοποίηση bits για κάθε σύμβολο
    decSymbols_simple = bi2de(symbols.', 'left-msb'); % Δεκαδικά σύμβολα (Απλή)
    Am_simple = (2 * (0:M-1) - (M-1)) * sqrt(Es / (2 * (M-1)^2));
    gray_map = bitxor(0:M-1, floor((0:M-1)/2)); % Δημιουργία αντιστοίχισης κώδικα Gray
    Am_gray = Am_simple(gray_map + 1); % Αντιστοίχιση πλατών χρησιμοποιώντας τον κώδικα Gray

    % Αντιστοίχιση Συμβόλων (Gray για M = 8)
    if M == 8
        gray_map = bitxor(0:M-1, floor((0:M-1)/2)); % Αντιστοίχιση κώδικα Gray
        Am_gray = Am_simple; % Ίδια επίπεδα πλάτους
        graySymbols = gray_map(decSymbols_simple + 1); % Εφαρμογή αντιστοίχισης Gray
    end

    % Διαμόρφωση PAM (Διαμόρφωση παλμού)
    gT = ones(1, Ns) / sqrt(Ns); % Κανονικοποιημένος ορθογώνιος παλμός
    s_t_simple = reshape(repmat(Am_simple(decSymbols_simple + 1), Ns, 1), 1, []);
% Απλή αντιστοίχιση
    if M == 8
        s_t_gray = reshape(repmat(Am_gray(graySymbols + 1), Ns, 1), 1, []); % Αντιστοίχιση Gray
    end

    % Διαμόρφωση φέρουσας
    num_samples = length(s_t_simple);
    t_carrier = linspace(0, (num_samples-1)*Tsamp, num_samples); % Χρονικό διάνυσμα
    carrier = cos(2 * pi * fc * t_carrier); % Κυματομορφή φέρουσας
    transmitted_signal_simple = s_t_simple .* carrier;
    if M == 8
        transmitted_signal_gray = s_t_gray .* carrier;
    end

    % Υπολογισμός BER
    for snr_idx = 1:length(SNR_dB_values)
        SNR_dB = SNR_dB_values(snr_idx);
        SNR_linear = 10^(SNR_dB / 10);

        % Προσθήκη θορύβου
        No = Es / (SNR_linear * log2M); % Ισχύς φασματικής πυκνότητας θορύβου
        sigma = sqrt(No / 2); % Τυπική απόκλιση θορύβου AWGN
        noise_simple = sigma * randn(size(transmitted_signal_simple));
% Θόρυβος για απλή αντιστοίχιση
```

```
        if M == 8
            noise_gray = sigma * randn(size(transmitted_signal_gray));
% Θόρυβος για αντιστοίχιση Gray
        end

        received_signal_simple = transmitted_signal_simple + noise_simple;
        if M == 8
            received_signal_gray = transmitted_signal_gray + noise_gray;
        end

        % Αποδιαμόρφωση και Ανίχνευση (Απλή)
        demodulated_simple = received_signal_simple .* carrier; % Πολλαπλασιασμός με φέρουσα
        integrated_simple = zeros(1, length(decSymbols_simple));
        for i = 1:length(decSymbols_simple)
            start_idx = (i-1)*Ns + 1;
            end_idx = i*Ns;
            integrated_simple(i) = sum(demodulated_simple(start_idx:end_idx) .* gT);
        end
        % Ανίχνευση πλησιέστερου επιπέδου πλάτους
        integrated_simple = integrated_simple / max(abs(integrated_simple))*max(abs(Am_simple));
        [~, detected_simple] = min(abs(integrated_simple(:) - Am_simple), [], 2);
% Διασφάλιση ότι το Am_simple είναι διάνυσμα γραμμής
        detected_simple = detected_simple - 1; % Προσαρμογή στο εύρος [0, M-1]

        % Μετατροπή ανιχνευμένων συμβόλων σε δυαδικά bits
        detected_bits = de2bi(detected_simple, log2M, 'left-msb')'; % Μετατροπή σε δυαδικό
        detected_bits = detected_bits(:)'; % Μετατροπή σε διάνυσμα γραμμής

        % Ευθυγράμμιση μηκών και υπολογισμός BER
        minLength = min(length(bits), length(detected_bits)); % Ευθυγράμμιση μηκών
        BER_simple(snr_idx, m_idx)=sum(bits(1:minLength)~=detected_bits(1:minLength))/minLength;

        detected_simple = detected_simple - 1; % Προσαρμογή στο εύρος [0, M-1]

        % BER για Απλή Αντιστοίχιση
        detected_bits = detected_bits(:)'; % Μετατροπή ανιχνευμένων bits σε διάνυσμα γραμμής
        minLength = min(length(bits), length(detected_bits)); % Ευθυγράμμιση μηκών
        BER = sum(bits(1:minLength) ~= detected_bits(1:minLength)) / Lb;


        % Αποδιαμόρφωση και Ανίχνευση (Gray, μόνο για M = 8)
        if M == 8
            demodulated_gray = received_signal_gray .* carrier;
            integrated_gray = zeros(1, length(graySymbols));
            for i = 1:length(graySymbols)
                start_idx = (i-1)*Ns + 1;
                end_idx = i*Ns;
                integrated_gray(i) = sum(demodulated_gray(start_idx:end_idx) .* gT);
            end
            integrated_gray = integrated_gray / max(abs(integrated_gray)) * max(abs(Am_gray));
            [~, detected_gray] = min(abs(integrated_gray(:) - Am_gray), [], 2);
            detected_gray = detected_gray - 1; % Προσαρμογή στο [0, M-1]
            % Αντιστροφή αντιστοίχισης Gray
            inv_gray_map = zeros(1, M);
            inv_gray_map(gray_map + 1) = 0:M-1;
            decoded_gray = inv_gray_map(detected_gray + 1);
% Χαρτογράφηση ανιχνευμένων συμβόλων πίσω στα αρχικά

            detected_bits_gray = de2bi(decoded_gray, log2M, 'left-msb')';
            detected_bits_gray = detected_bits_gray(:)';
            minLength_gray = min(length(bits), length(detected_bits_gray)); % Ευθυγράμμιση μηκών

BER_gray(snr_idx)=sum(bits(1:minLength_gray)~=detected_bits_gray(1:minLength_gray))/Lb_adj;

        end
    end

    disp('BER για M=2:');
    disp(BER_simple(:, 1));
```

```
end

%% Plot
figure;
semilogy(SNR_dB_values, BER_simple(:, 1), '-o', 'LineWidth', 1.5);
hold on;
semilogy(SNR_dB_values, BER_simple(:, 2), '-s', 'LineWidth', 1.5);
semilogy(SNR_dB_values, BER_gray, '-d', 'LineWidth', 1.5);
title('BER vs. SNR for M = 2 & 8');
xlabel('SNR (dB)');
ylabel('BER');
legend('M=2 (Simple)', 'M=8 (Simple)', 'M=8 (Gray)');
grid on;
```

3.

```
clc;
clear;
close all;

%% Παράμετροι
M_values = [2, 8]; % Επίπεδα M-PAM
log2M_values = log2(M_values); % Αριθμός bits ανά σύμβολο για κάθε M
Lb = 100000; % Συνολικός αριθμός bits
Es = 1; % Ενέργεια ανά σύμβολο
Rs = 250e3; % Ρυθμός συμβόλων (250 Ksymbols/sec)
Ts = 1 / Rs; % Διάρκεια συμβόλου
fc = 2.5e6; % Συχνότητα φέρουσας (2.5 MHz)
Tc = 1 / fc; % Περίοδος φέρουσας
Tsamp = Tc / 4; % Περίοδος δειγματοληψίας (4 δείγματα ανά περίοδο φέρουσας)
SNR_dB_values = 0:2:20; % Εύρος SNR (dB)

%% Αποθήκευση SER
SER_values = zeros(length(SNR_dB_values), length(M_values)); % Αποθήκευση SER για κάθε M

%% Επανάληψη για τις τιμές M
for m_idx = 1:length(M_values)
    M = M_values(m_idx);
    log2M = log2M_values(m_idx);
    Lb = floor(Lb / log2M) * log2M; % Προσαρμογή ώστε να είναι διαιρετός με log2(M)
    bits = randi([0 1], 1, Lb); % Τυχαία δυαδική ακολουθία

    % Αντιστοίχιση: Bits σε Σύμβολα
    symbols = reshape(bits, log2M, []); % Ομαδοποίηση bits σε log2(M) ανά σύμβολο
    decSymbols = bi2de(symbols.', 'left-msb'); % Μετατροπή σε δεκαδικά σύμβολα
    Am = (2 * (0:M-1) - (M - 1)) * sqrt(Es / (2 * (M - 1)^2)); % Επίπεδα πλάτους PAM

    % Διαμόρφωση Παλμού
    Ns = round(Ts / Tsamp); % Αριθμός δειγμάτων ανά σύμβολο
    gT = ones(1, Ns); % Ορθογώνιος παλμός
    gT = gT / sqrt(sum(gT.^2)); % Κανονικοποίηση σε μοναδιαία ενέργεια
    s_t = reshape(repmat(Am(decSymbols + 1), Ns, 1), 1, []); % Σήμα PAM με διαμόρφωση παλμού

    % Διαμόρφωση
    t_carrier = linspace(0, (length(s_t)-1)*Tsamp, length(s_t));
% Χρονικό διάνυσμα για τη φέρουσα
    carrier = cos(2 * pi * fc * t_carrier); % Κυματομορφή φέρουσας
    transmitted_signal = s_t .* carrier; % Διαμορφωμένο σήμα

    %% Υπολογισμός SER για κάθε τιμή SNR
    for idx = 1:length(SNR_dB_values)
        SNR_dB = SNR_dB_values(idx);
        SNR_linear = 10^(SNR_dB / 10);

        % Προσθήκη AWGN
```

```
        No = Es / (SNR_linear * log2M); % Ισχύς φασματικής πυκνότητας θορύβου
        sigma = sqrt(No / 2); % Τυπική απόκλιση θορύβου
        noise = sigma * randn(size(transmitted_signal));
        received_signal = transmitted_signal + noise;

        % Αποδιαμόρφωση
        demodulated_signal = received_signal .* carrier; % Πολλαπλασιασμός με φέρουσα

        % Ολοκλήρωση
        integrated_signal = zeros(1, length(decSymbols)); % Αρχικοποίηση ολοκληρωμένου σήματος
        for i = 1:length(decSymbols)
            start_idx = (i-1)*Ns + 1;
            end_idx = i*Ns;
            integrated_signal(i) = sum(demodulated_signal(start_idx:end_idx) .* gT);
        end
        integrated_signal = integrated_signal / max(abs(integrated_signal)) * max(abs(Am));
% Κλιμάκωση σήματος

        % Ανίχνευση Συμβόλων
        detected_symbols = zeros(size(integrated_signal));
        for i = 1:length(integrated_signal)
            [~, detected_symbols(i)] = min(abs(integrated_signal(i) - Am));
% Ανίχνευση πλησιέστερου πλάτους
        end
        detected_symbols = detected_symbols - 1; % Προσαρμογή στο εύρος [0, M-1]

        % Διασφάλιση ευθυγράμμισης διανυσμάτων
        decSymbols = decSymbols(:)'; % Μετατροπή σε διάνυσμα γραμμής
        detected_symbols = detected_symbols(:)'; % Μετατροπή σε διάνυσμα γραμμής

assert(length(decSymbols)==length(detected_symbols),'Ασυμφωνίασταμήκηδιανυσμάτωνσυμβόλων.');

        % Υπολογισμός αριθμού λαθών συμβόλων
        num_symbol_errors = sum(detected_symbols ~= decSymbols); % Μέτρηση λαθών συμβόλων

        % Υπολογισμός SER
        SER_values(idx, m_idx) = num_symbol_errors / length(decSymbols);
% Ποσοστό λανθασμένων συμβόλων
    end
end

%% Plot SER
figure;
hold on;
for m_idx = 1:length(M_values)
    semilogy(SNR_dB_values,SER_values(:,m_idx), ...
'o','LineWidth',1.5,'DisplayName',sprintf('%dPAM', M_values(m_idx)));
end
title('SER vs. SNR for M = 2 and M = 8 (Simple Mapping)');
xlabel('SNR (dB)');
ylabel('Symbol Error Rate (SER)');
legend show;
grid on;
ylim([1e-5, 1]);
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. https://engineerstutor.com/2018/08/09/quantization-in-pcm-with-example/