Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή

Τομέας Υλικού και Αρχιτεκτονικής των Υπολογιστών
Διδάσκων: Κωνσταντίνος Μπερμπερίδης
Ακαδημαϊκό Έτος: 2023 – 2024
Ημ/νία Παράδοσης: 15/01/2024

Ψηφιακές Τηλεπικοινωνίες
1ο Σετ Εργαστηριακών Ασκήσεων

Μάθημα Κορμού – CEID_NY384
Χειμερινό Εξάμηνο 2023

Στοιχεία Φοιτητή:

Ονοματεπώνυμο:	Μηλτιάδης Μαντές
Α.Μ.:	1084661
E – mail:	up1084661@upnet.gr
Εξάμηνο:	Ζ

ΜΕΡΟΣ Α – ΘΕΩΡΙΑ ΠΛΗΡΟΦΟΡΙΑΣ

Αρχικά, θα φορτώσουμε την πηγή μας, η οποία αναπαρίσταται ως ένας πίνακας ακεραίων $I \in Z^{N \times M}$ και έπειτα τη μετατρέπουμε σε ένα διάνυσμα *image_vector* για να μπορούμε να την επεξεργαστούμε στη συνέχεια.

```
% Read the PNG image
image = imread('parrot.png');

% Convert the image to a column vector
image_vector = image(:);
```

Βέβαια δεν γνωρίζουμε αν η πηγή μας είναι με ή χωρίς μνήμη. Έχει σημασία να προσδιορίσουμε αυτή τη παράμετρο, καθώς είναι απαραίτητο να γνωρίζουμε αν υπάρχουν συσχετίσεις μεταξύ των συμβόλων για αποδοτικότερη κωδικοποίηση. Υπολογίζουμε γι' αυτό το λόγο τη συσχέτιση κάθε εικονοστοιχείου με τα άμεσα γειτονικά του.

```
image_double = double(image);
image_vector = image_double(:);
lagged_image_vector = [image_vector(2:end); 0];  % Shift and pad with zero
[correlation, p_value] = corrcoef(image_vector(1:end-1), lagged_image_vector(1:end-1));
disp(['Correlation: ', num2str(correlation(2))]);
disp(['P-value: ', num2str(p_value(2))]);
```

Αν τρέξουμε τον κώδικα παρατηρούμε ότι η συσχέτιση ισούται με 0.94861, δηλαδή απέχει αρκετά από το 0, γεγονός που σημαίνει ότι υπάρχει μια ισχυρή γραμμική συσχέτιση ανάμεσα στα γειτονικά εικονοστοιχεία και άρα έχουμε να κάνουμε με **πηγή με μνήμη**.

Για να περιγράψουμε τη πηγή μας αρκεί να υπολογίσουμε το αλφάβητό της $\mathbf{\Phi = \{s_1, s_2, \ldots, s_N\}}$, δηλαδή όλα τα διακριτά εικονοστοιχεία της εικόνας, καθώς και τις πιθανότητες εμφάνισης κάθε εικονοστοιχείου $\mathbf{P = (p_1, p_2, \ldots, p_N\}}$.

(1) a. Αρχικά, καλείστε να εντοπίσετε τα σύμβολα της πηγής (δηλαδή τις διακριτές τιμές τις οποίες λαμβάνουν τα pixels της εικόνας) και να εκτιμήσετε τις πιθανότητες εμφάνισης τους.

Αρχικά, έχουμε τη $1^{ης}$ τάξης επέκταση πηγής, για την οποία μπορούμε να εργαστούμε με την παραδοχή ότι όλα τα συμβολοστοιχεία είναι ανεξάρτητα μεταξύ τους. Χρησιμοποιούμε τη συνάρτηση **[C,ia,ic] = unique(___)** της MATLAB πάνω στο διάνυσμα της εικόνας, η οποία μας επιστρέφει τις διακριτές τιμές των εικονοστοιχείων της εικόνας και τις αποθηκεύει στο διάνυσμα *unique_values*. Επίσης, χρησιμοποιώντας τη συνάρτηση **[n] = numel(___)** πάλι πάνω στο διάνυσμα εικόνας αποθηκεύουμε στο διάνυσμα *total_pixels* το συνολικό αριθμό στοιχειών του διανύσματος, δηλαδή όλα τα εικονοστοιχεία της εικόνας. Τέλος, για να εκτιμήσουμε τη πιθανότητα εμφάνισης κάθε εικονοστοιχείου αρχικά μέσω της συνάρτησης **[N] = histcounts(C,Categories)** κρατάμε την ιστογραμματική κατανομή των στοιχειών στο διάνυσμα *unique_values*, η οποία παράγει τον αριθμό εμφανίσεων κάθε διακριτού στοιχείου. Έπειτα, διαιρούμε τον αριθμό εμφανίσεων κάθε διακριτού στοιχείου με το συνολικό πλήθος εικονοστοιχείων και έτσι προκύπτει η πιθανότητα εμφάνισής του, την οποία και αποθηκεύουμε στο διάνυσμα *appearance_probabilities*.

```
%First Order Source
% Find unique pixel values and their appearance probabilities
[unique_values, ~, pixel_counts] = unique(image_vector);
total_pixels = numel(image_vector);
appearance_probabilities = histcounts(image_vector, [unique_values; max(unique_values)+1]) / total_pixels;
disp('First Order Source Pixel Values and Appearance Probabilities:');
for i = 1:length(unique_values)
    % Concatenate the values and probabilities as a string
    result_string = strcat(num2str(unique_values(i)), ' , ', num2str(appearance_probabilities(i)));
    disp(result_string);
end
```

Τρέχοντας το κομμάτι κώδικα λαμβάνουμε τα εξής αποτελέσματα:

```
First Order Source Pixel Values and Appearance Probabilities:
0 ,0.096167
17 ,0.0814
34 ,0.068333
51 ,0.0625
68 ,0.076833
85 ,0.090467
102 ,0.11317
119 ,0.0906
136 ,0.0965
153 ,0.067133
170 ,0.0389
187 ,0.032933
204 ,0.033667
221 ,0.0259
238 ,0.022367
255 ,0.0031333
```

(1) b. Στην συνέχεια, υπολογίστε την κωδικοποίηση Huffman για την συγκεκριμένη πηγή.

Η κωδικοποίηση Huffman αποτελεί μια περίπτωση προθεματικού κώδικα ο οποίος δημιουργεί ένα δυαδικό δένδρο με φύλλα τα διακριτά εικονοστοιχεία της πηγής και ρίζα αλλά και ενδιάμεσους κόμβους σύνθετα εικονοστοιχεία. Τα σύνθετα εικονοστοιχεία προκύπτουν αν ταξινομήσουμε τα αρχικά εικονοστοιχεία με βάση τη φθίνουσα πιθανότητα εμφάνισης και ενώσουμε τα δύο στοιχεία με τις μικρότερες πιθανότητες σε ένα καινούργιο στοιχείο με πιθανότητα το άθροισμα των δύο προηγούμενων πιθανότητων. Έπειτα, ανατίθενται στα δύο αρχικά εικονοστοιχεία οι τιμές 0, 1 και έχουμε εκ νέου ταξινόμηση και επαναληπτική εφαρμογή αυτού του αλγορίθμου. Τέλος, για να γίνει η κωδικοποίηση ξεκινάμε με κατεύθυνση προς κάθε φύλλο ξεχωριστά από τη ρίζα του δένδρου, οπότε προκύπτουν όλες οι διαφορετικές κωδικοποιήσεις των στοιχείων.

Σε κώδικα MATLAB η κωδικοποίηση πραγματοποιείται εφαρμόζοντας τη συνάρτηση **[dict,avglen] = huffmandict(symbols,prob)** πάνω στο διάνυσμα με τις διακριτές τιμές των στοιχειών μαζί με τις πιθανότητες εμφάνισής τους. Το λεξικό που προκύπτει είναι το huffman_dict και έχει ως κλειδί το κάθε εικονοστοιχείο και ως περιεχόμενο του κλειδιού τη κωδικοποίηση Huffman που του αντιστοιχεί.

```
% Encode the first order image using Huffman coding
huffman_dict = huffmandict(unique_values, appearance_probabilities);
disp('Huffman Encoding:');
disp(huffman_dict);
```

Η τελική κωδικοποίηση φαίνεται πιο κάτω:

```
Huffman Encoding:
    {[  0]}    {[       1 1 0]}
    {[ 17]}    {[     0 0 1 0]}
    {[ 34]}    {[     0 1 0 0]}
    {[ 51]}    {[     0 1 1 1]}
    {[ 68]}    {[     0 0 1 1]}
    {[ 85]}    {[     0 0 0 0]}
    {[102]}    {[       1 0 0]}
    {[119]}    {[       1 1 1]}
    {[136]}    {[       1 0 1]}
    {[153]}    {[     0 1 0 1]}
    {[170]}    {[   0 0 0 1 1]}
    {[187]}    {[   0 1 1 0 1]}
    {[204]}    {[   0 1 1 0 0]}
    {[221]}    {[  0 0 0 1 0 0]}
    {[238]}    {[0 0 0 1 0 1 0]}
    {[255]}    {[0 0 0 1 0 1 1]}
```

(i) Να υπολογίσετε την εντροπία της κωδικοποίησης σας.

Η εντροπία της κωδικοποίησης H(X) είναι το μέτρο της μέσης ποσότητας πληροφορίας ανά στοιχείο και αντιπροσωπεύει με λίγα λόγια το θεωρητικό κάτω φράγμα του μέσου μήκους κώδικα κάθε εικονοστοιχείου της πηγής. Δίνεται από τον τύπο:

$$\mathbf{H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)} \qquad \mathbf{(I)}$$

Αυτόν τον τύπο εφαρμόζουμε και στην υλοποίηση σε MATLAB μέσω της συνάρτησης **[C] = dot(A,B)** πάνω στο διάνυσμα *appearance_probabilities*, καθώς ο τύπος στην ουσία εκφράζει το εσωτερικό γινόμενο των διανυσμάτων *appearance_probabilities* και $\log_2$(*appearance_probabilities*) και η τελική τιμή αποθηκεύεται στην μεταβλητή entropy.

```
% Calculate entropy
entropy = -dot(appearance_probabilities, log2(appearance_probabilities));
disp('Entropy of Huffman Encoding:');
disp(num2str(entropy));
```

Έτσι, έχουμε το εξής αποτέλεσμα που εκτυπώνεται:

```
Entropy of Huffman Encoding:
3.7831
```

(ii) Να υπολογίσετε το μέσο μήκος κώδικα.

Το μέσο μήκος κώδικα έχει να κάνει με το μέσο πλήθος δυαδικών ψηφίων που χρησιμοποιούνται κατά τη κωδικοποίηση Huffman. Επομένως, θα μπορούσαμε να πούμε ότι είναι ένας δείκτης της αποδοτικότητας της συμπίεσης πληροφορίας. Δίνεται από τον τύπο:

$$\mathbf{L = \sum_{i=1}^{n} p(x_i) l(x_i)} \qquad \textbf{(II)}$$

Επομένως, εκφράζεται ξανά το εσωτερικό γινόμενο ανάμεσα στο διάνυσμα *appearance_probabilities* και *huffman_code_lengths*.

Για να αποθηκεύσουμε το μήκος της κωδικοποίησης κάθε εικονοστοιχείου μεταβαίνουμε στο λεξικό που δημιουργήσαμε πριν και αποθηκεύουμε στο διάνυσμα *huffman_code_lengths* όλα τα μήκη κώδικα, δηλαδή όλες τις τιμές της δεύτερης στήλης αφού πρώτα εφαρμόσουμε σε αυτές τη συνάρτηση [**L] = length(X)**, με ακρίβεια δύο δεκαδικών ψηφίων. Στη συνέχεια, καλούμε πάλι τη συνάρτηση [**C] = dot(A,B)** πάνω στα διανύσματα *appearance_probabilities* και *huffman_code_lengths* και η τελική τιμή αποθηκεύεται στην μεταβλητή mean_length.

```
% Calculate mean length of code
% Convert Huffman codes to double
huffman_codes_double = cellfun(@double, huffman_dict(:, 2), 'UniformOutput', false);
% Create a vector of lengths of Huffman codes
huffman_code_lengths = cellfun(@length, huffman_codes_double);
mean_length = dot(huffman_code_lengths, appearance_probabilities);
disp('Mean Length of Huffman Encoding:');
disp(num2str(mean_length));
```

Έτσι, έχουμε το εξής αποτέλεσμα που εκτυπώνεται:

```
Mean Length of Huffman Encoding:
3.8374
```

(iii) Να υπολογίσετε την αποδοτικότητα του κώδικα σας. Σχολιάστε συνοπτικά τα αποτελέσματά σας.

Η απόδοση της κωδικοποίησης δείχνει πόσο κοντά βρίσκεται ο κωδικοποιητής στο όριο συμπίεσης της πηγής (εντροπία), δηλαδή αν η συμπίεση μας εκτελείται σωστά και χάνεται μικρή ποσότητα της αρχικής πληροφορίας. Η απόδοση παίρνει τιμές στο διάστημα [0, 1] και όσο πιο κοντά στο 1 τείνει τόσο καλύτερη είναι η κωδικοποίηση που πραγματοποιήσαμε. Δίνεται από τον τύπο:

$$\mathbf{\eta = \frac{H(X)}{L}} \qquad \textbf{(III)}$$

Επομένως, το μόνο που μένει να κάνουμε είναι να διαιρέσουμε τη μεταβλητή entropy με τη μεταβλητή mean_length.

```
% Calculate efficiency of code
efficiency = entropy / mean_length;
disp('Efficiency of Huffman Encoding:');
disp(num2str(efficiency));
```

Έτσι, έχουμε το εξής αποτέλεσμα που εκτυπώνεται:

```
Efficiency of Huffman Encoding:
0.98585
```

Συμπέρασμα

Παρατηρούμε ότι η απόδοση του κώδικα είναι πολύ κοντά στο 100% (98.59%), δηλαδή ο κώδικας που υλοποιήσαμε κάνει αρκετά ικανοποιητική συμπίεση δεδομένων, καθώς με την κωδικοποίηση αυτή το μέσο μήκος κώδικα προσεγγίζει την εντροπία.

Σε μια Ν-τάξης επέκταση πηγής τα σύμβολα δεν είναι πλέον ανεξάρτητα, αλλά κάθε νέο σύμβολο αποτελείται από Ν συνεχόμενα σύμβολα της αρχικής πηγής, για τα οποία λαμβάνονται υπόψη και οι συσχετίσεις μεταξύ τους. Συνεπώς, για τη $2^{ης}$ τάξης επέκταση, κάθε σύνθετο σύμβολο αποτελείται από τα ζεύγη των συνεχόμενων διακριτών τιμών των εικονοστοιχείων και συνεπώς η πιθανότητα εμφάνισης του σύνθετου συμβόλου έχει πλέον να κάνει με το πόσο πιθανό είναι να εμφανιστεί ένα συγκεκριμένο εικονοστοιχείο ακριβώς μετά από ένα άλλο. Το αλφάβητο της νέας πηγής θα είναι $\mathbf{\Phi^2 = \{\sigma_1, \sigma_2, \ldots, \sigma_K\}}$ με $\mathbf{\sigma_K = (s_i, s_j)}$ για i, j = 1, 2, …, N και πιθανότητες εμφάνισης $\mathbf{P^2 = \{p_1, p_2, \ldots, p_K\}}$. Να σημειωθεί ότι η συγκεκριμένη υλοποίηση που ακολουθεί οφείλεται στο γεγονός ότι η πηγή είναι διακριτή με μνήμη και η κατεύθυνση συσχέτισης που ακολουθούμε για κάθε ζεύγος είναι ανά γραμμή. Όμοια θα μπορούσαν οι συσχετίσεις να υπολογιστούν για κάθε ζεύγος του πίνακα και κατά στήλες, διαγωνίως κ. ο. κ.

(2) a. Θεωρήστε τώρα τη δεύτερης τάξης επέκταση πηγής του Ερωτήματος 1. Καλείστε να εντοπίσετε τα σύμβολα της δεύτερης τάξης επέκταση πηγής (ζεύγη χαρακτήρων) και για κάθε ζεύγος να εκτιμήσετε την πιθανότητα εμφάνισής του.

Για να υλοποιήσουμε την επέκταση ξεκινάμε ορίζοντας ένα array *pairs* που θα αποθηκεύει τα καινούργια σύμβολα (ζεύγη των διπλανών εικονοστοιχείων) παίρνοντας όλα τα εικονοστοιχεία εκτός από το τελευταίο και συνδυάζοντάς τα με τα επόμενα. Έπειτα, μέσω της **[C,ia,ic] = unique(___)** αποθηκεύουμε στο *unique_pairs* όλα τα μοναδικά ζεύγη του πίνακα με βάση τις γραμμές του και στο *pair_counts* μέσω της **[B] = accumarray(ind,data)** το πλήθος εμφανίσεων κάθε ζεύγους. Τέλος, βρίσκουμε τις από κοινού πιθανότητες διαιρώντας για κάθε ζεύγος το πλήθος εμφανίσεών του με το πλήθος των εμφανίσεων όλων των ζευγαριών και τις αποθηκεύουμε στο *pair_probabilities*.

```
% Second Order Source
% Create pairs of neighboring pixels
pairs = [image_vector(1:end-1), image_vector(2:end)];

% Find unique pairs and their counts
[unique_pairs, ~, idx] = unique(pairs, 'rows');
pair_counts = accumarray(idx, 1);

% Calculate probabilities
pair_probabilities = pair_counts / sum(pair_counts);

% Display the pairs and their probabilities
disp('Second Order Source Pixel Values and Appearance Probabilities:');
for i = 1:length(unique_pairs)
    disp([num2str(unique_pairs(i, :)), ', ', num2str(pair_probabilities(i))]);
end
```

Τρέχοντας το κομμάτι κώδικα λαμβάνουμε τα εξής αποτελέσματα:

0 0, 0.081036	0 68, 0.00030001	17 17, 0.050235	17 85, 0.00053335
0 17, 0.0129	0 85, 6.6669e-05	17 34, 0.013334	17 102, 0.00043335
0 34, 0.0013	0 102, 0.0001	17 51, 0.0026334	17 119, 0.00020001
0 51, 0.00046668	17 0, 0.012267	17 68, 0.0013	17 136, 0.00026668

17 153, 0.0001
17 170, 6.6669e-05
17 187, 3.3334e-05
34 0, 0.0011
34 17, 0.012934
34 34, 0.036001
34 51, 0.0119
34 68, 0.0025668
34 85, 0.0011667
34 102, 0.0010334
34 119, 0.00040001
34 136, 0.00036668
34 153, 0.00033334
34 170, 0.00020001
34 187, 0.00016667
34 204, 0.00013334
51 0, 0.00046668
51 17, 0.0031334
51 34, 0.012367
51 51, 0.028901
51 68, 0.012067
51 85, 0.0023667
51 102, 0.0014
51 119, 0.00073336
51 136, 0.00020001
51 153, 0.00033334
51 170, 0.00023334
51 187, 6.6669e-05
51 204, 0.0001
51 221, 0.0001
51 238, 3.3334e-05
68 0, 0.00016667
68 17, 0.0009667
68 34, 0.0026001
68 51, 0.0119
68 68, 0.041668
68 85, 0.0135
68 102, 0.0025668
68 119, 0.0014
68 136, 0.00073336
68 153, 0.00050002
68 170, 0.00020001
68 187, 0.00023334
68 204, 0.00013334
68 221, 0.00016667
68 238, 0.0001
85 0, 0.00030001
85 17, 0.00046668
85 34, 0.0016667
85 51, 0.0042335
85 68, 0.012967
85 85, 0.049268
85 102, 0.016501
85 119, 0.0024001
85 136, 0.0013334
85 153, 0.00040001
85 170, 0.00040001
85 187, 0.00023334
85 204, 0.0001
85 221, 0.0001
85 238, 0.0001
102 0, 0.00013334
102 17, 0.00020001
102 34, 0.00040001
102 51, 0.0017001
102 68, 0.0036001
102 85, 0.016467
102 102, 0.069802
102 119, 0.015567
102 136, 0.0026334
102 153, 0.0010667
102 170, 0.00056669
102 187, 0.00063335
102 204, 0.00016667
102 221, 0.0001
102 238, 0.0001
102 255, 3.3334e-05
119 0, 0.00013334
119 17, 0.00030001
119 34, 0.00020001
119 51, 0.00046668
119 68, 0.0017667
119 85, 0.0047002
119 102, 0.015734
119 119, 0.049902
119 136, 0.0133
119 153, 0.0024334
119 170, 0.00063335
119 187, 0.00056669
119 204, 0.00020001
119 221, 0.0001
119 238, 0.00013334
119 255, 3.3334e-05
136 0, 0.00046668
136 17, 0.00013334
136 34, 0.00030001
136 51, 0.00020001
136 68, 0.00036668
136 85, 0.0015667
136 102, 0.0047002
136 119, 0.015501
136 136, 0.056069
136 153, 0.0135
136 170, 0.0020001
136 187, 0.001
136 204, 0.00030001
136 221, 0.0001
136 238, 0.00013334
136 255, 0.00016667
153 0, 6.6669e-05
153 17, 3.3334e-05
153 34, 0.00013334
153 51, 6.6669e-05
153 68, 0.00013334
153 85, 0.00036668
153 102, 0.00063335
153 119, 0.0035001
153 136, 0.018301
153 153, 0.034234
153 170, 0.0068002
153 187, 0.0014667
153 204, 0.00093336
153 221, 0.00023334
153 238, 0.00016667
153 255, 6.6669e-05
170 17, 6.6669e-05
170 34, 3.3334e-05
170 51, 3.3334e-05
170 68, 0.0001
170 85, 0.00033334
170 102, 0.00013334
170 119, 0.00066669
170 136, 0.0022001
170 153, 0.010467
170 170, 0.016834
170 187, 0.0060002
170 204, 0.0013667
170 221, 0.00040001
170 238, 0.00016667
170 255, 0.0001

187 17, 3.3334e-05
187 85, 3.3334e-05
187 102, 6.6669e-05
187 119, 0.00023334
187 136, 0.00060002
187 153, 0.0024001
187 170, 0.0079336
187 187, 0.013567
187 204, 0.0059002
187 221, 0.0014667
187 238, 0.00050002
187 255, 0.00020001
204 102, 6.6669e-05
204 119, 3.3334e-05
204 136, 0.00043335
204 153, 0.0010667
204 170, 0.0022334
204 187, 0.0073336
204 204, 0.016667
204 221, 0.0046335
204 238, 0.001
204 255, 0.00020001
221 85, 6.6669e-05
221 119, 3.3334e-05
221 136, 6.6669e-05
221 153, 0.00013334
221 170, 0.00066669
221 187, 0.0013667
221 204, 0.0068002
221 221, 0.012834
221 238, 0.0036335
221 255, 0.00030001
238 85, 3.3334e-05
238 119, 3.3334e-05
238 153, 0.00013334
238 170, 6.6669e-05
238 187, 0.00026668
238 204, 0.0008667
238 221, 0.0055669
238 238, 0.0141
238 255, 0.0013
255 153, 3.3334e-05
255 170, 6.6669e-05
255 221, 0.0001
255 238, 0.0022001
255 255, 0.0007333

(2) b. Στην συνέχεια, υπολογίστε την κωδικοποίηση Huffman για την συγκεκριμένη πηγή. Να υπολογίσετε: **i)** την εντροπία της κωδικοποίησης σας, **ii)** το μέσο μήκος κώδικα και **iii)** την αποδοτικότητα του κώδικα σας. Σχολιάστε συνοπτικά τα αποτελέσματά σας.

Για τα υποερωτήματα **(i)**, **(ii)** και **(iii)** ακολουθούμε το ίδιο σκεπτικό που εφαρμόσαμε και στη 1ης τάξης πηγή.

```
% Convert pairs to numeric symbols
symbols = 1:size(unique_pairs, 1);

% Encode the second order image using Huffman coding and numeric symbols
[numericdict, ~] = huffmandict(symbols, pair_probabilities);

% Create a new dictionary with numeric pairs as one symbol for the first column
dict = cell(size(numericdict, 1), 2);
for i = 1:size(numericdict, 1)
    dict{i, 1} = unique_pairs(numericdict{i, 1}, :);
    dict{i, 2} = numericdict{i, 2};
end
disp('Second Order Huffman Encoding:');
disp(dict);

% Calculate entropy
second_order_entropy = -dot(pair_probabilities, log2(pair_probabilities));
disp('Entropy of Second Order Huffman Encoding:');
disp(num2str(second_order_entropy));

% Calculate mean length of code
% Create a vector of lengths of Huffman codes
huffman_code_lengths = cellfun(@length, dict(:, 2));
```

```
second_order_mean_length = dot(huffman_code_lengths, pair_probabilities);
disp('Mean Length of Second Order Huffman Encoding:');
disp(num2str(second_order_mean_length));

% Calculate efficiency of code
second_order_efficiency = second_order_entropy / second_order_mean_length;
disp('Efficiency of Second Order Huffman Encoding:');
disp(num2str(second_order_efficiency));
```

Τα αποτελέσματα που λαμβάνουμε αυτή τη φορά είναι τα εξής:

```
Entropy of Second Order Huffman Encoding:
5.6520

Mean Length of Second Order Huffman Encoding:
5.6801

Efficiency of Second Order Huffman Encoding:
0.99505
```

Συμπέρασμα

Παρατηρούμε ότι η απόδοση του κώδικα είναι ακόμα πιο κοντά στο 100% (99.50%), δηλαδή το μέσο μήκος κώδικα προσεγγίζει ακόμα περισσότερο την εντροπία. Επίσης, είναι αναμενόμενο με την επέκταση πηγής η εντροπία και το μέσο μήκος κώδικα να αυξάνονται επιτρέποντας έτσι τη χρήση πιο κοντών κωδίκων για συχνά εμφανιζόμενα μπλοκ και μακρύτερων κωδίκων για λιγότερο συχνά μπλοκ. Γι' αυτό έχουμε ξανά πολύ καλή συμπίεση δεδομένων.

(2) c. Πως συγκρίνονται τα αποτελέσματα με τα αντίστοιχα του Ερωτήματος 1;

Συγκρίνοντας τώρα την αρχική πηγή με την επέκτασή της παρατηρούμε ότι αυξάνεται η αποδοτικότητα του κώδικα, γεγονός το οποίο είναι αναμενόμενο μιας και αυτή είναι η χρησιμότητα της επέκτασης πηγής. Γενικότερα, η επέκταση επιτρέπει την ανάθεση κωδικών με βάση την πιθανότητα εμφάνισης ολόκληρων μπλοκ συμβόλων, αντί για μεμονωμένα σύμβολα, καθώς η παρουσία μνήμης μεταξύ των συμβόλων σημαίνει ότι κάποιοι συνδυασμοί είναι αναπόφευκτα πιο πιθανοί από κάποιους άλλους. Έτσι, πλησιάζουμε όλο και περισσότερο το όριο συμπίεσης.

Για επεκταμένη πηγή τάξης N χωρίς μνήμη αποδεικνύεται ότι:

$$\mathbf{H(X^N) = NH(X)} \qquad \mathbf{(IV)}$$

(3) a. Εξηγήστε συνοπτικά γιατί στη συγκεκριμένη περίπτωση δεν ισχύει ο τύπος: $H(X^2) = 2H(X)$ για τον υπολογισμό της εντροπίας της δεύτερης τάξης επέκτασης πηγής.

Επομένως, για N = 2 η σχέση **(IV)** γίνεται $H(X^2) = 2H(X)$. Ωστόσο, για να ισχύει η συγκεκριμένη σχέση θα πρέπει να συντρέχουν ταυτόχρονα τα εξής:

- Τα εικονοστοιχεία πρέπει να είναι ανεξάρτητα ΚΑΙ ισοπίθανα, ώστε η εμφάνιση ενός εικονοστοιχείου να μην επηρεάζει τη πιθανότητα εμφάνισης κάποιου άλλου.
- Η πηγή πρέπει να είναι χωρίς μνήμη ή συσχετίσεις, ώστε η πληροφορία που θα παρέχεται από το ένα σύμβολο να μην μπορεί να επηρεάσει την πληροφορία που θα παρέχεται από το επόμενο.

Ωστόσο, εφόσον η πηγή μας είναι με μνήμη δεν μπορούμε να θεωρήσουμε αυθαίρετα ότι τα εικονοστοιχεία είναι ανεξάρτητα άρα και ασυσχέτιστα, καθώς για παράδειγμα συνεχόμενα εικονοστοιχεία μπορεί να έχουν παρόμοιες τιμές λόγω της φύσης της εικόνας. Επιπλέον, όπως φαίνεται και από το ερώτημα **(2) a.** τα νέα σύμβολα που προκύπτουν δεν είναι όλα ισοπίθανα. Για αυτούς τους δύο λόγους λοιπόν η σχέση **(IV)** δεν ισχύει στη προκειμένη περίπτωση.

Για το μέσο μήκος κώδικα αποδεικνύεται ότι για οποιασδήποτε τάξης N επέκτασης της πηγής ισχύει η ανισότητα:

$$\mathbf{H(X^N) \leq L_N < H(X^N) + 1} \qquad \mathbf{(V)}$$

(3) b. Να παραθέσετε ένα φράγμα της μορφής $a \leq \bar{L} < b$ για το μέσο μήκος κώδικα που υπολογίσατε στα Ερωτήματα 1,2.

Υπολογίζουμε τώρα τα φράγματα του μέσου μήκους κώδικα για τις δύο πηγές των ερωτημάτων **(1)** και **(2)** με N = 1, 2 αντίστοιχα από τη σχέση **(V)**:

- **N = 1**: $H(X) \leq L_1 < H(X) + 1$ => $\mathbf{3.7831 \leq L_1 < 4.7831}$
- **N = 2**: $H(X^2) \leq L_2 < H(X^2) + 1$ => $\mathbf{7.5662 \leq L_2 < 8.5662}$

Ο λόγος συμπίεσης (compression ratio) αναφέρεται στο λόγο μεταξύ του μεγέθους της πληροφορίας μετά τη κωδικοποίηση και του μεγέθους της αρχικής πληροφορίας. Χρησιμοποιείται ως εναλλακτικό μέτρο αποδοτικότητας της συμπίεσης και αν είναι μικρότερος του 1 σημαίνει ότι τα δεδομένα μετά τη συμπίεση έχουν μετατραπεί σε μια πιο συμπαγή μορφή σε σχέση με πριν τη συμπίεση, το οποίο είναι και το επιθυμητό για τη επίτευξη της μείωσης του αποθηκευτικού χώρου. Δίνεται από τη σχέση:

$$J = \frac{\#\text{bits Huffman encoding}}{\#\text{bits binary representation}} \qquad \textbf{(VI)}$$

(4) Κωδικοποιήστε την πηγή χρησιμοποιώντας των κώδικα Huffman που υπολογίσατε στο Ερώτημα 1. Επιβεβαιώστε, την ορθότητα της κωδικοποίησης σας (π.χ. μέσω της αντίστροφης διαδικασίας – αποκωδικοποίησης). Συγκρίνετε τον αριθμό bits της δυαδικής αναπαράστασης της εικόνας σε σχέση με την κωδικοποίηση Huffman υπολογίζοντας τον λόγο συμπίεσης.

Αρχικά κωδικοποιούμε την 1ης τάξης πηγή μας μέσω της συνάρτησης **[code] = huffmanenco(sig,dict)** πάνω στο *image_vector* με βάση το λεξικό που κατασκευάσαμε πριν. Έπειτα, εκτελούμε την αντίστροφη διαδικασία και ανακατασκευάζουμε την αρχική εικόνα μέσω αποκωδικοποίησης για να βεβαιωθούμε ότι όλα τρέχουν ομαλά. Αυτό γίνεται εύκολα αν εφαρμόσουμε την συνάρτηση [**sig] = huffmandeco(code,dict)** πάνω στο encoded_image και αποθηκεύσουμε το αποκωδικοποιημένο διάνυσμα στο *decompressed_image_vector* και μετά μέσω της **[B] = reshape(A,sz)** μετατρέψουμε το διάνυσμα αυτό που προκύπτει από την αποκωδικοποίηση πάλι σε εικόνα decompressed_image.

```
encoded_image = huffmanenco(image_vector, huffman_dict);
% Check if Huffman-encoded image is correct
decompressed_image_vector = huffmandeco(encoded_image, huffman_dict);
decompressed_image = reshape(decompressed_image_vector, size(image));
imshow(decompressed_image);
```

Εκτελούμε τον κώδικα και παρατηρούμε τις δύο εικόνες που προκύπτουν:

Συμπέρασμα

Παρατηρούμε ότι η αρχική και η τελική εικόνα ταυτίζονται και συνεπώς έχουμε κάνει βέλτιστη κωδικοποίηση Huffman και άρα και ανακατασκευή.

Εικόνα 1: Κωδικοποίηση **Εικόνα 2**: Αποκωδικοποίηση

Προχωρώντας τώρα στον υπολογισμό του λόγου συμπίεσης πάλι μέσω της **[n] = numel(___)** παίρνουμε το πλήθος των δυαδικών ψηφίων της encoded_image και του *image_vector*, πάνω στο οποίο καλούμε βέβαια και την [**binStr] = dec2bin(D)** ώστε να πάρουμε τα δυαδικά ψηφία δυαδικής αναπαράστασης της αρχικής. Έπειτα, διαιρούμε τα δύο πλήθη και παίρνουμε το τελικό αποτέλεσμα.

```
% Calculate compression ratio J
j = numel(encoded_image) / numel(dec2bin(image_vector));
disp('Compression Ratio:');
disp(num2str(j));
```

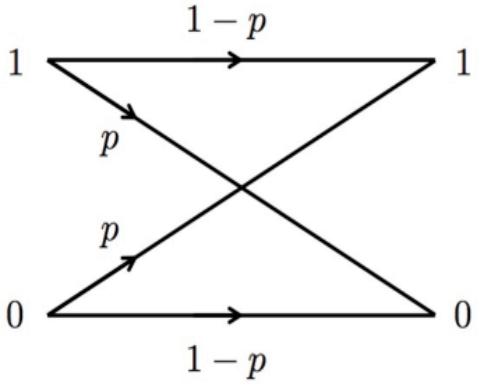
Πράγματι, το τελικό αποτέλεσμα που βλέπουμε στην οθόνη μας είναι το εξής:

```
Compression Ratio:
0.47967
```

Συμπέρασμα

Βλέπουμε ότι ο λόγος συμπίεσης είναι μικρότερος της μονάδας και συγκεκριμένα σε ποσοστιαία μορφή ισούται με 47.97%, δηλαδή τα τελικά δεδομένα μετά τη συμπίεση έχουν σχεδόν το μισό μέγεθος από τα αρχικά και άρα επιτυγχάνουμε καλύτερη συμπίεση από τη δυαδική αναπαράσταση.

Το Δυαδικό Συμμετρικό Κανάλι (Binary Symmetric Channel) αποτελεί μια απλοποιημένη μορφή καναλιού επικοινωνίας ανάμεσα σε πομπό και δέκτη, το οποίο χρησιμοποιεί δυαδικά αλφάβητα {0, 1} για είσοδο και έξοδο. Το κύριο χαρακτηριστικό του καναλιού είναι ότι κατά τη μετάδοση το δυαδικό ψηφίο μπορεί να μετατραπεί στο συμπληρωματικό του (0 σε 1 ή 1 σε 0) με ίδια πιθανότητα σφάλματος p. Η συμμετρία, τέλος, προκύπτει από το γεγονός ότι το κανάλι αντιμετωπίζει ισότιμα τις τιμές '0' και '1' καθώς η παραμόρφωση του ενός συμβόλου δεν επηρεάζει τη μετάδοση των υπολοίπων.



Το κανάλι λαμβάνει στη συγκεκριμένη περίπτωση ως είσοδο την κωδικοποιημένη ακολουθία Huffman από το προηγούμενο ερώτημα την οποία θέλουμε τώρα να τη μεταδώσουμε στον δέκτη. Η μοντελοποίηση του καναλιού BSC επίσης δίνεται από την έτοιμη συνάρτηση **[y] = binary_symmetric_channel(x)**, η οποία υπολογίζει κατευθείαν την ακολουθία της εξόδου που λαμβάνει ο δέκτης. Σκοπός μας λοιπόν είναι να εκτιμήσουμε τη πιθανότητα σφάλματος μετάδοσης p.

(5) Επιθυμούμε τώρα να μεταδώσουμε την κωδικοποιημένη ακολουθία που προέκυψε μέσα από ένα δυαδικό συμμετρικό κανάλι για το οποίο δεν γνωρίζουμε την πιθανότητα p σωστής μετάβασης. Πιο συγκεκριμένα, θεωρούμε την συνάρτηση «$y = binary\_symmetric\_channel(x)$» η όποια μοντελοποιεί το κανάλι και λαμβάνει ως είσοδο το διάνυσμα του κώδικα σε bit (x) και παράγει την αντίστοιχη ακολουθία από bits που παρατηρεί ο δέκτης (y). Χρησιμοποιώντας τις δυο παραπάνω ακολουθίες bits (x, y) να εκτιμήσετε την παράμετρο p (σε ακρίβεια 2 δεκαδικών ψηφίων) και να υπολογίσετε την χωρητικότητα του καναλιού. Υπολογίστε την αμοιβαία πληροφορία αναμεσά στην είσοδο και την έξοδο του καναλιού.

Για να εφαρμόσουμε όλο αυτό το σκεπτικό σε κώδικα MATLAB αρχικά καλούμε τη συνάρτηση που περιγράψαμε πριν πάνω στο encoded_image και αποθηκεύουμε την ακολουθία εξόδου στη received_sequence. Στη συνέχεια, ελέγχουμε ανάμεσα στις δύο ακολουθίες για κάθε ένα δυαδικό ψηφίο ξεχωριστά αν είναι το ίδιο και στις δύο ή το συμπληρωματικό του και μέσω της **[S] = sum(A)** αποθηκεύουμε στην μεταβλητή differing_bits το άθροισμα όλων των περιπτώσεων όπου γίνεται toggle κατά τη μετάδοση. Άρα, η πιθανότητα σφάλματος θα είναι το πλήθος των διαφορετικών δυαδικών ψηφίων προς το μήκος της κωδικοποιημένης εικόνας που λαμβάνεται ως είσοδος.

```
%Binary Symmetrical Channel
% Calculate error probability (p) of BSC
received_sequence = binary_symmetric_channel(encoded_image);
differing_bits = sum(encoded_image ~= received_sequence); % Count the number of differing bits between the two sequences
p = differing_bits / length(encoded_image);
disp('Estimated Error Probability:');
fprintf('%.2f\n', p);
```

Πράγματι, το τελικό αποτέλεσμα που βλέπουμε στην οθόνη μας είναι το εξής:

```
Estimated Error Probability:
0.12
```

Η χωρητικότητα του καναλιού καθορίζει τη μέγιστη τιμή της αμοιβαίας πληροφορίας I(X; Y) ως προς όλες τις δυνατές κατανομές του αλφαβήτου εισόδου X, δηλαδή του αλφαβήτου της κωδικοποιημένης ακολουθίας Huffman. Για να υπολογίσουμε τη χωρητικότητα του καναλιού προσπαθούμε να μεγιστοποιήσουμε τη ποσότητα I(X; Y) ως προς τις πιθανότητες $p(X_j)$ και συγκεκριμένα στο BSC ορίζεται ως:

$$\mathbf{C(p) = 1 - H_b(p)} \qquad \mathbf{(VII)}$$

όπου: $H_b(p) = -p\log_2 p - (1 - p)\log_2(1 - p)$ είναι η δυαδική εντροπία.

Αυτή τη διαδικασία εφαρμόζουμε και στον κώδικα παρακάτω:

```
% Calculate capacity of BSC
% Calculate output entropy H(Y)
output_entropy = -p * log2(p) - (1 - p) * log2(1 - p);
bsc_capacity = 1 - output_entropy;
disp('BSC Capacity:');
disp(num2str(bsc_capacity));
```

Πράγματι, το τελικό αποτέλεσμα που βλέπουμε στην οθόνη μας είναι το εξής:

```
BSC Capacity:
0.4717
```

Όσον αφορά τώρα την αμοιβαία πληροφορία I(X; Y) γνωρίζουμε ότι είναι το μέτρο της ποσότητας πληροφορίας για κάθε σύμβολο που μεταβιβάζεται από το πομπό στο δέκτη του καναλιού, δηλαδή μια μετρική του πόσο καλά μπορεί η έξοδος να προβλέψει την είσοδο του καναλιού. Υπολογίζεται από τον τύπο:

$$\mathbf{I(X; Y) = H(Y) - H(Y|X)} \qquad \mathbf{(VIII)}$$

όπου: H(Y) είναι η εντροπία της εξόδου και εξαρτάται από τη πιθανότητα σφάλματος,

H(Y|X) είναι η εντροπία της εξόδου δεδομένης της εισόδου (υπό συνθήκη εντροπία).

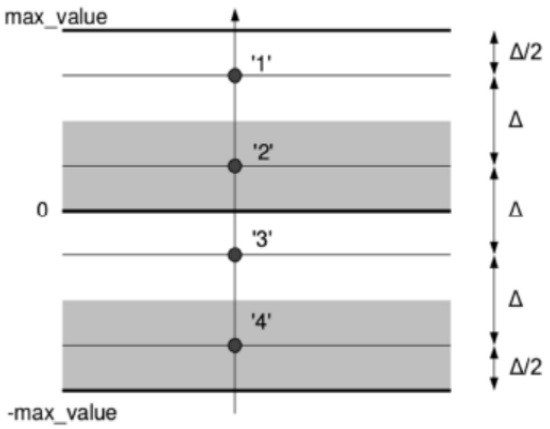
Αυτή τη διαδικασία εφαρμόζουμε και στον κώδικα παρακάτω όπου αρχικά υπολογίζουμε την υπό συνθήκη εντροπία βρίσκοντας πρώτα τις πιθανότητες το δυαδικό ψηφίο στην ακολουθία εισόδου να είναι 0 και 1 αντίστοιχα. Βρίσκουμε έτσι το πλήθος των δυαδικών ψηφίων που είναι 0 και το διαιρούμε με το συνολικό μήκος της ακολουθίας αποθηκεύοντας το αποτέλεσμα στη p_x0. Η πιθανότητα το δυαδικό ψηφίο να είναι 1 είναι συμπληρωματική της προηγούμενης πιθανότητας άρα στη p_x1 αποθηκεύουμε τη τιμή 1 - p_x0. Για τη δεσμευμένη πιθανότητα το δυαδικό ψηφίο εξόδου να είναι 0 αν της εισόδου είναι επίσης 0 διαιρούμε το πλήθος των δυαδικών ψηφίων στην ακολουθία εξόδου που είναι 0 για τα οποία ισχύει ότι και τα δυαδικά ψηφία της εισόδου είναι 0 με το πλήθος των δυαδικών ψηφίων της εισόδου που είναι 0. Την αντίστοιχη διαδικασία ακολουθούμε και για τη περίπτωση που το δυαδικό ψηφίο είναι 1. Τέλος, υπολογίζουμε την αμοιβαία πληροφορία αφαιρώντας από την εντροπία εξόδου που υπολογίσαμε πιο πάνω την υπό συνθήκη εντροπία.

```
% Calculate joint information
% Calculate conditional entropy H(Y|X)
p_x0 = sum(encoded_image == 0) / length(encoded_image);
p_x1 = 1 - p_x0;
p_y_given_x0 = sum(received_sequence(encoded_image == 0) == 0) / sum(encoded_image == 0);
p_y_given_x1 = sum(received_sequence(encoded_image == 1) == 1) / sum(encoded_image == 1);
conditional_entropy = - (p_x0 * p_y_given_x0 * log2(p_y_given_x0) + p_x1 * p_y_given_x1 * log2(p_y_given_x1));
joint_information = output_entropy - conditional_entropy;
disp('Joint Information:');
disp(num2str(joint_information));
```

Πράγματι, το τελικό αποτέλεσμα που βλέπουμε στην οθόνη μας είναι το εξής:

```
Joint Information:
0.36647
```

ΜΕΡΟΣ Β – ΚΩΔΙΚΟΠΟΙΗΣΗ ΔΙΑΚΡΙΤΗΣ ΠΗΓΗΣ ΜΕ ΜΕΘΟΔΟ DPCM



Για την υλοποίηση της συγκεκριμένης άσκησης θα χρειαστεί να υλοποιήσουμε έναν ομοιόμορφο κβαντιστή 2^N επιπέδων, ο οποίος θα δέχεται στην είσοδό του ένα σήμα και κβαντίζοντας κάθε δείγμα του θα το περιορίζει σε ένα προκαθορισμένο διάστημα [min_value, max_value] δυναμικής περιοχής. Ο κβαντιστής θα υπολογίζει αρχικά το βήμα δέλτα (Δ) της κβάντισης, το οποίο θα είναι το μήκος του διαστήματος της δυναμικής περιοχής δια το πλήθος των επιπέδων κβάντισης που κυμαίνονται από 1 ως 2^N. Έπειτα, υπολογίζουμε τα κέντρα των επιπέδων, τα οποία σύμφωνα με το διπλανό σχήμα θα είναι τοποθετημένα στο διάστημα [**min_value +** $\frac{\Delta}{2}$, min_value + $\frac{3\Delta}{2}$, …, **max_value –** $\frac{\Delta}{2}$].

Τέλος, για κάθε δείγμα από το σήμα εισόδου, αφού το θέσουμε στην αντίστοιχη ακραία αποδεκτή τιμή αν βρίσκεται εκτός δυναμικής περιοχής, υπολογίζουμε σε ποια περιοχή κβάντισης ανήκει και το αντιστοιχίζουμε στο κατάλληλο κέντρο. Το τελικό διάνυσμα που θα προκύψει από όλες αυτές τις τιμές θα είναι το κβαντισμένο σήμα εισόδου.

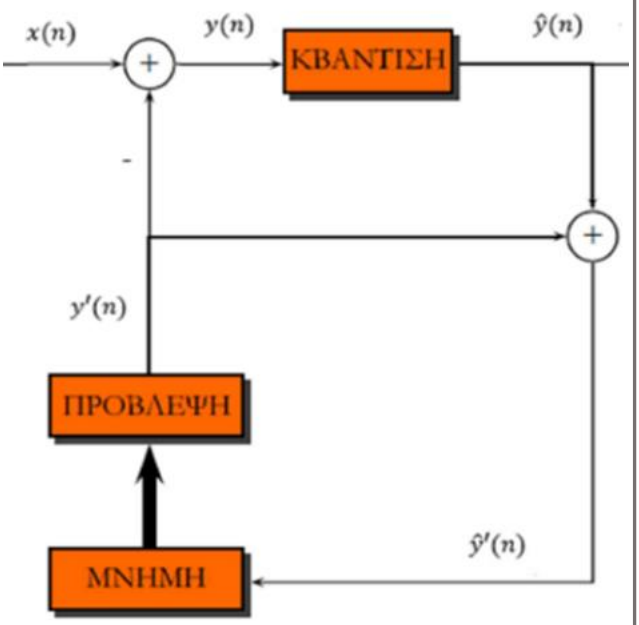
Για να υλοποιήσουμε το κώδικα του κβαντιστή αρχικά ορίζουμε τη συνάρτηση **[y_hat] = my_quantizer(y, N, min_value, max_value)**, η οποία αποθηκεύει στο διάνυσμα *y_hat* το τελικό κβαντισμένο διάνυσμα *y* που λαμβάνει σαν παράμετρο, μαζί με τα δυαδικά ψηφία κβάντισης, την ελάχιστη και τη μέγιστη τιμή της δυναμικής περιοχής. Πρώτα αποθηκεύουμε στη μεταβλητή delta το βήμα κβαντισμού και μετά στο διάνυσμα *centers* όλα τα κέντρα των επιπέδων. Ο τρόπος με τον οποίο αρχικοποιούμε την αρχική και τη τελική τιμή του *centers* είναι με βάση την πιο πάνω περιγραφή και για όλα τα ενδιάμεσα σημεία, επειδή μεταβαίνουμε από την μέγιστη τιμή κέντρου προς την ελάχιστη (στο σχήμα μας δηλαδή με φορά από πάνω προς τα κάτω), θα έχουμε αρνητικό βήμα μεγέθους delta. Έπειτα, «κόβουμε» τις εκτός εύρους τιμές για κάθε δείγμα του *y*. Τέλος, στο προκύπτον αυτό σήμα για κάθε δείγμα του υπολογίζεται η ποσότητα του αντίστοιχου κβαντιστικού επιπέδου και στη συνέχεια ελέγχεται σε ποιο από όλα τα επίπεδα ανήκει. Αν το δείγμα πράγματι ανήκει στο διάστημα [*centers*(i) – delta/2, *centers*(i) – delta/2], τότε το κβαντισμένο δείγμα του *y_hat* θα είναι το κέντρο αυτού του κβαντιστικού επιπέδου.

```
function [y_hat] = my_quantizer(y, N, min_value, max_value)

   % Quantization step
   delta = (max_value - min_value)/ 2^N;

   % Calculation of centers of quantization areas
   centers = zeros(2^N, 1);
   centers(1) = max_value - delta/2;
   centers(2^N) = min_value + delta/2;
   for i = 2:2^N - 1
      centers(i) = centers(i - 1) - delta;
   end

   % Process each element in y
   % Limiting the dynamic range of the current element
```

```
    if y < min_value
        y = min_value;
    end
    if y > max_value
        y = max_value;
    end

    %Find the area of each input sample
    for i = 1: 2^N
        if((y <= centers(i) + delta/2) && (y >= centers(i) - delta/2))
            %Create quantized input signal
            y_hat = centers(i);
        end
    end
end
```

(1) Να υλοποιήσετε το παραπάνω σύστημα κωδικοποίησης/αποκωδικοποίησης DPCM.

Ο κωδικοποιητής DPCM χρησιμοποιείται για την κβάντιση αλλά και την κωδικοποίηση της τρέχουσας τιμής του ηχητικού σήματος προκειμένου να αποσταλεί από το πομπό στο δέκτη για να ανακατασκευαστεί. Ωστόσο, ο κωδικοποιητής δεν χρησιμοποιεί τη πραγματική τιμή του σήματος εισόδου στην κβάντιση αλλά προβλέπει την επόμενη τιμή σειράς δειγμάτων βασιζόμενος στις προηγούμενες κωδικοποιημένες τιμές. Πιο συγκεκριμένα, προσεγίζουμε το σφάλμα κωδικοποίησης υπολογίζοντας τη διαφορά ανάμεσα στην πραγματική και την προβλεπόμενη τιμή του σήματος εισόδου και το σήμα σφάλματος είναι αυτό που εν τέλει κβαντίζεται και αποστέλλεται στο δέκτη. Άρα, Ο DPCM κωδικοποιητής παράγει σήματα κβαντισμένων διαφορών, τα οποία στη συνέχεια κωδικοποιούνται με τη χρήση κβαντιστών με σκοπό τη μείωση του αριθμού των απαιτούμενων δυαδικών ψηφίων για την αποθήκευση.



Για την πρόβλεψη του σήματος εισόδου έχουμε ως γνωστές τις p πρώτες κωδικοποιημένες τιμές του, οι οποίες δηλώνουν και την τάξη του φίλτρου πρόβλεψης. Όσο μεγαλύτερη είναι η τιμή αυτής της παραμέτρου, τόσο ακριβέστερη πρόβλεψη θα έχουμε στο τέλος. Η τελική πρόβλεψη ωστόσο θα προκύψει από τον γραμμικό συνδυασμό των p γνωστών τιμών και πιο συγκεκριμένα αν $\hat{y}'(n)$ είναι η πρόβλεψη, τότε θα δίνεται από τη σχέση:

$$\hat{y}'(n) = \sum_{i=1}^{p} a_i \hat{y}'(n-i) \qquad \textbf{(IX)}$$

όπου a_i είναι οι συντελεστές κβάντισης.

Οι συντελεστές κβάντισης είναι το διάνυσμα *a* διάστασης px1 που προκύπτει λύνοντας το σύστημα εξισώσεων Yule – Walker $\mathbf{Ra = r \Rightarrow a = R^{-1}r}$ με R να είναι το μητρώο αυτοσυσχέτισης pxp με R(i, j) = E[x(n – i)x(n – j)] και *r* το αντίστοιχο px1 διάνυσμα αυτοσυσχέτισης. Μετά τον υπολογισμό

τους οι συντελεστές αποστέλλονται και στον δέκτη αφού πρώτα κβαντιστούν και οι ίδιοι για να λειτουργούν και οι δύο πλευρές σε συμφωνία.

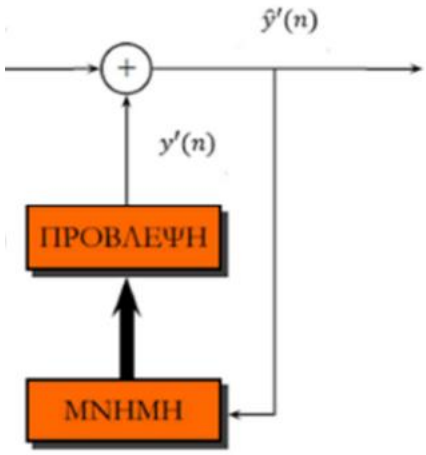
Αυτή η διαδικασία περιγράφεται από τον πιο κάτω κώδικα MATLAB:

```
% Question 1
function [a_quantized, y, y_hat, y_hat_prediction] = dpcm_encoder(x, p, quantization_bits,
quantization_min_value, quantization_max_value)
    N = length(x);
    r = zeros(p, 1);
    R = zeros(p, p);
    a_quantized = zeros(p, 1);

    % Create autocorrelation matrix R and autocorrelation vector r
    for i = 1:p
        for j = 1:p
            sum = 0;
            for n = p+1:N
                sum = sum + x(n - j) * x(n - i);
            end
            R(i, j) = sum / (N - p);
        end
    end

    for k = 1:p
        sum = 0;
        for n = p+1:N
            sum = sum + x(n) * x(n - k);
        end
        r(k) = sum / (N - p);
    end

    % Quantize coefficients a with N=8 bits and dynamic range [-2,2]
    a = R\r;
    for i = 1:p
        a_quantized(i) = my_quantizer(a(i), 8, -2, 2);
    end

    % Initialize y(n), y_hat_prediction(n) and y_hat(n)
    y = zeros(N, 1);
    y_hat = zeros(N, 1);
    y_hat_prediction = 0;
    mem = zeros(p, 1);

    % Calculate all the values from 1 to N
    for i = 1:N
        % Calculate the error and quantize it
        y(i) = x(i) - y_hat_prediction;
```

```
        y_hat(i) = my_quantizer(y(i), quantization_bits, quantization_min_value, quantization_max_value);
        % Update the y_hat_prediction to include the quantized error (reconstruction)
        y_hat_prediction = y_hat_prediction + y_hat(i);
        % Calculate the prediction y_hat'(n) using previous p y_hat_prediction values
        mem = [y_hat_prediction; mem(1:p -1)];
        y_hat_prediction = a_quantized.' * mem;
    end
end
```

Αρχικά, ορίζουμε τα μητρώο R, το διάνυσμα *r* και το διάνυσμα *a_quantized* με τις αντίστοιχες διαστάσεις. Για την κατασκευή του R υπολογίζουμε για κάθε κελί (i, j) του πίνακα το άθροισμα των γινομένων των N – p επόμενων τιμών του διανύσματος *x* ολισθημένες κατά i και j χρονικές στιγμές αντίστοιχα. Για τη κατασκευή του *r* όμοια υπολογίζουμε για κάθε κελί του k το άθροισμα των γινομένων των N – p επόμενων τιμών του διανύσματος *x* ολισθημένες κατά 0 και k χρονικές στιγμές αντίστοιχα. Τόσο το μητρώο όσο και το διάνυσμα διαιρούνται με τη ποσότητα N – p για να προκύψει η μέση τιμή της αυτοσυσχέτισης που είναι και το ζητούμενο με βάση τη παραπάνω περιγραφή. Μετά αποθηκεύουμε στο διάνυσμα *a* το γινόμενο $R^{-1}r$ και αφού το κβαντίσουμε με N = 8 στο διάστημα [-2, 2] μέσω της συνάρτησης **[ ] = my_quantizer(___)** που κατασκευάσαμε πριν το αποθηκεύουμε στο *a_quantized*. Ακόμα, αρχικοποιούμε τα διανύσματα *y*, *y_hat*, *y_hat_prediction* και *mem* που αντιπροσωπεύουν το σήμα σφάλματος, το κβαντισμένο σήμα σφάλματος, την πρόβλεψη του σήματος *x* και τη μνήμη αντίστοιχα. Έπειτα, υπολογίζουμε για τη τρέχουσα τιμή του σήματος *x* το σφάλμα πρόβλεψης *y* και το κβαντίζουμε πάλι μέσω της **[ ] = my_quantizer(___)** αποθηκεύοντάς το στο *y_hat*. Στη συνέχεια, ανανεώνουμε την προβλεπόμενη τιμή για το επόμενο βήμα, προσθέτοντας την κβαντισμένη τιμή του σφάλματος στην τρέχουσα πρόβλεψη *y_hat_prediction* και ανανεώνουμε και το διάνυσμα μνήμης *mem* για το φίλτρο πρόβλεψης. Η νέα τιμή του *y_hat_prediction* προστίθεται στην αρχή του διανύσματος, ενώ το παλαιότερο στοιχείο αφαιρείται. Τέλος, υπολογίζουμε την επόμενη πρόβλεψη με το εσωτερικό γινόμενο του *a_quantized* με το ενημερωμένο διάνυσμα μνήμης, σύμφωνα με τον τύπο **(IX)**. Έτσι ολοκληρώνεται η υλοποίηση του κωδικοποιητή.

Ο DPCM αποκωδικοποιητής λαμβάνει το κβαντισμένο σήμα σφάλματος από τον πομπό και αναλαμβάνει να ανακατασκευάσει το αρχικό σήμα με βάση την ακρίβεια της προσέγγισής του, γεγονός το οποίο σημαίνει ότι το σφάλμα πρόβλεψης πρέπει να είναι μικρό. Όπως αναφέρθηκε και πριν μαζί με το κβαντισμένο σφάλμα αποστέλλονται και οι κβαντισμένοι συντελεστές, συνεπώς ο αποκωδικοποιητής για την πρόβλεψη του ανακατασκευασμένου σήματος εισόδου θα εκτελεί ακριβώς το ίδιο σκεπτικό με τον κωδικοποιητή και θα μπορεί να κάνει ακριβώς την ίδια πρόβλεψη. Έτσι, αφού υπολογιστεί η πρόβλεψη συνδυάζεται με το κβαντισμένο σφάλμα και δίνει σαν τελικό αποτέλεσμα το ανακατασκευασμένο σήμα, μειώνοντας παράλληλα σημαντικά τη συσσώρευση του σφάλματος.



Σε κώδικα MATLAB εκτελούμε τις εξής εντολές:

```
% Question 1
function [x_reconstructed] = dpcm_decoder(y_hat, p, a_quantized)
    N = length(y_hat);
    x_reconstructed = zeros(N, 1);
    mem = zeros(p, 1);
    y_hat_prediction = 0;

    % Calculate all the values from 1 to N
    for i = 1:N
        % Add the prediction to the quantized error to reconstruct x
        x_reconstructed(i) = y_hat_prediction + y_hat(i);
        % Calculate the prediction y_hat_prediction(n) using the quantized coefficients a_quantized
        % and the previously reconstructed values
        mem = [x_reconstructed(i); mem(1:p - 1)];
        y_hat_prediction = a_quantized.' * mem;
    end
end
```

Ξεκινάμε αρχικοποιώντας τα διανύσματα *x_recontructed*, *mem* και *y_hat_prediction* που αντιπροσωπεύουν το ανακατασκευασμένο σήμα εισόδου *x*, τη μνήμη και την πρόβλεψη αντίστοιχα. Υπολογίζουμε την τρέχουσα ανακατασκευασμένη τιμή ως άθροισμα της τρέχουσας πρόβλεψης και του τρέχοντος κβαντισμένου σφάλματος πρόβλεψης. Έπειτα, ενημερώνουμε το διάνυσμα μνήμης *mem*, το οποίο χρησιμοποιείται για τον υπολογισμό της επόμενης προβλεπόμενης τιμής. Το διάνυσμα *mem* περιέχει τις τελευταίες p ανακατασκευασμένες τιμές, με την τρέχουσα τιμή *x_reconstructed*(i) να προστίθεται στην αρχή του διανύσματος και το παλαιότερο στοιχείο να αφαιρείται. Η επόμενη πρόβλεψη υπολογίζεται πάλι με βάση το τύπο **(IX)** όπως πριν. Έτσι ολοκληρώνεται η υλοποίηση του αποκωδικοποιητή.

Ξεκινάμε τώρα την επεξεργασία της πηγής μας. Αρχικά, με το παρακάτω κώδικα τη φορτώνουμε, διαβάζουμε το σήμα που περιέχει, το αποθηκεύουμε στο διάνυσμα *x* και τέλος καθορίζουμε και τις σταθερές έτσι όπως μας ζητούνται:

```
data = load('source.mat');
x = data.t;

% Define the constants
min_value = -3.5;
max_value = 3.5;
```

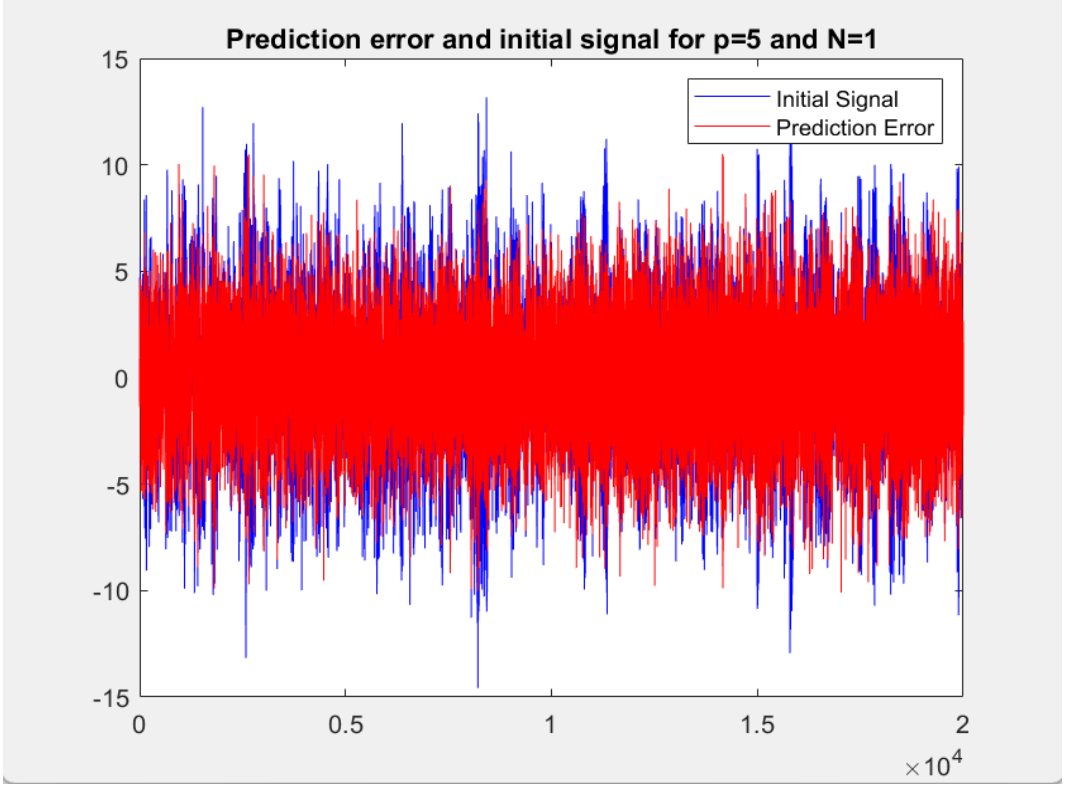
(2) Επιλέξτε δύο τιμές του $p \geq 5$ και για N = 1,2,3bits σχεδιάστε στο ίδιο γράφημα το αρχικό σήμα και το σφάλμα πρόβλεψης y. Σχολιάστε τα αποτελέσματα. Τί παρατηρείτε;

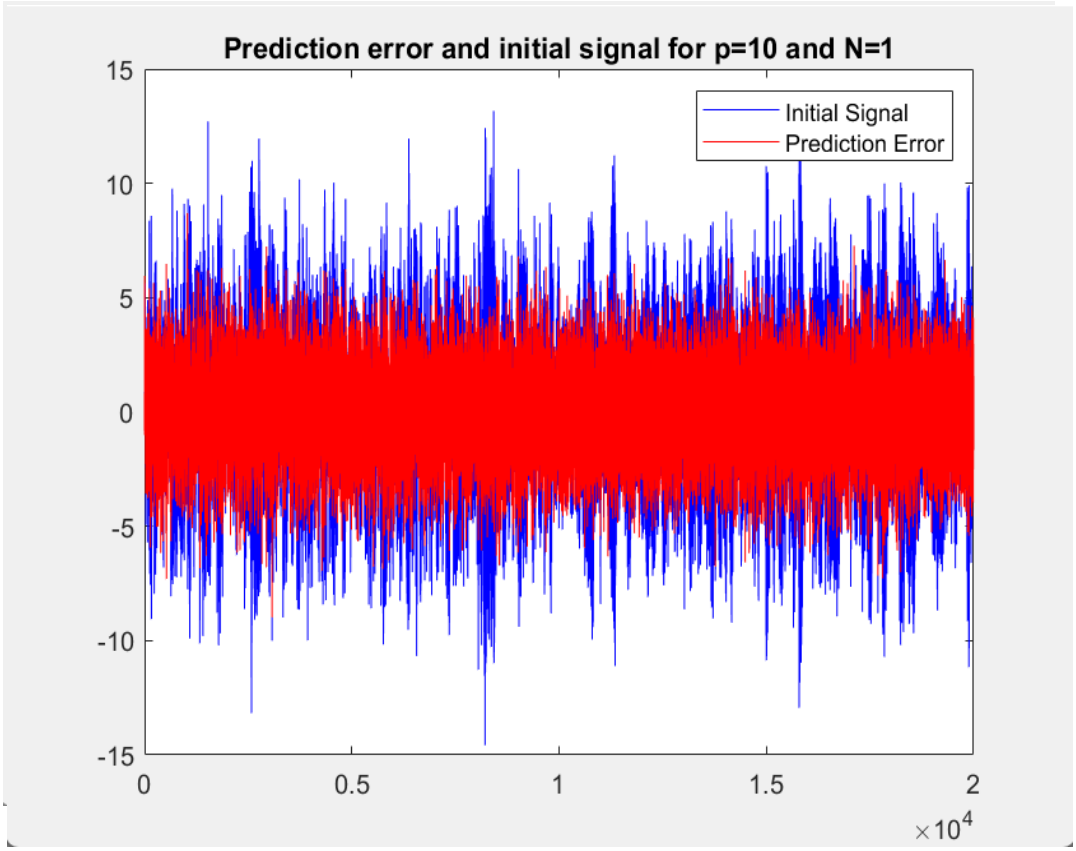
Για τις δύο τιμές της τάξης του φίλτρου πρόβλεψης επιλέγουμε τις τιμές 5 και 10 και για το πλήθος των δυαδικών ψηφίων κβάντισης τις τιμές 1, 2, 3. Πρώτα, καλούμε τη συνάρτηση **[ ] = dpcm_encoder(___)** που υλοποιήσαμε πριν με ορίσματα το διάνυσμα *x* και τις σταθερές p, N, min_value και max_value και αποθηκεύουμε το σήμα σφάλματος στο διάνυσμα *y*. Με τις κατάλληλες συναρτήσεις της MATLAB σχεδιάζουμε στο ίδιο γράφημα το αρχικό σήμα και το σήμα σφάλματος.

```
% Question 2
for p = 5:5:10
   for N = 1:3
      [~, ~, y, ~, ~] = dpcm_encoder(x, p, N, min_value, max_value);

      figure;
      plot(x, 'b');
      hold on;
      plot(y, 'r');
      hold off;

      title(['Prediction error and initial signal for p=', num2str(p), ' and N=', num2str(N)]);
      legend('Initial Signal', 'Prediction Error');
   end
end
```

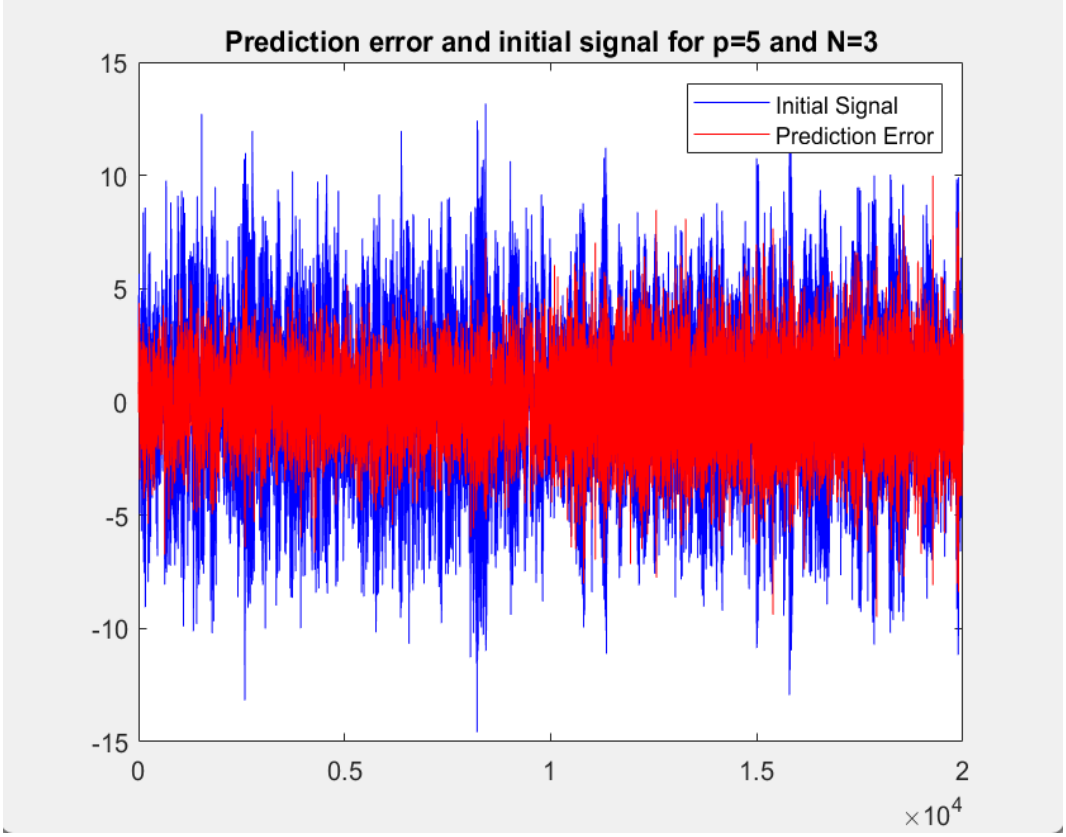
Οι γραφικές παραστάσεις που λαμβάνουμε είναι οι εξής:

Εικόνα 3: p=5, N=1

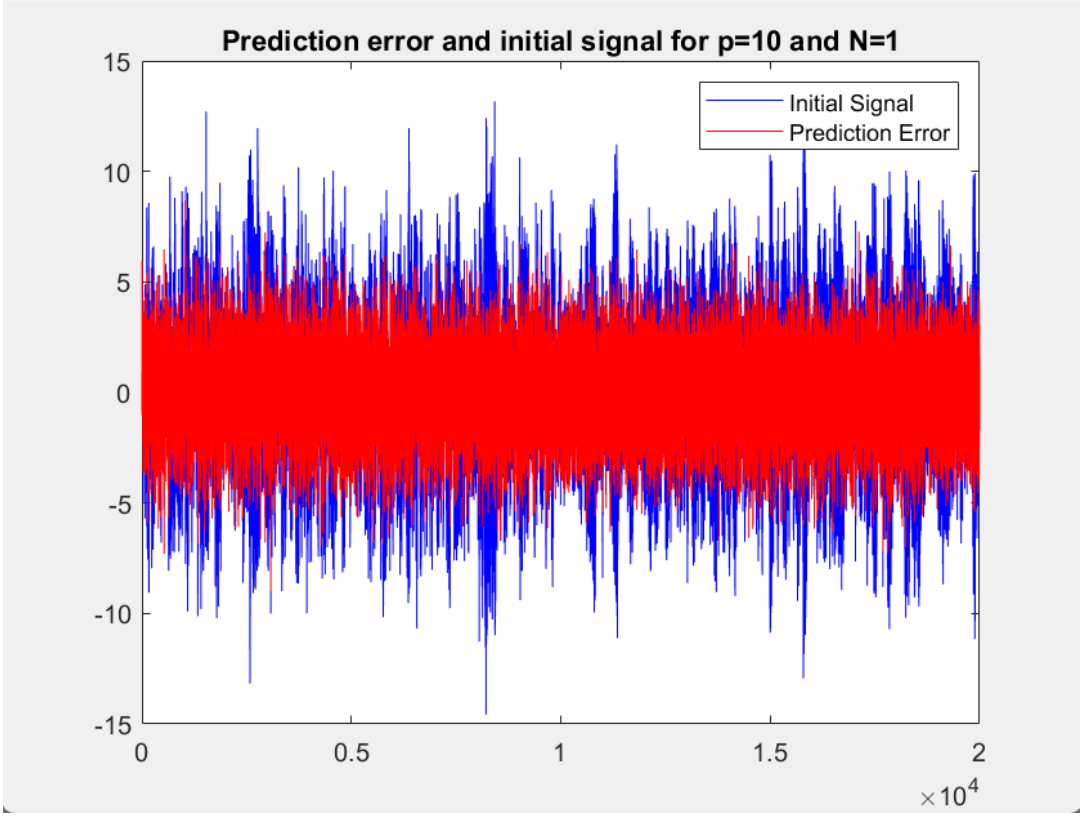




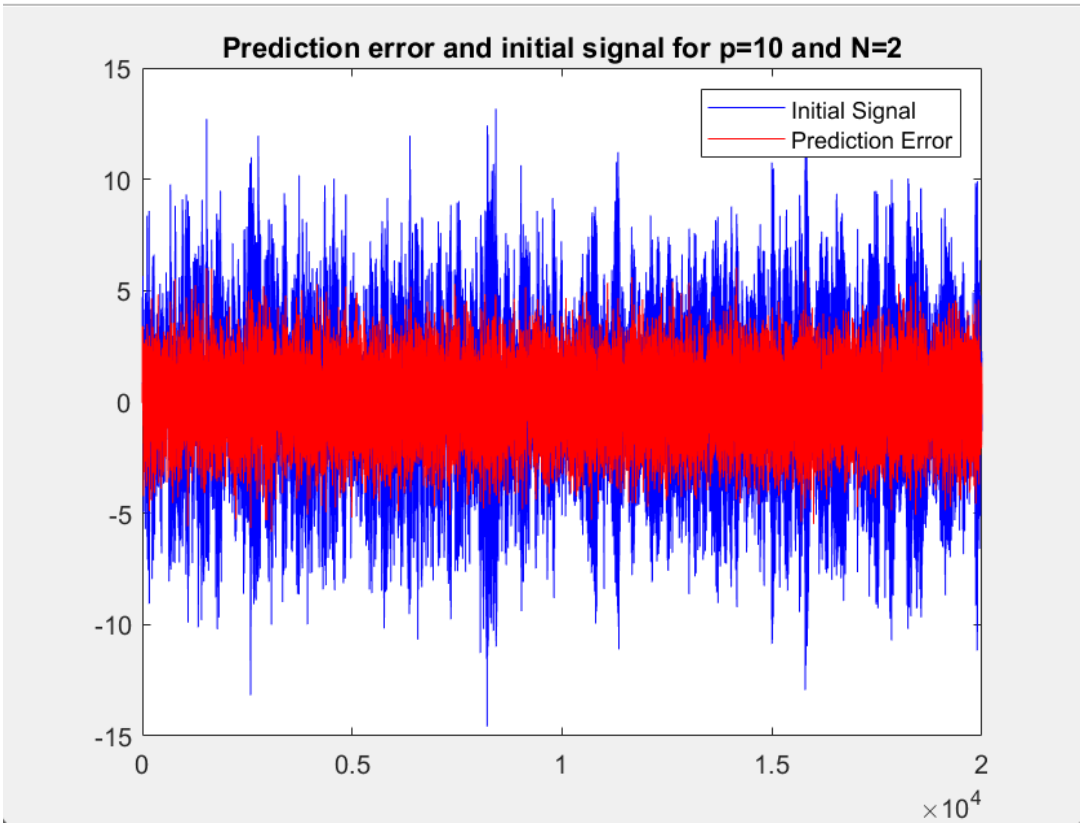
Εικόνα 4: p=5, N=2



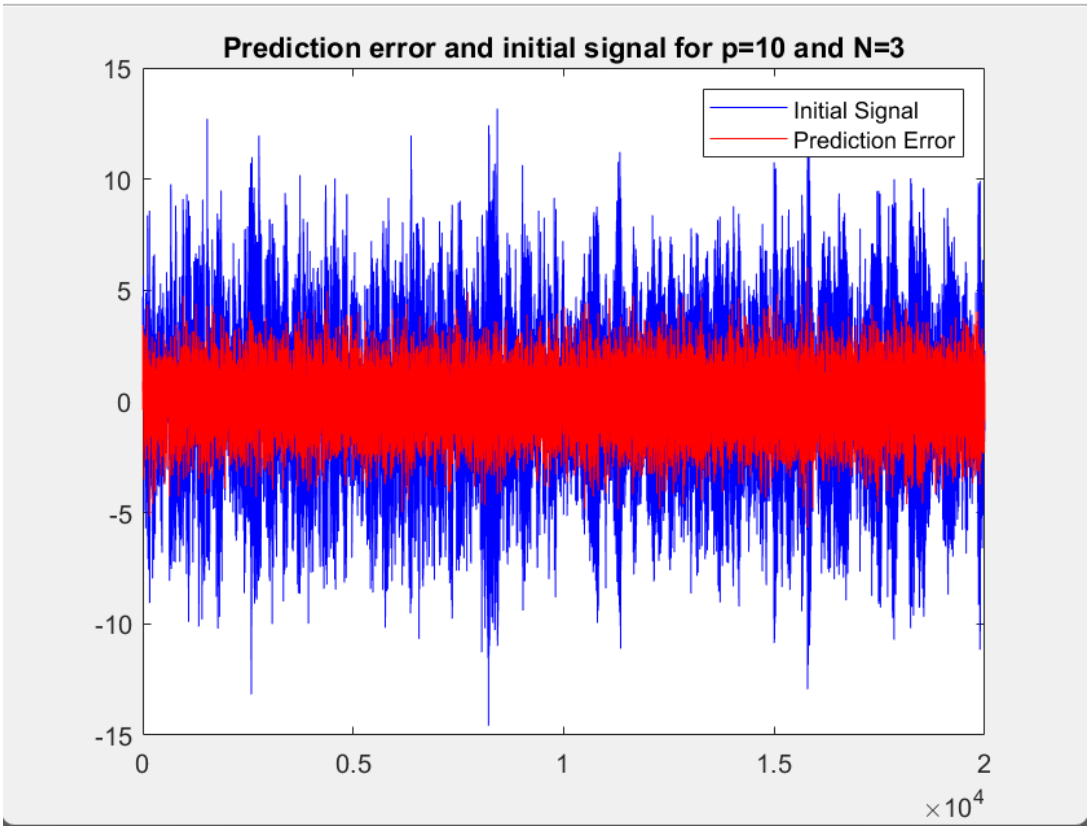
Εικόνα 5: p=5, N=3



Εικόνα 6: p=10, N=1



Εικόνα 7: p=10, N=2



Εικόνα 8: p=10, N=3

Συμπέρασμα

Παρατηρούμε ότι το σφάλμα της πρόβλεψης τείνει να ταυτιστεί με το αρχικό σήμα για τιμές N = 1 και p = 5, 10, γεγονός το οποίο σημαίνει ότι η πρόβλεψη τείνει στο μηδέν, δηλαδή είναι κάκιστη. Αντίθετα, το σφάλμα της πρόβλεψης τείνει στο μηδέν για τιμές N = 3 και p = 5, 10, γεγονός το οποίο σημαίνει ότι η πρόβλεψη μοιάζει πολύ με το αρχικό σήμα, δηλαδή είναι βέλτιστη. Για όλες τις υπόλοιπες ενδιάμεσες τιμές η πρόβλεψη προσεγγίζει μέτρια το αρχικό σήμα. Άρα, βλέπουμε πάλι το καθοριστικό ρόλο που παίζει το πόσα δυαδικά ψηφία θα χρησιμοποιήσουμε κατά την κβάντιση.

Το Μέσο Τετραγωνικό Σφάλμα πρόβλεψης (MSE) αποτελεί μια μετρική της ακρίβειας της πρόβλεψης του σήματος της πηγής, δηλαδή την απόδοση του φίλτρου πρόβλεψης. Το φίλτρο λειτουργεί βέλτιστα όταν επιτυγχάνουμε να ελαχιστοποιήσουμε το MSE, δηλαδή τη διασπορά του τετραγώνου του σήματος σφάλματος γύρω από τη μέση τιμή του. Ένα χαμηλό MSE υποδεικνύει ότι το φίλτρο πρόβλεψης είναι ακριβές, ενώ ένα υψηλό MSE δείχνει ότι υπάρχουν μεγάλες αποκλίσεις μεταξύ των προβλεπόμενων και των πραγματικών τιμών. Δίνεται από τη σχέση:

$$\mathbf{MSE = E[y^2(n)] = E[(x(n) - \hat{y}'(n))^2] = E[(x(n) - \sum_{i=1}^{p} a_i \hat{y}'(n-i))^2]} \qquad \mathbf{(X)}$$

Γενικά, η απόδοση της πρόβλεψης άρα και το σφάλμα επηρεάζονται άμεσα από τη τάξη p του φίλτρου, καθώς όσο μεγαλύτερη είναι η τιμή της τόσο περισσότερα προηγούμενα δείγματα λαμβάνονται υπόψη, επιτρέποντας τη δημιουργία μιας πιο ακριβούς πρόβλεψης. Επιπλέον, σημαντικό ρόλο παίζουν και τα δυαδικά ψηφία της κβάντισης, καθώς η διακριτική ικανότητα είναι υψηλότερη και το πλήθος της πληροφορίας που χάνεται κατά την κβάντιση είναι χαμηλότερο. Αυτό συνεπώς μειώνει το σφάλμα κβάντισης, βελτιώνοντας την ποιότητα του ανακατασκευασμένου σήματος.

(3) Αξιολογείστε την απόδοση του με γράφημα που να δείχνει το Μέσο Τετραγωνικό Σφάλμα πρόβλεψης ως προς το N και για διάφορες τιμές του p. Συγκεκριμένα για πλήθος δυαδικών ψηφίων N = 1,2,3bits τα όποια χρησιμοποιεί ο ομοιόμορφος κβαντιστής για την κωδικοποίηση του σήματος πρόβλεψης και για τάξη προβλεπτή p = 5: 10. Επιπλέον, για κάθε p καταγράψτε στην αναφορά σας και σχολιάστε τις τιμές των συντελεστών του προβλεπτή.

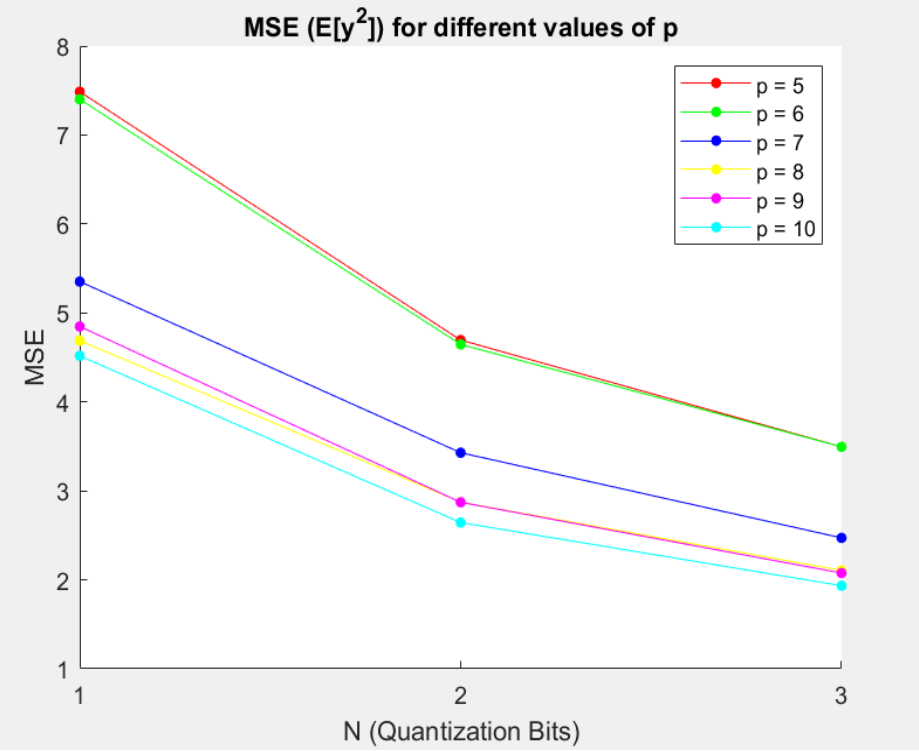
Στο συγκεκριμένο κώδικα MATLAB αρχικά φτιάχνουμε ένα μητρώο mse_matrix 6x3 στο οποίο αποθηκεύουμε τη τιμή του MSE για κάθε συνδυασμό των τιμών N, p που μας ζητείται. Έπειτα, στη μεταβλητή currentMSE αποθηκεύουμε το Μέσο Τετραγωνικό Σφάλμα που προκύπτει αν εφαρμόσουμε τη συνάρτηση **[ ] = mean(___)** στο διάνυσμα *y* σύμφωνα με τη σχέση **(X)**, το οποίο είναι το σήμα σφάλματος που προκύπτει από την **[ ] = dpcm_encoder(___)**. Τέλος, αποθηκεύουμε τη τιμή αυτή στην αντίστοιχη θέση του μητρώου και εκτυπώνουμε τόσο τους συντελεστές κβάντισης για κάθε περίπτωση όσο και τις γραφικές παραστάσεις μέσω κατάλληλων έτοιμων συναρτήσεων.

```
% Question 3
% MSE matrix initialization
mse_matrix = zeros(6, 3);

% Loop over each p and N value
for p = 5:10
   result_string = strcat(' p = ', num2str(p));
   disp(result_string);
   for N = 1:3
      [~, a, y, ~, ~] = dpcm_encoder(x, p, N, min_value, max_value);
      disp(a);
      currentMSE = mean(y.^2);  % Calculate current MSE
      mse_matrix(p - 4, N) = currentMSE;  % Assign to the matrix
   end
end

% Define a set of colors
colors = [1, 0, 0; % Red
          0, 1, 0; % Green
          0, 0, 1; % Blue
```

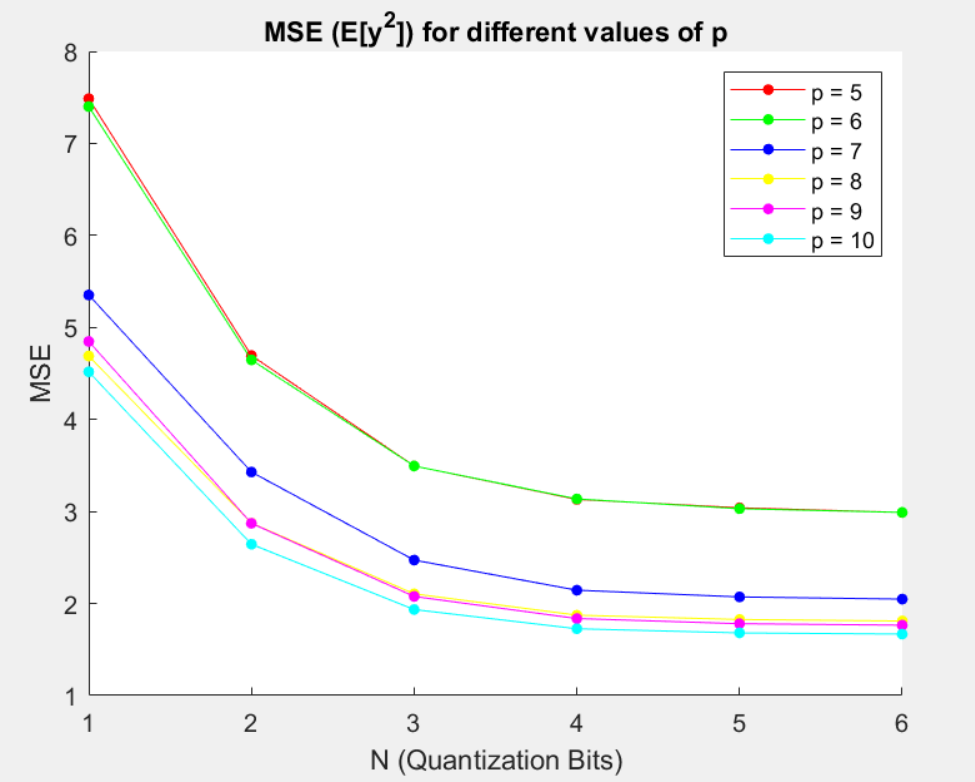
```
        1, 1, 0; % Yellow
        1, 0, 1; % Magenta
        0, 1, 1]; % Cyan

figure;
hold on;

% Loop over each p value
for p = 5:10
    plot(1:3, mse_matrix(p - 4, :), 'Color', colors(p - 4, :), 'Marker', '.', 'MarkerSize', 15);
end

title('MSE (E[y^2]) for different values of p');
xlabel('N (Quantization Bits)');
xticks(1:3);
ylabel('MSE');
legend('p = 5', 'p = 6', 'p = 7', 'p = 8', 'p = 9', 'p = 10');

hold off;
```

Συμπέρασμα

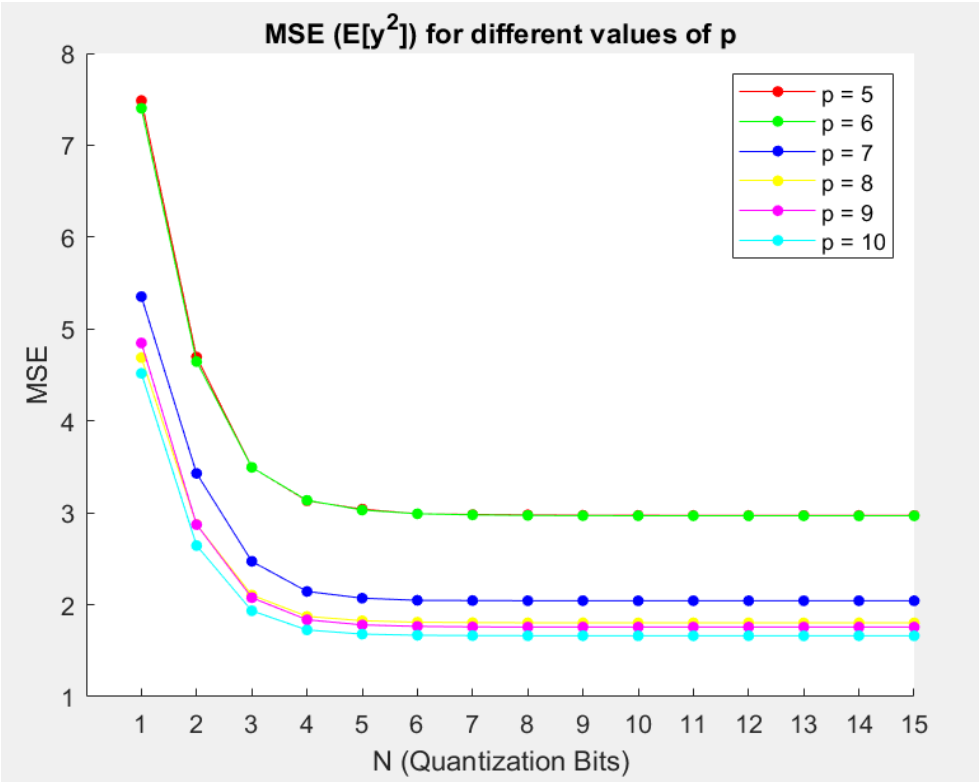
Παρατηρούμε όντως ότι το Μέσο Τετραγωνικό Σφάλμα ακολουθεί μια φθίνουσα πορεία καθώς το πλήθος N των δυαδικών ψηφίων κβάντισης αυξάνεται, ξεκινώντας από τη μέγιστη τιμή του για N = 1 με τη μικρότερη ακρίβεια ανακατασκευής και τείνει ασυμπτωτικά σε μια σταθερή ελάχιστη τιμή για πολύ μεγάλες τιμές του N. Αυτό επιβεβαιώνεται και από τις πιο κάτω γραφικές παραστάσεις όπου μελετάμε και περιπτώσεις με μεγαλύτερη τιμή N από 3.



Εικόνα 9: MSE για N = [1:3]



Εικόνα 10: MSE για N = [1:6]



Εικόνα 11: MSE για N = [1:15]

Το κριτήριο με βάση το οποίο επιλέγουμε τους συντελεστές κβάντισης είναι η ελαχιστοποίηση του Μέσου Τετραγωνικού Σφάλματος. Το παραπάνω κριτήριο ωστόσο, είναι δύσκολο να ελαχιστοποιηθεί αφού το MSE εξαρτάται από τους συντελεστές αλλά και από τον κβαντιστή που χρησιμοποιούμε. Επομένως, αποτελεί ένα μη-γραμμικό πρόβλημα ελαχιστοποίησης. Για να ξεφύγουμε από αυτή τη δυσκολία, αντικαθιστούμε στην σχέση **(IX)** την ποσότητα $\hat{y}'(n-i)$ με το $x(n-i)$ θεωρώντας πως αφού το δεύτερο αποτελεί την κβαντισμένη εκδοχή του πρώτου δεν κάνουμε μεγάλο σφάλμα. Έτσι, οι συντελεστές που προκύπτουν θα πρέπει να διασφαλίζουν τη σταθερότητα του συστήματος. Για το φίλτρο πρόβλεψης αυτό σημαίνει ότι οι πόλοι του φίλτρου πρέπει να βρίσκονται μέσα στη μοναδιαία κυκλική περιοχή στο επίπεδο.

Επιπλέον, για επαλήθευση παρατίθενται πιο κάτω και οι τιμές των συντελεστών κβάντισης του διανύσματος *a_quantized* για κάθε τιμή του παράγοντα p. Βλέπουμε ότι όντως για μεγαλύτερες τιμές του p οι συντελεστές που προκύπτουν οδηγούν σε μειωμένο σφάλμα, άρα το κριτήριο επαληθεύεται.

p =5	**p =6**	**p =7**	**p =8**	**p =9**	**p =10**
1.2852	1.2865	1.2650	1.0876	1.1432	1.1063
-1.5856	-1.5626	-1.5207	-1.3073	-1.2975	-1.1763
0.9901	0.9481	1.1885	0.9403	0.9285	0.8947
-0.5424	-0.4750	-0.9547	-0.6198	-0.5740	-0.5567
-0.0287	-0.0834	0.7069	0.2901	0.1919	0.2361
	0.0425	-0.6081	-0.0751	0.0741	-0.0585
		0.5056	0.0622	-0.1455	0.0694
			0.3505	0.5233	0.2230
				-0.1588	0.1058
					-0.2314

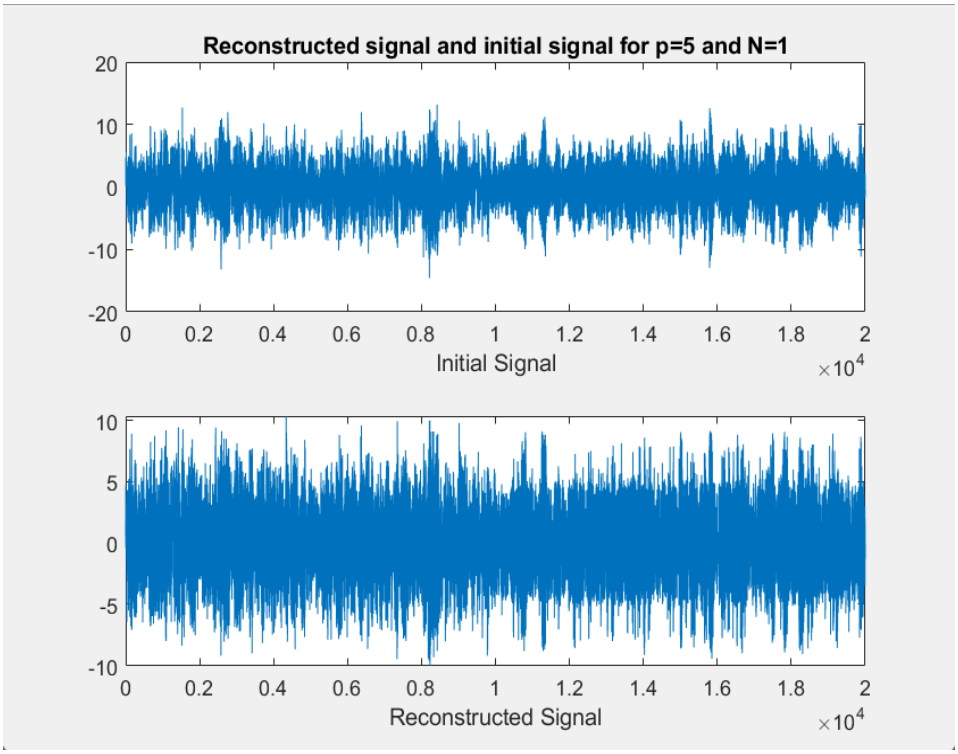
(4) για N = 1,2,3 bits να απεικονισετε το αρχικο και το ανακατασκευασμενο σημα στο δεκτη για p = 5,10 που επιλεξατε και να σχολιασετε τα αποτελεσματα της ανακατασκευης σε σχεση με τα bits κβαντισης.

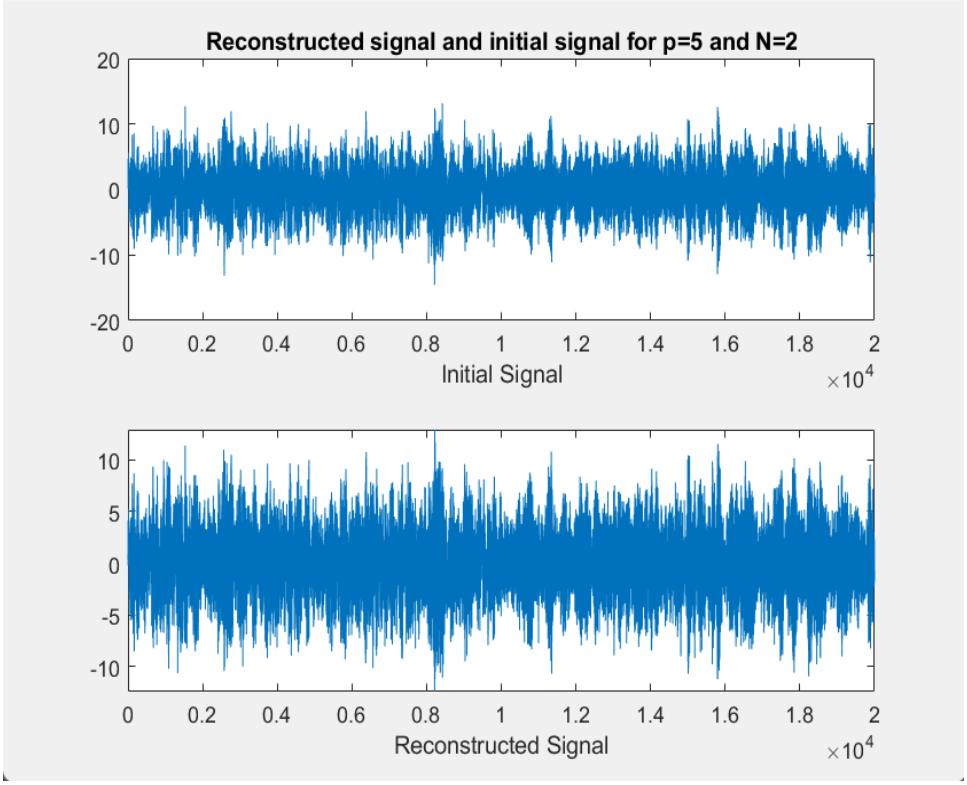
Για τις δύο τιμές της τάξης του φίλτρου πρόβλεψης επιλέγουμε τις τιμές 5 και 10 και για το πλήθος των δυαδικών ψηφίων κβάντισης τις τιμές 1, 2, 3. Πρώτα, καλούμε τη συνάρτηση **[ ] = dpcm_encoder(___)** που υλοποιήσαμε πριν με ορίσματα το διάνυσμα *x* και τις σταθερές p, N, min_value και max_value και αποθηκεύουμε το κβαντισμένο σήμα σφάλματος και τους κβαντισμένους συντελεστές πρόβλεψης στα διανύσματα *y_hat* και *a_quantized* αντίστοιχα. Έπειτα, καλούμε την **[ ] = dpcm_decoder(___)** με ορίσματα τα διανύσματα *y_hat, a_quantized* και την σταθερά p και αποθηκεύουμε το ανακατασκευασμένο σήμα στο διάνυσμα *x_reconstructed*. Με τις κατάλληλες συναρτήσεις της MATLAB σχεδιάζουμε το αρχικό σήμα και το ανακατασκευασμένο σήμα.

```
% Question 4
for p = 5:5:10
   for N = 1:3
      [a_quantized, ~, ~, y_hat, ~] = dpcm_encoder(x, p, N, min_value, max_value);
      x_reconstructed = dpcm_decoder(y_hat, p, a_quantized);

      figure;
      subplot(2,1,1);
      plot(x);
      title(['Reconstructed signal and initial signal for p=', num2str(p), ' and N=', num2str(N)]);
      xlabel('Initial Signal');

      subplot(2,1,2);
      plot(x_reconstructed);
      xlabel('Reconstructed Signal');
   end
end
```

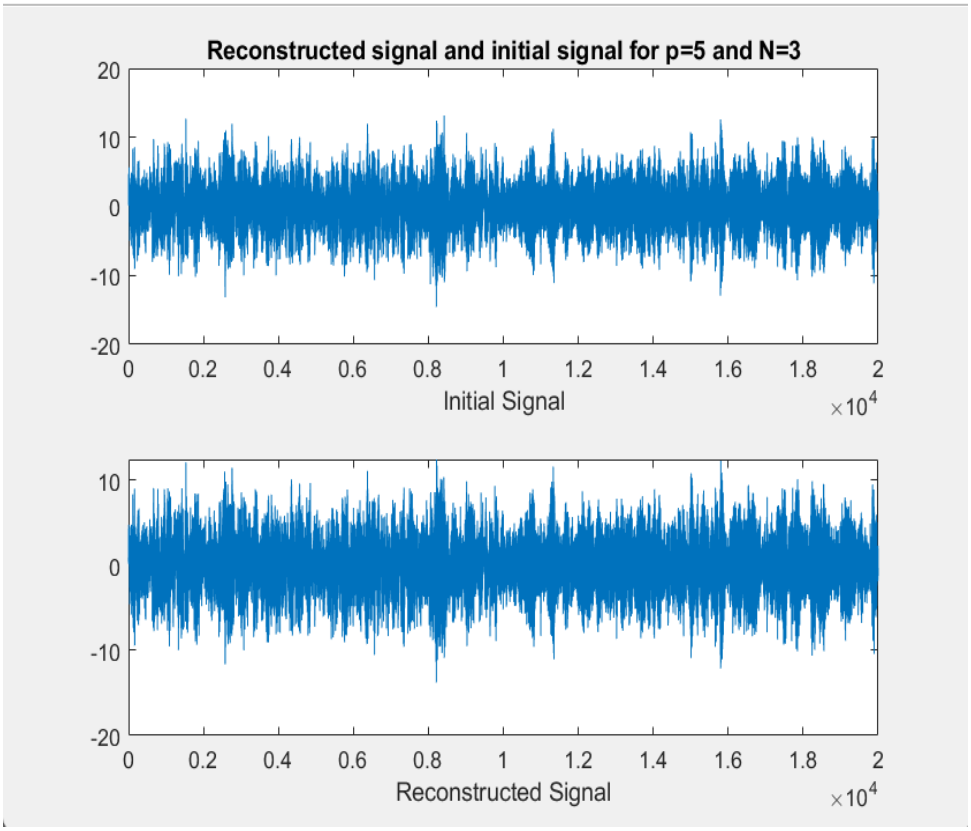
Οι γραφικές παραστάσεις που λαμβάνουμε είναι οι εξής:

Εικόνα 12: p=5, N=1

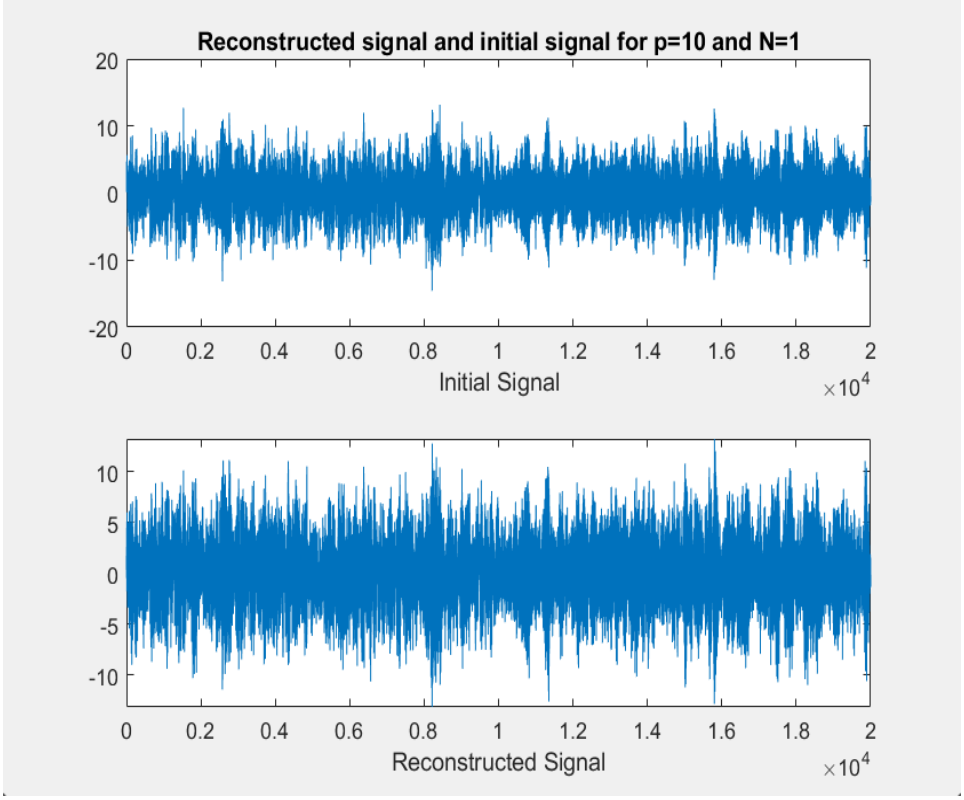




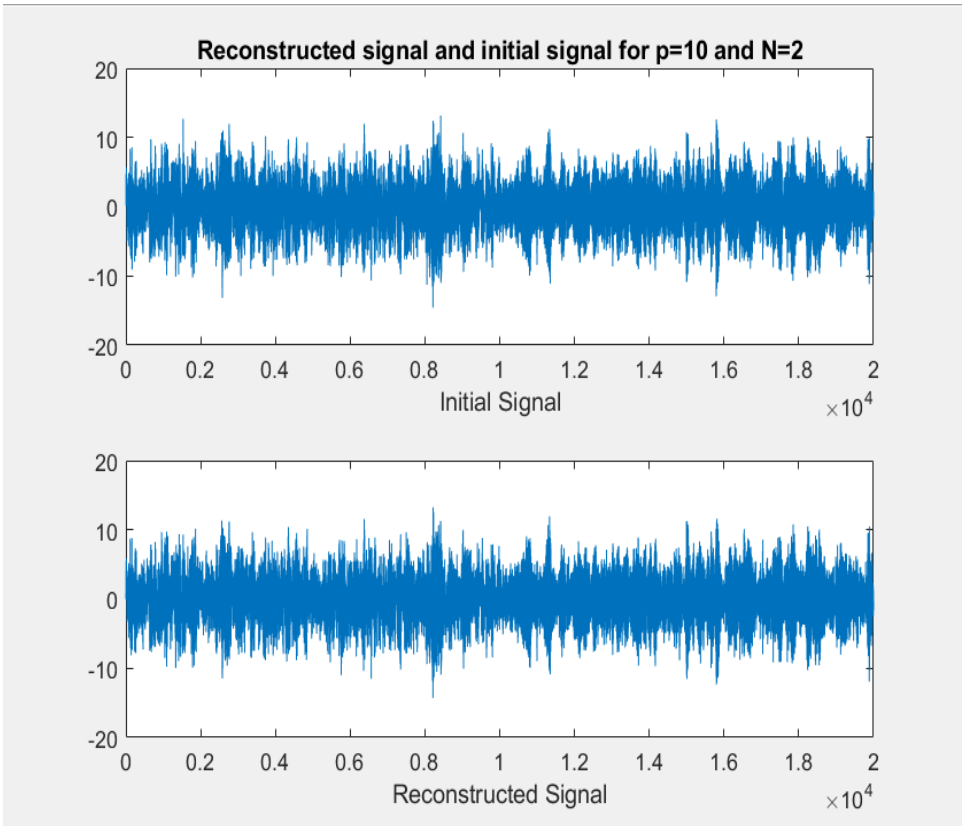
Εικόνα 13: p=5, N=2



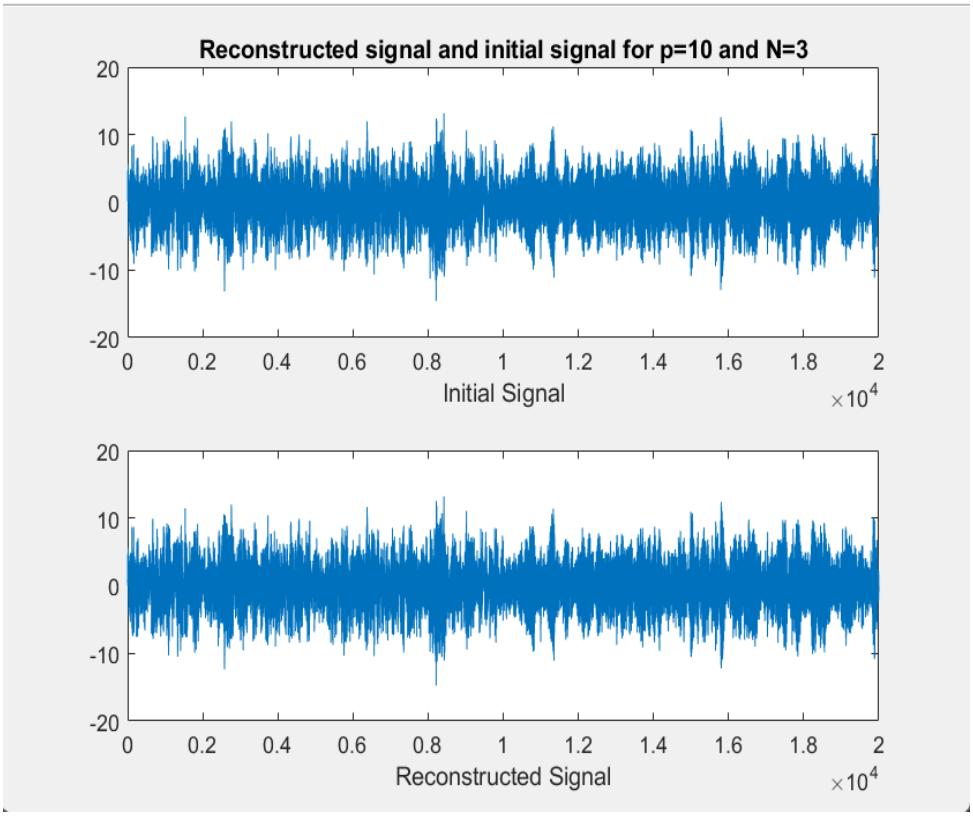
Εικόνα 14: p=5, N=3



Εικόνα 15: p=10, N=1



Εικόνα 16: p=10, N=2



Εικόνα 17: p=10, N=3

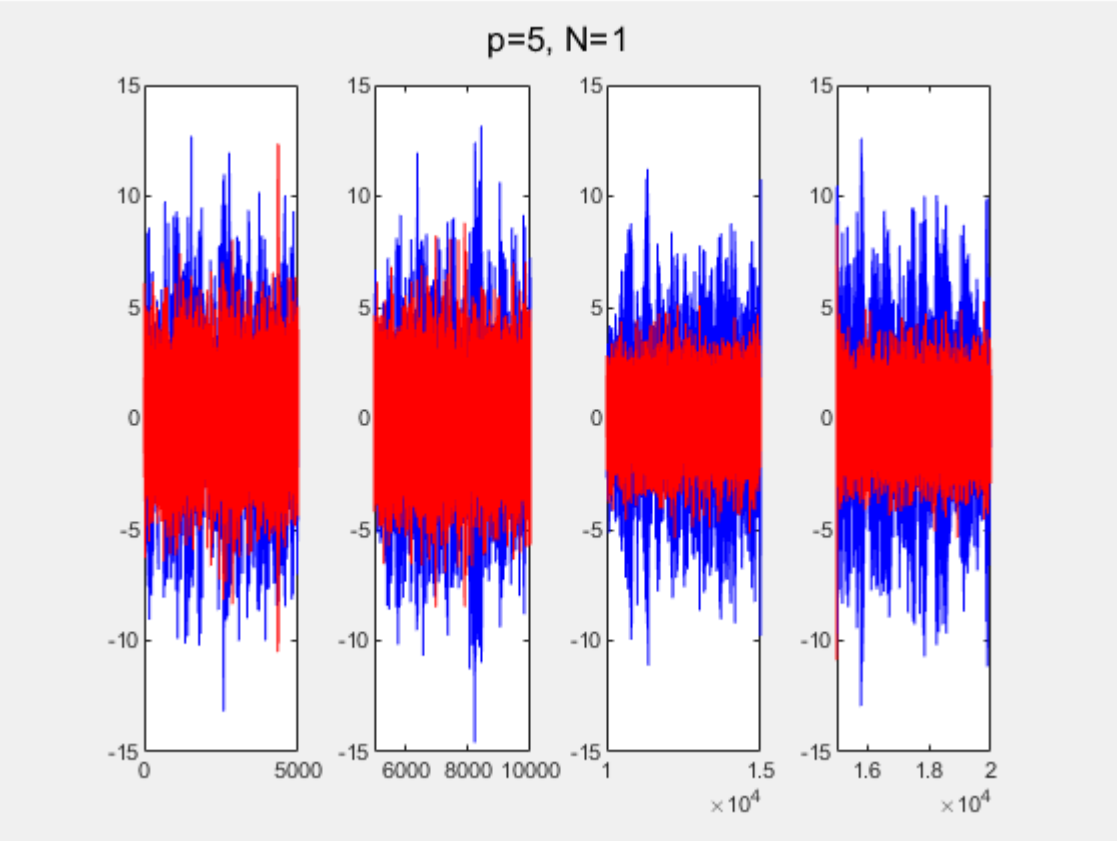
Συμπέρασμα

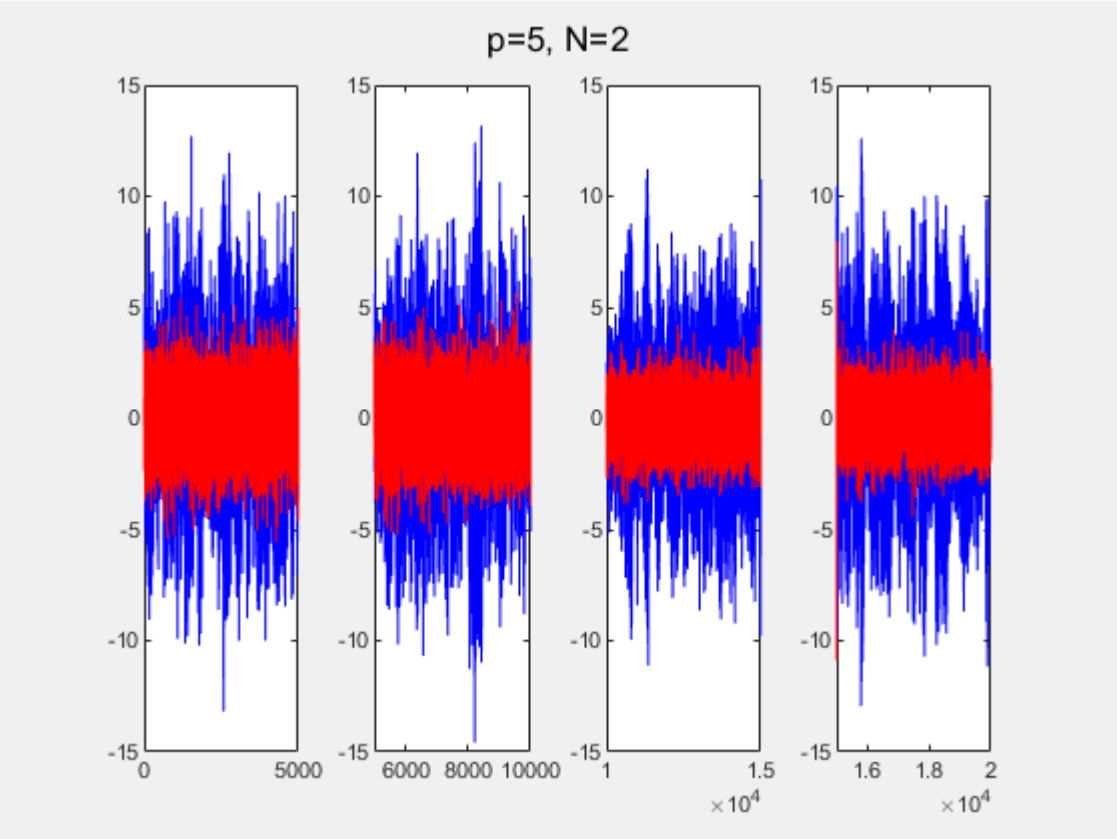
Παρατηρούμε ότι όντως η καλύτερη ανακατασκευή πραγματοποιείται όταν είναι p = 10 και N = 3, ενώ η χειρότερη όταν είναι p = 5 και N = 1. Με αυτό το συμπέρασμα φαίνεται ότι συμφωνούν και οι γραφικές παραστάσεις του Μέσου Τετραγωνικού Σφάλματος. Γενικά, η αποδοτικότερη ανακατασκευή συμβαίνει με τον ιδανικό συνδυασμό μιας μεγάλης τιμής N και μιας μεγάλης τιμής p. Πράγματι βλέπουμε ότι η ανακατασκευή που προκύπτει για p = 10 αλλά N = 1 απέχει πάρα πολύ από το πραγματικό σήμα παρά τη μεγάλη τιμή p, όπως επίσης το ίδιο συμβαίνει και για p = 5 αλλά N = 3 παρά τη μεγάλη τιμή του N.

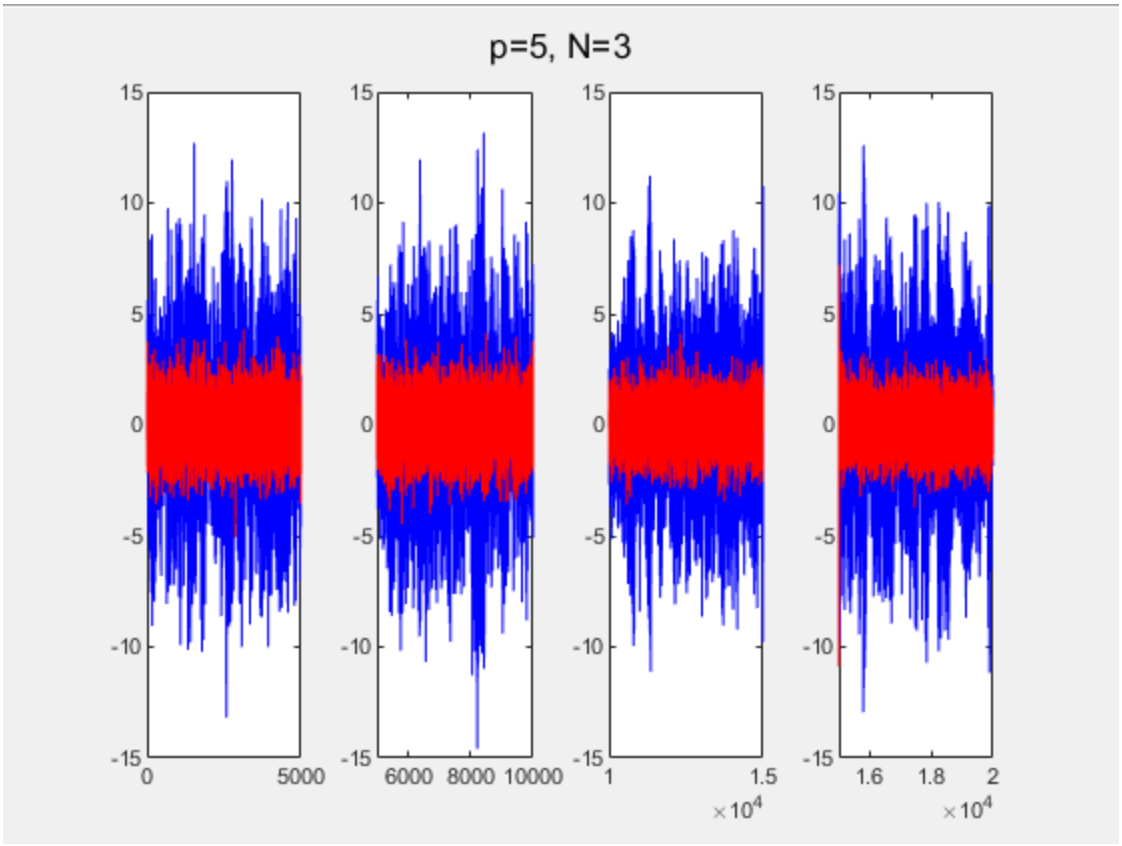
Χωρίστε τα δείγματα της πηγής εισόδου σε ισομεγέθη μη επικαλυπτόμενα υποσύνολα των 5000 δειγμάτων. Επαναλάβετε τα παραπάνω ερωτήματα για κάθε ένα από τα υποσύνολα. Σχολιάστε τα αποτελέσματα που λάβατε στα υποσύνολα των 5000 δειγμάτων και συγκρίνετε με αυτά των 20000 δειγμάτων. Τι παρατηρείτε;

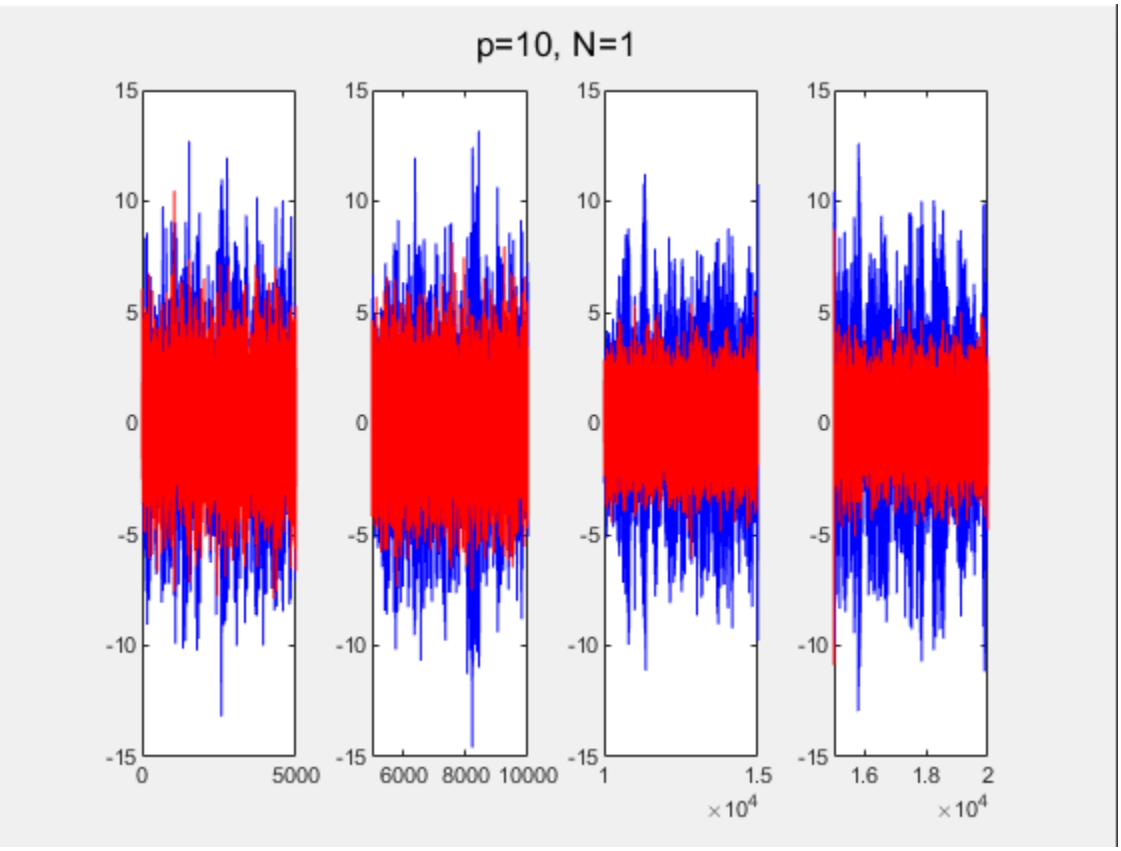
Χωρίζουμε τώρα τη πηγή εισόδου σε 4 ισομεγέθη μη επικαλυπτόμενα υποσύνολα των 5.000 δειγμάτων το καθένα και εκτελούμε όλα τα παραπάνω ερωτήματα σε κάθε υποσύνολο ξεχωριστά.

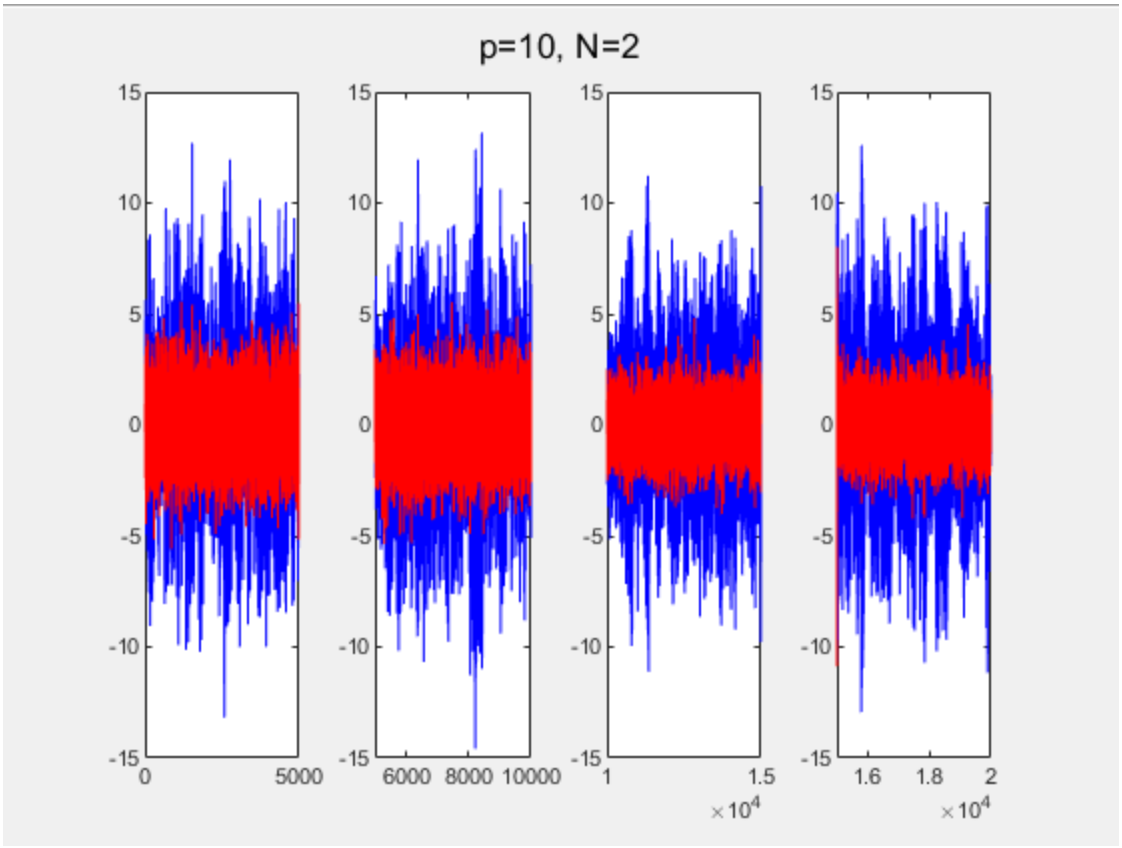
Αρχικά, πηγαίνουμε σε κάθε υποσύνολο και παρατηρούμε το αρχικό σήμα και το σφάλμα πρόβλεψής του:

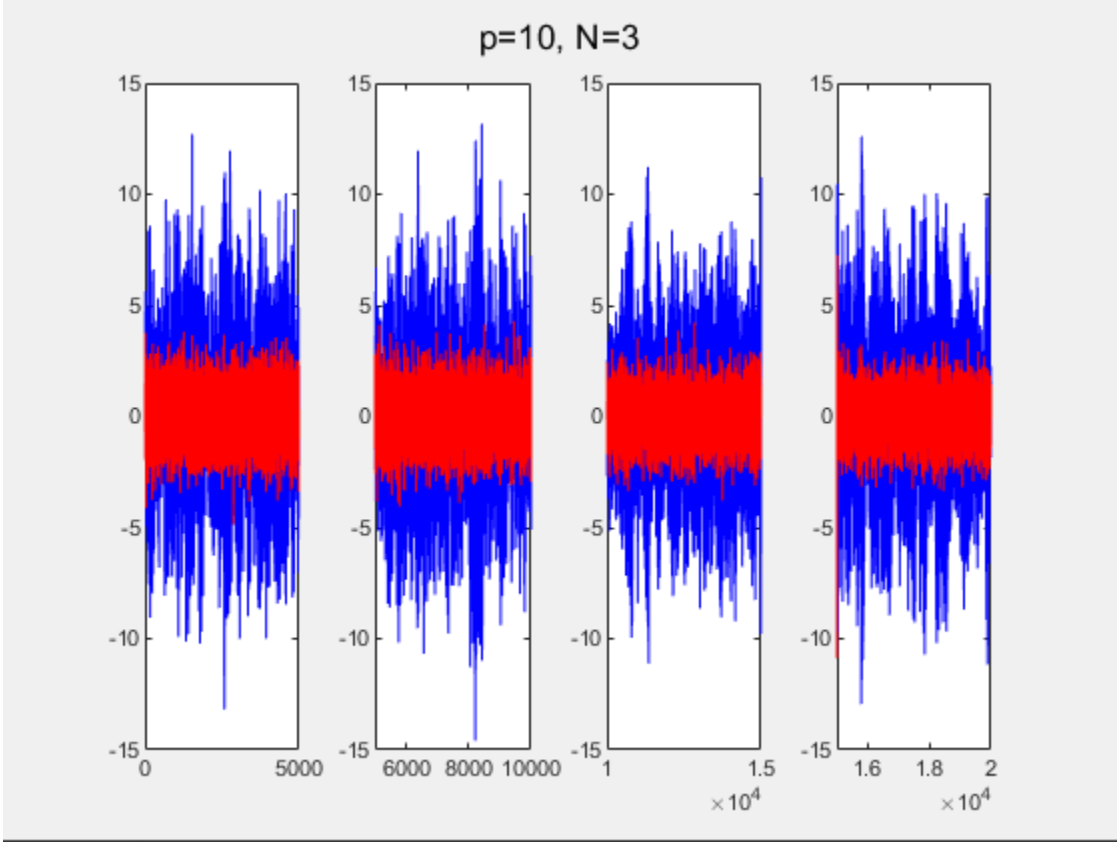





p=5, N=3
15
10
5
0
-5
-10
-15
0
5000
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2
×10^4

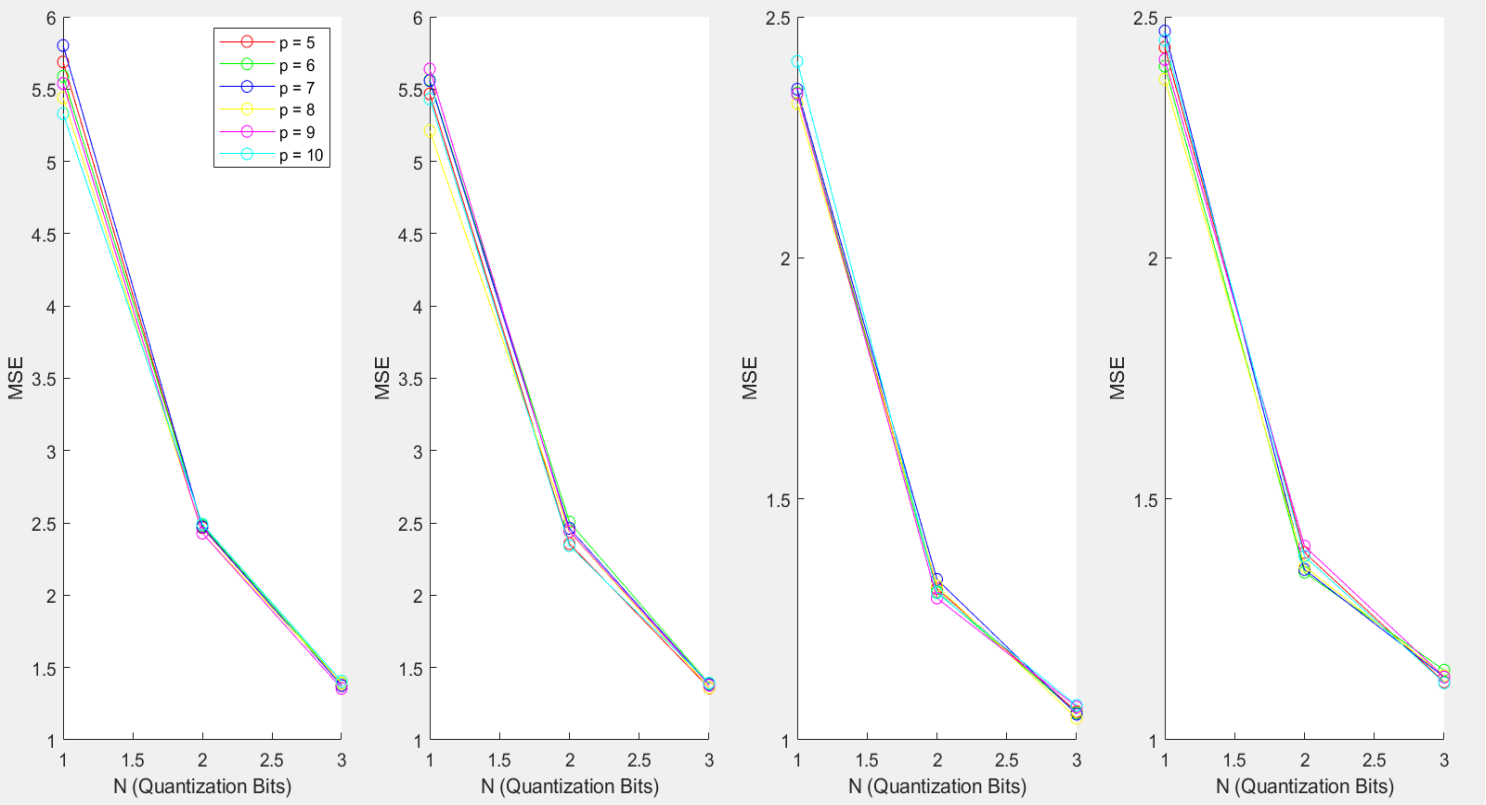

p=10, N=1
15
10
5
0
-5
-10
-15
0
5000
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2
×10^4


p=10, N=2
15
10
5
0
-5
-10
-15
0
5000
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2
×10^4


p=10, N=3
15
10
5
0
-5
-10
-15
0
5000
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2
×10^4

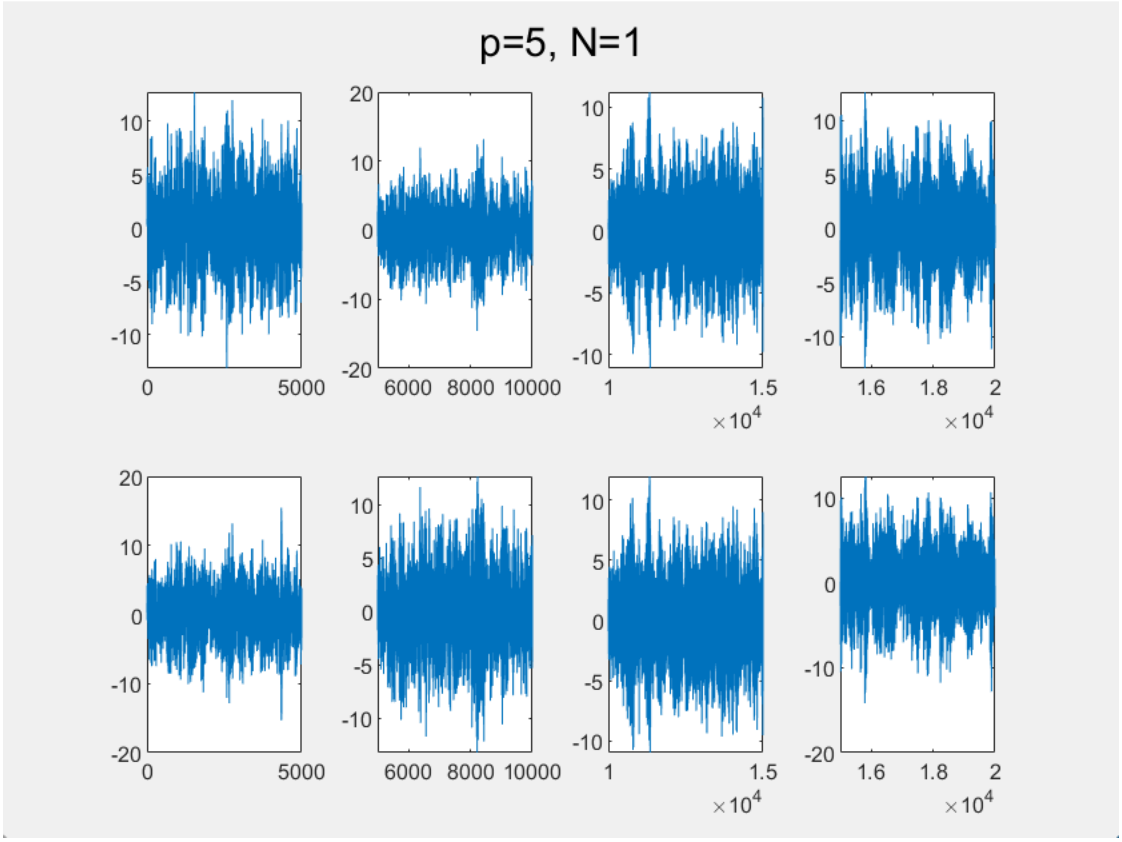
Παρατηρούμε ότι το σήμα σφάλματος σε κάποια υποσύνολα είναι μικρότερο σε σχέση με το σήμα σφάλματος στο ίδιο διάστημα όταν το φίλτρο πρόβλεψης εφαρμόζεται στο ενιαίο σήμα. Πιο συγκεκριμένα, αν το σήμα περιέχει τοπικές ανωμαλίες ή ασυνέχειες σε συγκεκριμένα μόνο τμήματα, τότε η διαίρεση σε μικρότερα υποσύνολα είναι δυνατόν να μειώσει το συνολικό σφάλμα πρόβλεψης καθώς οι ανωμαλίες δεν θα επηρεάσουν τη συνολική πρόβλεψη αλλά μόνο το τμήμα στο οποίο εντοπίζεται. Για παράδειγμα, για τιμές N = 1 και p = 5 παρατηρούμε ότι στα διαστήματα **[10.001, 15.000]** και **[15.001, 20.000]** το τοπικό σφάλμα πρόβλεψης είναι μικρότερο συγκριτικά με το ολικό σφάλμα πρόβλεψης σε αυτές τις περιοχές έτσι όπως αποτυπώνεται στην **Εικόνα 3**. Γενικά, μικρότερα υποσύνολα μπορούν να επιτρέψουν την καλύτερη προσαρμογή του φίλτρου σε κάθε τμήμα ξεχωριστά, οπότε συμβάλλουν και στη μεγαλύτερη ακρίβεια πρόβλεψης.

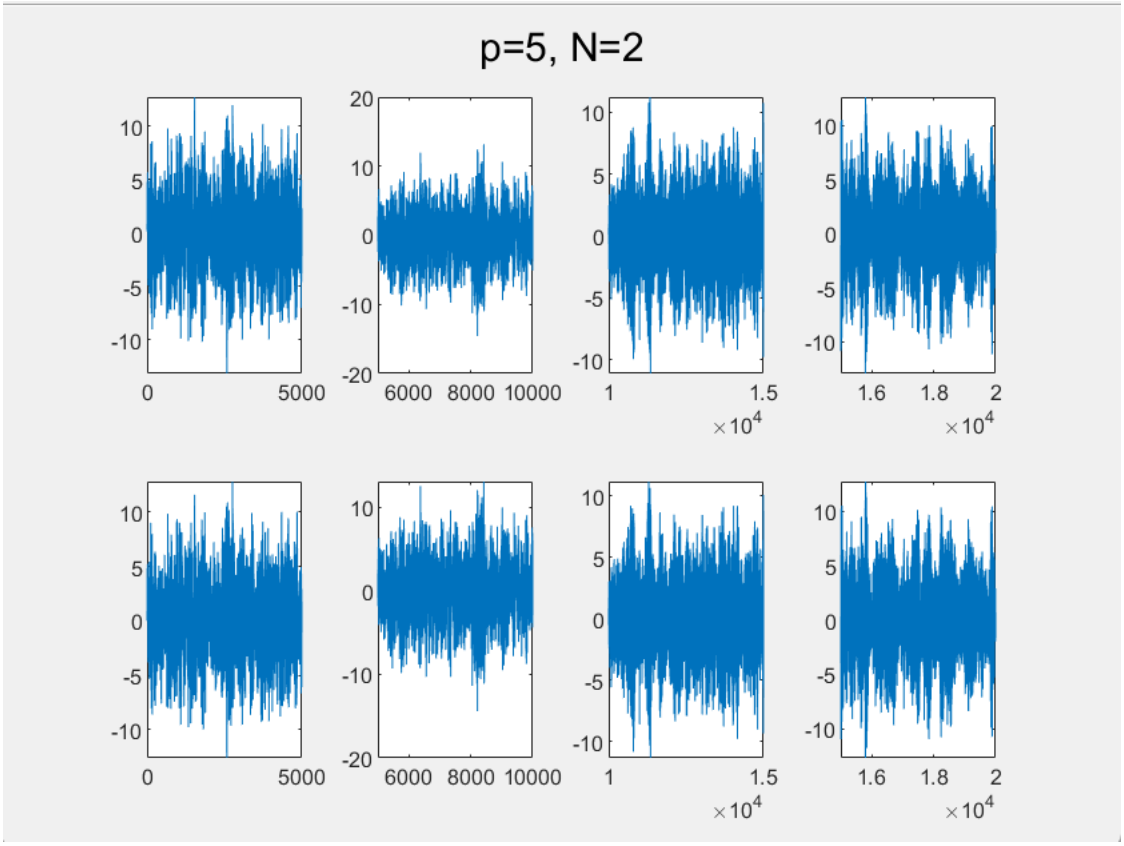
Όσον αφορά το MSE λαμβάνουμε το εξής αποτέλεσμα:

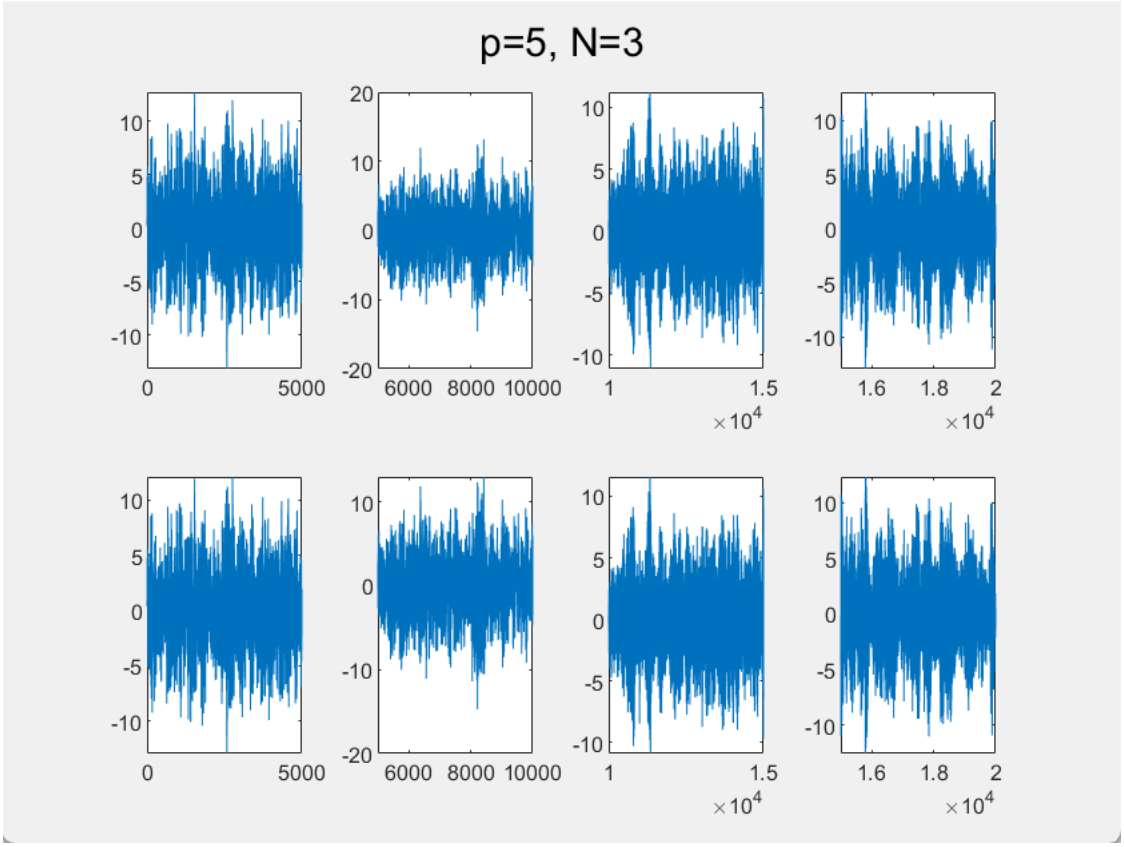


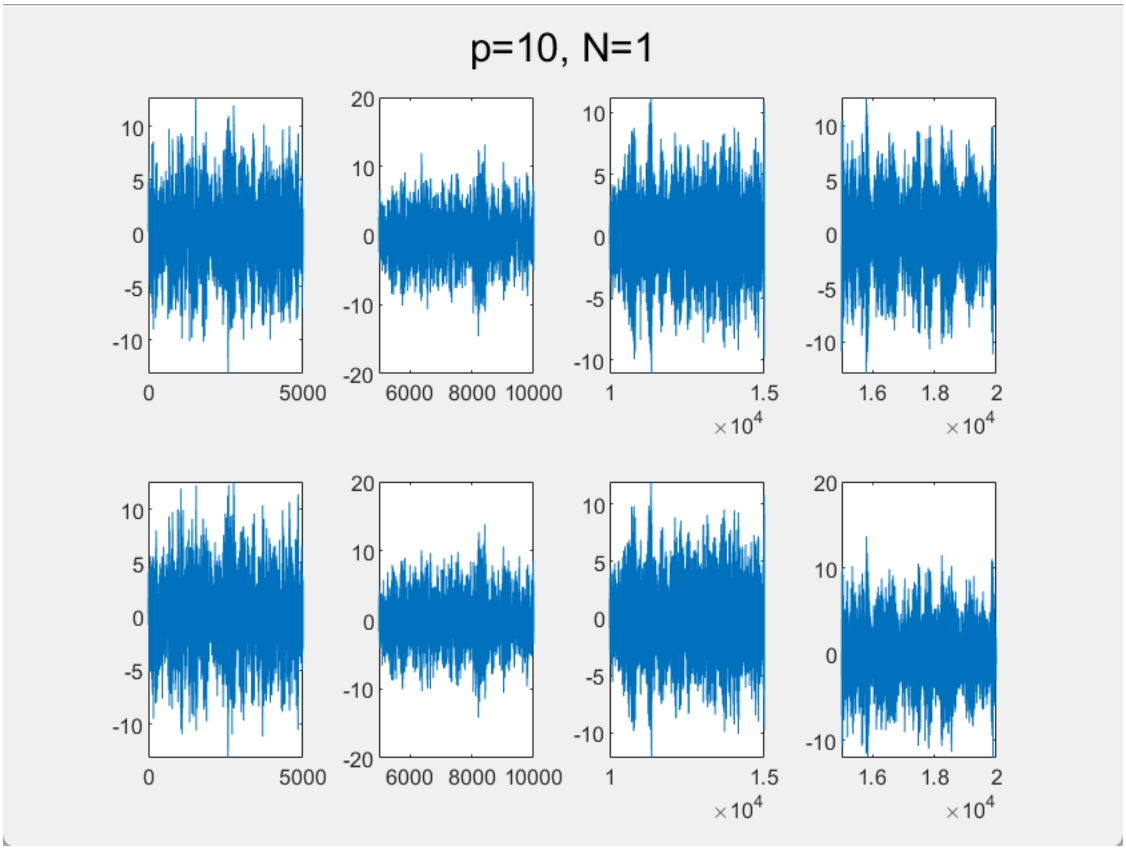
Παρατηρούμε ακόμα ότι σημειώνεται σημαντική διαφοροποίηση και στο MSE κάθε υποσυνόλου. Για την ακρίβεια, βλέπουμε ότι στη χειρότερη περίπτωση το MSE έχει μέγιστη τιμή λίγο μικρότερη από 6 και ότι όσο ακολουθεί φθίνουσα πορεία η τιμή του δεν πρόκειται να πέσει ποτέ κάτω από το 1,5 στη χειρότερη περίπτωση, ενώ για τη καλύτερη περίπτωση τείνει ιδανικά και στο μηδέν. Όλα αυτά έρχονται σε αντίθεση με όσα βλέπουμε στις Εικόνες 9, 10, 11, όπου το συνολικό MSE έχει μέγιστη τιμή γύρω στο 7,5 και όσο φθίνει δεν πέφτει ποτέ κάτω από μια τιμή λίγο μικρότερη από το 2. Συνεπώς, βλέπουμε ότι το MSE λειτουργεί καλύτερα για μικρότερου μεγέθους δείγματα όταν το κάθε δείγμα ακολουθεί διαφορετικό μοτίβο σε σχέση με τα υπόλοιπα ή έχει τοπικές ανωμαλίες. Επίσης, κάτι άλλο που παρατηρούμε είναι ότι η αύξηση του παράγοντα p δεν επιφέρει τόσο απότομη πτώση του MSE για κάθε τιμή του N σε αντίθεση με την προηγούμενη περίπτωση όπου βλέπουμε κατά τη μετάβαση από p = 6 σε p = 7 πτώση της τάξεως των 2 μονάδων για N = 1.

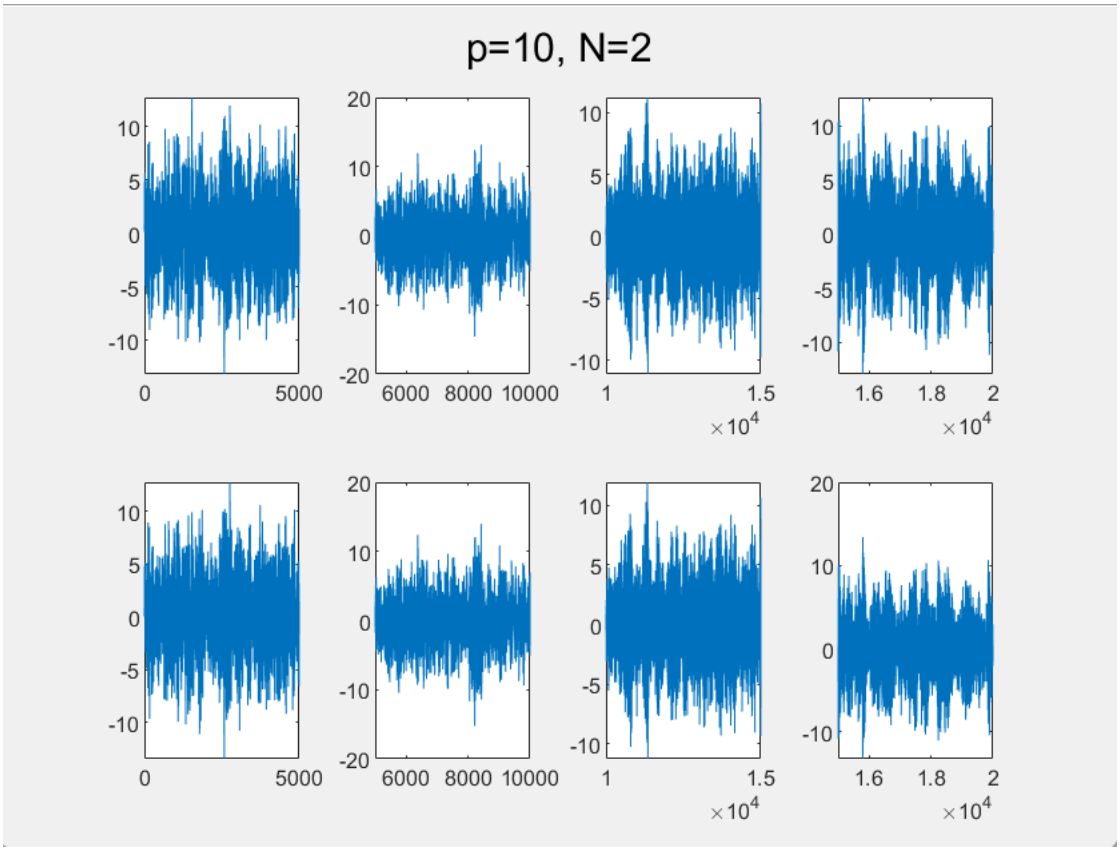
Τέλος, βλέπουμε πιο κάτω και τη τιμή του ανακατασκευασμένου σήματος σε σχέση με το αρχικό για κάθε υποσύνολο:

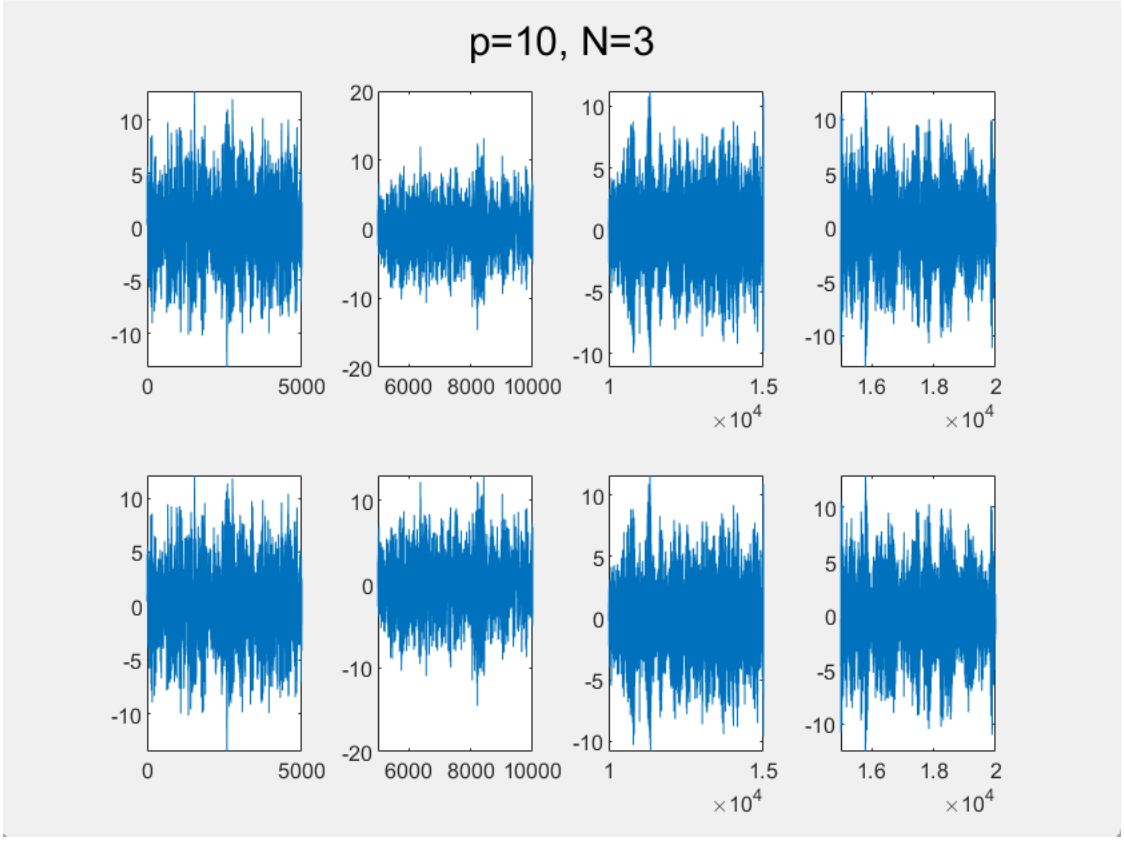





p=5, N=3
10
5
0
-5
-10
0
5000
20
10
0
-10
-20
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2


p=10, N=1
10
5
0
-5
-10
0
5000
20
10
0
-10
-20
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2


p=10, N=2
10
5
0
-5
-10
0
5000
20
10
0
-10
-20
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2


p=10, N=3
10
5
0
-5
-10
0
5000
20
10
0
-10
-20
6000
8000
10000
1
1.5
×10^4
1.6
1.8
2

Πάλι παρατηρούμε ότι η ανακατασκευή σε συγκεκριμένα σημεία των υποσυνόλων είναι ακριβέστερη από ό, τι στο συνολικό σήμα στο ίδιο σημείο της γραφικής παράστασης, Για παράδειγμα για N = 1 και p = 5 σύμφωνα με την **Εικόνα 12** το ανακατασκευασμένο σήμα έχει τιμές που απέχουν αρκετά από τα άκρα του διαστήματος [–10, 10] για δείγματα κοντά στο 20.000. Από την άλλη στο αντίστοιχο υποσύνολο που περιέχει αυτά τα δείγμα για τις ίδιες τιμές των παραμέτρων η ανακατασκευή φαίνεται να είναι πολύ πιο ακριβής, καθώς το ανακατασκευασμένο σήμα «ξεφεύγει» ελάχιστα από το εύρος [–10, 10]. Οπότε, πάλι βλέπουμε ότι ο διαχωρισμός σε μικρότερα σύνολα λειτουργεί βελτιωτικά στην όλη διαδικασία.

Συμπέρασμα

Τα μικρότερα υποσύνολα επιτρέπουν στον αλγόριθμο να προσαρμόζεται πιο ευκολά σε τοπικές ιδιαιτερότητες του σήματος, πιθανώς βελτιώνοντας έτσι την ποιότητα της λειτουργίας του DPCM συστήματος.

Όλος ο κώδικας από τον οποίον προκύπτουν οι γραφικές παραστάσεις ποτά αξιοποιήσαμε βρίσκεται στο **Παράρτημα** από κάτω στη χωρίο **main2.m** μαζί με όλους τους κώδικες που χρησιμοποιήθηκαν σε αυτή την εργασία.

ΠΑΡΑΡΤΗΜΑ

ΜΕΡΟΣ Α

huffman_encoding.m

```
% Read the PNG image
image = imread('parrot.png');

% Convert the image to a column vector
image_vector = image(:);

%First Order Source
% Find unique pixel values and their appearance probabilities
[unique_values, ~, pixel_counts] = unique(image_vector);
total_pixels = numel(image_vector);
appearance_probabilities = histcounts(image_vector, [unique_values; max(unique_values)+1]) / total_pixels;
disp('First Order Source Pixel Values and Appearance Probabilities:');
for i = 1:length(unique_values)
    % Concatenate the values and probabilities as a string
    result_string = strcat(num2str(unique_values(i)), ' , ', num2str(appearance_probabilities(i)));
    disp(result_string);
end

% Encode the first order image using Huffman coding
huffman_dict = huffmandict(unique_values, appearance_probabilities);
encoded_image = huffmanenco(image_vector, huffman_dict);
% Check if Huffman-encoded image is correct
decompressed_image_vector = huffmandeco(encoded_image, huffman_dict);
decompressed_image = reshape(decompressed_image_vector, size(image));
imshow(decompressed_image);
disp('Huffman Encoding:');
disp(huffman_dict);

% Calculate compression ratio J
j = numel(encoded_image) / numel(dec2bin(image_vector));
disp('Compression Ratio:');
disp(num2str(j));

% Calculate entropy
entropy = -dot(appearance_probabilities, log2(appearance_probabilities));
disp('Entropy of Huffman Encoding:');
disp(num2str(entropy));

% Calculate mean length of code
% Convert Huffman codes to double
huffman_codes_double = cellfun(@double, huffman_dict(:, 2), 'UniformOutput', false);
% Create a vector of lengths of Huffman codes
huffman_code_lengths = cellfun(@length, huffman_codes_double);
```

```
mean_length = dot(huffman_code_lengths, appearance_probabilities);
disp('Mean Length of Huffman Encoding:');
disp(num2str(mean_length));

% Calculate efficiency of code
efficiency = entropy / mean_length;
disp('Efficiency of Huffman Encoding:');
disp(num2str(efficiency));

%Binary Symmetrical Channel
% Calculate error probability (p) of BSC
received_sequence = binary_symmetric_channel(encoded_image);
differing_bits = sum(encoded_image ~= received_sequence); % Count the number of differing bits between the two sequences
p = differing_bits / length(encoded_image);
disp('Estimated Error Probability:');
fprintf('%.2f\n', p);

% Calculate capacity of BSC
% Calculate output entropy H(Y)
output_entropy = -p * log2(p) - (1 - p) * log2(1 - p);
bsc_capacity = 1 - output_entropy;
disp('BSC Capacity:');
disp(num2str(bsc_capacity));

% Calculate joint information
% Calculate conditional entropy H(Y|X)
p_x0 = sum(encoded_image == 0) / length(encoded_image);
p_x1 = 1 - p_x0;
p_y_given_x0 = sum(received_sequence(encoded_image == 0) == 0) / sum(encoded_image == 0);
p_y_given_x1 = sum(received_sequence(encoded_image == 1) == 1) / sum(encoded_image == 1);
conditional_entropy = - (p_x0 * p_y_given_x0 * log2(p_y_given_x0) + p_x1 * p_y_given_x1 * log2(p_y_given_x1));
joint_information = output_entropy - conditional_entropy;
disp('Joint Information:');
disp(num2str(joint_information));

% Second Order Source
% Create pairs of neighboring pixels
pairs = [image_vector(1:end-1), image_vector(2:end)];

% Find unique pairs and their counts
[unique_pairs, ~, idx] = unique(pairs, 'rows');
pair_counts = accumarray(idx, 1);

% Calculate probabilities
pair_probabilities = pair_counts / sum(pair_counts);
```

```
% Display the pairs and their probabilities
disp('Second Order Source Pixel Values and Appearance Probabilities:');
for i = 1:length(unique_pairs)
    disp([num2str(unique_pairs(i, :)), ', ', num2str(pair_probabilities(i))]);
end

% Convert pairs to numeric symbols
symbols = 1:size(unique_pairs, 1);

% Encode the second order image using Huffman coding and numeric symbols
[numericdict, ~] = huffmandict(symbols, pair_probabilities);

% Create a new dictionary with numeric pairs as one symbol for the first column
dict = cell(size(numericdict, 1), 2);
for i = 1:size(numericdict, 1)
    dict{i, 1} = unique_pairs(numericdict{i, 1}, :);
    dict{i, 2} = numericdict{i, 2};
end
disp('Second Order Huffman Encoding:');
disp(dict);

% Calculate entropy
second_order_entropy = -dot(pair_probabilities, log2(pair_probabilities));
disp('Entropy of Second Order Huffman Encoding:');
disp(num2str(second_order_entropy));

% Calculate mean length of code
% Create a vector of lengths of Huffman codes
huffman_code_lengths = cellfun(@length, dict(:, 2));
second_order_mean_length = dot(huffman_code_lengths, pair_probabilities);
disp('Mean Length of Second Order Huffman Encoding:');
disp(num2str(second_order_mean_length));

% Calculate efficiency of code
second_order_efficiency = second_order_entropy / second_order_mean_length;
disp('Efficiency of Second Order Huffman Encoding:');
disp(num2str(second_order_efficiency));
```

ΜΕΡΟΣ Β

my_quantizer.m

```
function [y_hat] = my_quantizer(y, N, min_value, max_value)

    % Quantization step
    delta = (max_value - min_value)/ 2^N;

    % Calculation of centers of quantization areas
    centers = zeros(2^N, 1);
    centers(1) = max_value - delta/2;
    centers(2^N) = min_value + delta/2;
    for i = 2:2^N - 1
        centers(i) = centers(i - 1) - delta;
    end

    % Process each element in y
    % Limiting the dynamic range of the current element
    if y < min_value
        y = min_value;
    end
    if y > max_value
        y = max_value;
    end

    %Find the area of each input sample
    for i = 1: 2^N
        if((y <= centers(i) + delta/2) && (y >= centers(i) - delta/2))
            %Create quantized input signal
            y_hat = centers(i);
        end
    end
end
```

dpcm_encoder.m

```
% Question 1
function [a_quantized, a, y, y_hat, y_hat_prediction] = dpcm_encoder(x, p, quantization_bits,
quantization_min_value, quantization_max_value)
    N = length(x);
    r = zeros(p, 1);
    R = zeros(p, p);
    a_quantized = zeros(p, 1);

    % Create autocorrelation matrix R and autocorrelation vector r
    for i = 1:p
        for j = 1:p
```

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

    for k = 1:p
        sum = 0;
        for n = p+1:N
            sum = sum + x(n) * x(n - k);
        end
        r(k) = sum / (N - p);
    end

    % Quantize coefficients a with N=8 bits and dynamic range [-2,2]
    a = R\r;
    for i = 1:p
        a_quantized(i) = my_quantizer(a(i), 8, -2, 2);
    end

    % Initialize y(n), y_hat_prediction(n) and y_hat(n)
    y = zeros(N, 1);
    y_hat = zeros(N, 1);
    y_hat_prediction = 0;
    mem = zeros(p, 1);

    % Calculate all the values from 1 to N
    for i = 1:N
        % Calculate the error and quantize it
        y(i) = x(i) - y_hat_prediction;
        y_hat(i) = my_quantizer(y(i), quantization_bits, quantization_min_value, quantization_max_value);
        % Update the y_hat_prediction to include the quantized error (reconstruction)
        y_hat_prediction = y_hat_prediction + y_hat(i);
        % Calculate the prediction y_hat'(n) using previous p y_hat_prediction values
        mem = [y_hat_prediction; mem(1:p -1)];
        y_hat_prediction = a_quantized.' * mem;
    end
end
```

dpcm_decoder.m

```
% Question 1
function [x_reconstructed] = dpcm_decoder(y_hat, p, a_quantized)
    N = length(y_hat);
    x_reconstructed = zeros(N, 1);
```

```
    mem = zeros(p, 1);
    y_hat_prediction = 0;

    % Calculate all the values from 1 to N
    for i = 1:N
        % Add the prediction to the quantized error to reconstruct x
        x_reconstructed(i) = y_hat_prediction + y_hat(i);
        % Calculate the prediction y_hat_prediction(n) using the quantized coefficients a_quantized
        % and the previously reconstructed values
        mem = [x_reconstructed(i); mem(1:p - 1)];
        y_hat_prediction = a_quantized.' * mem;
    end
end
```

main.m

```
data = load('source.mat');
x = data.t;

% Define the constants
min_value = -3.5;
max_value = 3.5;

% Question 2
for p = 5:5:10
    for N = 1:3
        [~, ~, y, ~, ~] = dpcm_encoder(x, p, N, min_value, max_value);

        figure;
        plot(x, 'b');
        hold on;
        plot(y, 'r');
        hold off;

        title(['Prediction error and initial signal for p=', num2str(p), ' and N=', num2str(N)]);
        legend('Initial Signal', 'Prediction Error');
    end
end

% Question 3
% MSE matrix initialization
mse_matrix = zeros(6, 3);

% Loop over each p and N value
for p = 5:10
    result_string = strcat(' p = ', num2str(p));
    disp(result_string);
    for N = 1:3
```

```
        [~, a, y, ~, ~] = dpcm_encoder(x, p, N, min_value, max_value);
        disp(a);
        currentMSE = mean(y.^2);  % Calculate current MSE
        mse_matrix(p - 4, N) = currentMSE;  % Assign to the matrix
    end
end

% Define a set of colors
colors = [1, 0, 0; % Red
          0, 1, 0; % Green
          0, 0, 1; % Blue
          1, 1, 0; % Yellow
          1, 0, 1; % Magenta
          0, 1, 1]; % Cyan

figure;
hold on;

% Loop over each p value
for p = 5:10
    plot(1:3, mse_matrix(p - 4, :), 'Color', colors(p - 4, :), 'Marker', '.', 'MarkerSize', 15);
end

title('MSE (E[y^2]) for different values of p');
xlabel('N (Quantization Bits)');
xticks(1:3);
ylabel('MSE');
legend('p = 5', 'p = 6', 'p = 7', 'p = 8', 'p = 9', 'p = 10');

hold off;

% Question 4
for p = 5:5:10
    for N = 1:3
        [a_quantized, ~, ~, y_hat, ~] = dpcm_encoder(x, p, N, min_value, max_value);
        x_reconstructed = dpcm_decoder(y_hat, p, a_quantized);

        figure;
        subplot(2,1,1);
        plot(x);
        title(['Reconstructed signal and initial signal for p=', num2str(p), ' and N=', num2str(N)]);
        xlabel('Initial Signal');

        subplot(2,1,2);
        plot(x_reconstructed);
        xlabel('Reconstructed Signal');
    end
```

```
end
```

main2.m

```
data = load('source.mat');
x = data.t;
numSamples = 20000;
subsetSize = 5000; % Size of each subset
numSubsets = ceil(numSamples / subsetSize); % Number of subsets
subsets = cell(numSubsets, 1); % Initialize cell array for subsets

% Define the constants
min_value = -3.5;
max_value = 3.5;

% Question 2
% Iterate through each combination of p and N
for p = 5:5:10
    for N = 1:3
        % Create a new figure for each combination of p and N
        figure;
        title = ['p=', num2str(p), ', N=', num2str(N)];
        sgtitle(title); % Super title for the figure

        for k = 1:numSubsets
            startIdx = (k - 1) * subsetSize + 1;
            endIdx = min(k * subsetSize, numSamples);
            subsets{k} = x(startIdx:endIdx);

            [~, ~, y, ~, ~] = dpcm_encoder(subsets{k}, p, N, min_value, max_value);

            subplot(1, numSubsets, k);
            plot((startIdx:endIdx), subsets{k}, 'b');
            hold on;
            plot((startIdx:endIdx), y, 'r');
            hold off;
        end
    end
end

% Question 3
mse_matrix = zeros(6, 3, numSubsets);

% Calculate MSE for each subset
for k = 1:numSubsets
    startIdx = (k - 1) * subsetSize + 1;
    endIdx = min(k * subsetSize, numSamples);
    subsets{k} = x(startIdx:endIdx);
```

```
    for p = 5:10
        for N = 1:3
            [~, ~, y, ~, ~] = dpcm_encoder(subsets{k}, p, N, min_value, max_value);
            currentMSE = mean(y.^2);
            mse_matrix(p - 4, N, k) = currentMSE;
        end
    end
end

% Define a set of colors
colors = [1, 0, 0; % Red
          0, 1, 0; % Green
          0, 0, 1; % Blue
          1, 1, 0; % Yellow
          1, 0, 1; % Magenta
          0, 1, 1]; % Cyan

% Create one figure
figure;

% Loop over each subset
for k = 1:numSubsets
    subplot(1, numSubsets, k);
    hold on;

    % Plot MSE for each value of p
    for p = 5:10
        mse_values = mse_matrix(p - 4, :, k); % Extract MSE values for current p and subset
        plot(1:3, mse_values, 'Color', colors(p - 4, :), 'Marker', 'o', 'LineStyle', '-');
        % The 'LineStyle', '-' argument adds lines between points
    end

    hold off;
    xlabel('N (Quantization Bits)');
    ylabel('MSE');
    if k == 1
        legend(arrayfun(@(p) ['p = ' num2str(p)], 5:10, 'UniformOutput', false), 'Location', 'best');
    end
end

% Question 4
% Iterate through each value of p
for p = 5:5:10
    for N = 1:3
        figure;
        sgtitle(['p=', num2str(p), ', N=', num2str(N)]);
```

```
        for k = 1:numSubsets
            startIdx = (k - 1) * subsetSize + 1;
            endIdx = min(k * subsetSize, numSamples);
            subsets{k} = x(startIdx:endIdx);

            [a_quantized, ~, ~, y_hat, ~] = dpcm_encoder(subsets{k}, p, N, min_value, max_value);
            x_reconstructed = dpcm_decoder(y_hat, p, a_quantized);

            % Plot the initial signal for subset k
            subplot(2, numSubsets, k);
            plot((startIdx:endIdx), subsets{k});

            % Plot the reconstructed signal for subset k
            subplot(2, numSubsets, k + numSubsets);
            plot((startIdx:endIdx), x_reconstructed);
        end
    end
end
```