

**Ψηφιακές Τηλεπικοινωνίες 1ο Σετ Εργαστηριακών Ασκήσεων**

Μάθημα Κορμού – CEID\_ΝΥ384

Χειμερινό Εξάμηνο 2023

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

Τμήμα Μηχανικών Η/Υ και Πληροφορικής Πανεπιστήμιο Πατρών **Πολυτεχνική Σχολή**

δ

**Στοιχεία Φοιτητή:**

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

By Chela Arapelli

**ΠΕΡΙΕΧΟΜΕΝΑ**

**ΜΕΡΟΣ Α – ΘΕΩΡΙΑ ΠΛΗΡΟΦΟΡΙΑΣ**

**ΜΕΡΟΣ Α – ΘΕΩΡΙΑ ΠΛΗΡΟΦΟΡΙΑΣ**

Αρχικά, θα φορτώσουμε την πηγή μας, η οποία αναπαρίσταται ως ένας πίνακας ακεραίων ΙΖNxM 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 **Φ = {s1, s2, …, sΝ}**, δηλαδή όλα τα διακριτά εικονοστοιχεία της εικόνας, καθώς και τις πιθανότητες εμφάνισης κάθε εικονοστοιχείου **P =** **(p1, p2, …, pN}**. Αυτή τη διαδικασία ακολουθούμε στο παρακάτω ερώτημα:

**(1) a.** Αρχικά, καλείστε να εντοπίστε 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 1ης 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(\_\_\_)**](https://www.mathworks.com/help/matlab/ref/double.unique.html#d126e1751092) 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(\_\_\_)**](https://www.mathworks.com/help/matlab/ref/double.numel.html#d126e1082414) 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**](https://www.mathworks.com/help/matlab/ref/histcounts.html#buijikm-1-N)**] = histcounts(**[**C**](https://www.mathworks.com/help/matlab/ref/histcounts.html#buijikm-1-C)**,**[**Categories**](https://www.mathworks.com/help/matlab/ref/histcounts.html#buijikm-1-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 o 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)**](https://www.mathworks.com/help/comm/ref/huffmandict.html#d126e56062)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 Η(Χ) 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:

**Η(Χ) = – (Ι)**

Αυτόν τον τύπο εφαρμόζουμε και στην υλοποίηση σε MATLAB μέσω της συνάρτησης [[**C] = dot(A,B)**](https://www.mathworks.com/help/matlab/ref/dot.html#d126e393869)πάνω στο διάνυσμα *appearance\_probabilities*, καθώς ο τύπος στην ουσία εκφράζει το εσωτερικό γινόμενο των διανυσμάτων *appearance\_probabilities* και log2(*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:

**L = (ΙΙ)**

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)**](https://www.mathworks.com/help/matlab/ref/length.html#d126e908581), με ακρίβεια δύο δεκαδικών ψηφίων. Στη συνέχεια, καλούμε πάλι τη συνάρτηση [[**C] = dot(A,B)**](https://www.mathworks.com/help/matlab/ref/dot.html#d126e393869)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)** 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:

**η = (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 2ης τάξης επέκταση, κάθε σύνθετο σύμβολο αποτελείται από τα ζεύγη των διακριτών τιμών των εικονοστοιχείων. Το αλφάβητο της νέας πηγής θα είναι **Φ2 = {σ1, σ2, …, σΚ}** με **σΚ = (si, sj)** για I, j = 1, 2, …, N και πιθανότητες εμφάνισης **Ρ2 = {p1, p2, …, pK}**. Να σημειωθεί ότι η συγκεκριμένη υλοποίηση που ακολουθεί οφείλεται στο γεγονός ότι η πηγή είναι διακριτή και με μνήμη.

**(2) a.** Για να υλοποιήσουμε την επέκτασηξεκινάμε ορίζοντας ένα cell array *symbols* που θα αποθηκεύει τα καινούργια σύμβολα (ζεύγη) και ένα numeric array *probabilities* που αποθηκεύει τις αντίστοιχες πιθανότητες εμφάνισης κάθε συνδυασμού εικονοστοιχείων. Στη συνέχεια, εκτελούμε δύο εμφωλευμένους επαναληπτικούς βρόχους και εξετάζουμε κάθε δυνατό συνδυασμό εικονοστοιχείων από το διάνυσμα unique\_values. Για κάθε συνδυασμό υπολογίζουμε έτσι τη πιθανότητα εμφάνισής του ως το γινόμενο των αντίστοιχων πιθανοτήτων των διακριτών τιμών του *appearance\_probabilities* και την αποθηκεύουμε στο διάνυσμα *second\_order\_appearance\_probabilities*. Μετά τις δυάδες συμβόλων από το *unique\_values* τις ενώνουμε σε ένα νέο με τη συνάρτηση **strcat( )** και το αποθηκεύουμε στο διάνυσμα *symbol.*

%Second Order Source

% Initialize variables to store symbols and their probabilities

symbols = cell(0);

probabilities = [];

% Find each pair of unique pixel values

disp('Second Order Source Pixel Values and Appearance Probabilities:');

for i = 1:length(unique\_values)

for j = 1:length(unique\_values)

% Calculate the new appearance probability of the pair of values

second\_order\_appearance\_probabilities = appearance\_probabilities(i) \* appearance\_probabilities(j);

% Concatenate the values as a string

symbol = strcat(num2str(unique\_values(i)), num2str(unique\_values(j)));

result\_string = strcat(symbol, ' , ', num2str(second\_order\_appearance\_probabilities));

disp(result\_string);

% Store the symbol and its probability

symbols = [symbols; symbol];

probabilities = [probabilities; second\_order\_appearance\_probabilities];

end

end

Τρέχοντας το κομμάτι κώδικα λαμβάνουμε τα εξής αποτελέσματα:

Second Order Source Pixel Values and Appearance Probabilities:

00 ,0.009248  
017 ,0.007828  
034 ,0.0065714  
051 ,0.0060104  
068 ,0.0073888  
085 ,0.0086999  
0102 ,0.010883  
0119 ,0.0087127  
0136 ,0.0092801  
0153 ,0.006456  
0170 ,0.0037409  
0187 ,0.0031671  
0204 ,0.0032376  
0221 ,0.0024907  
0238 ,0.0021509  
0255 ,0.00030132  
170 ,0.007828  
1717 ,0.006626  
1734 ,0.0055623  
1751 ,0.0050875  
1768 ,0.0062542  
1785 ,0.007364  
17102 ,0.0092118  
17119 ,0.0073748  
17136 ,0.0078551  
17153 ,0.0054647  
17170 ,0.0031665  
17187 ,0.0026808  
17204 ,0.0027405  
17221 ,0.0021083  
17238 ,0.0018206  
17255 ,0.00025505  
340 ,0.0065714  
3417 ,0.0055623  
3434 ,0.0046694  
3451 ,0.0042708  
3468 ,0.0052503  
3485 ,0.0061819  
34102 ,0.0077331  
34119 ,0.006191  
34136 ,0.0065942  
34153 ,0.0045874  
34170 ,0.0026582  
34187 ,0.0022504  
34204 ,0.0023006  
34221 ,0.0017698  
34238 ,0.0015284  
34255 ,0.00021411  
510 ,0.0060104  
5117 ,0.0050875  
5134 ,0.0042708  
5151 ,0.0039062  
5168 ,0.0048021  
5185 ,0.0056542  
51102 ,0.0070729  
51119 ,0.0056625  
51136 ,0.0060313  
51153 ,0.0041958  
51170 ,0.0024312  
51187 ,0.0020583  
51204 ,0.0021042  
51221 ,0.0016187  
51238 ,0.0013979  
51255 ,0.00019583  
680 ,0.0073888  
6817 ,0.0062542  
6834 ,0.0052503  
6851 ,0.0048021  
6868 ,0.0059034  
6885 ,0.0069509  
68102 ,0.008695  
68119 ,0.0069611  
68136 ,0.0074144  
68153 ,0.0051581  
68170 ,0.0029888  
68187 ,0.0025304  
68204 ,0.0025867  
68221 ,0.00199  
68238 ,0.0017185  
68255 ,0.00024074  
850 ,0.0086999  
8517 ,0.007364  
8534 ,0.0061819  
8551 ,0.0056542  
8568 ,0.0069509  
8585 ,0.0081842  
85102 ,0.010238  
85119 ,0.0081963  
85136 ,0.00873  
85153 ,0.0060733  
85170 ,0.0035192  
85187 ,0.0029794  
85204 ,0.0030457  
85221 ,0.0023431  
85238 ,0.0020234  
85255 ,0.00028346  
1020 ,0.010883  
10217 ,0.0092118  
10234 ,0.0077331  
10251 ,0.0070729  
10268 ,0.008695  
10285 ,0.010238  
102102 ,0.012807  
102119 ,0.010253  
102136 ,0.010921  
102153 ,0.0075973  
102170 ,0.0044022  
102187 ,0.003727  
102204 ,0.0038099  
102221 ,0.002931  
102238 ,0.0025312  
102255 ,0.00035459  
1190 ,0.0087127  
11917 ,0.0073748  
11934 ,0.006191  
11951 ,0.0056625  
11968 ,0.0069611  
11985 ,0.0081963  
119102 ,0.010253  
119119 ,0.0082084  
119136 ,0.0087429  
119153 ,0.0060823  
119170 ,0.0035243  
119187 ,0.0029838  
119204 ,0.0030502  
119221 ,0.0023465  
119238 ,0.0020264  
119255 ,0.00028388  
1360 ,0.0092801  
13617 ,0.0078551  
13634 ,0.0065942  
13651 ,0.0060313  
13668 ,0.0074144  
13685 ,0.00873  
136102 ,0.010921  
136119 ,0.0087429  
136136 ,0.0093123  
136153 ,0.0064784  
136170 ,0.0037538  
136187 ,0.0031781  
136204 ,0.0032488  
136221 ,0.0024994  
136238 ,0.0021584  
136255 ,0.00030237  
1530 ,0.006456  
15317 ,0.0054647  
15334 ,0.0045874  
15351 ,0.0041958  
15368 ,0.0051581  
15385 ,0.0060733  
153102 ,0.0075973  
153119 ,0.0060823  
153136 ,0.0064784  
153153 ,0.0045069  
153170 ,0.0026115  
153187 ,0.0022109  
153204 ,0.0022602  
153221 ,0.0017388  
153238 ,0.0015015  
153255 ,0.00021035  
1700 ,0.0037409  
17017 ,0.0031665  
17034 ,0.0026582  
17051 ,0.0024312  
17068 ,0.0029888  
17085 ,0.0035192  
170102 ,0.0044022  
170119 ,0.0035243  
170136 ,0.0037538  
170153 ,0.0026115  
170170 ,0.0015132  
170187 ,0.0012811  
170204 ,0.0013096  
170221 ,0.0010075  
170238 ,0.00087006  
170255 ,0.00012189  
1870 ,0.0031671  
18717 ,0.0026808  
18734 ,0.0022504  
18751 ,0.0020583  
18768 ,0.0025304  
18785 ,0.0029794  
187102 ,0.003727  
187119 ,0.0029838  
187136 ,0.0031781  
187153 ,0.0022109  
187170 ,0.0012811  
187187 ,0.0010846  
187204 ,0.0011088  
187221 ,0.00085297  
187238 ,0.00073661  
187255 ,0.00010319  
2040 ,0.0032376  
20417 ,0.0027405  
20434 ,0.0023006  
20451 ,0.0021042  
20468 ,0.0025867  
20485 ,0.0030457  
204102 ,0.0038099  
204119 ,0.0030502  
204136 ,0.0032488  
204153 ,0.0022602  
204170 ,0.0013096  
204187 ,0.0011088  
204204 ,0.0011334  
204221 ,0.00087197  
204238 ,0.00075301  
204255 ,0.00010549  
2210 ,0.0024907  
22117 ,0.0021083  
22134 ,0.0017698  
22151 ,0.0016187  
22168 ,0.00199  
22185 ,0.0023431  
221102 ,0.002931  
221119 ,0.0023465  
221136 ,0.0024994  
221153 ,0.0017388  
221170 ,0.0010075  
221187 ,0.00085297  
221204 ,0.00087197  
221221 ,0.00067081  
221238 ,0.0005793  
221255 ,8.1153e-05  
2380 ,0.0021509  
23817 ,0.0018206  
23834 ,0.0015284  
23851 ,0.0013979  
23868 ,0.0017185  
23885 ,0.0020234  
238102 ,0.0025312  
238119 ,0.0020264  
238136 ,0.0021584  
238153 ,0.0015015  
238170 ,0.00087006  
238187 ,0.00073661  
238204 ,0.00075301  
238221 ,0.0005793  
238238 ,0.00050027  
238255 ,7.0082e-05  
2550 ,0.00030132  
25517 ,0.00025505  
25534 ,0.00021411  
25551 ,0.00019583  
25568 ,0.00024074  
25585 ,0.00028346  
255102 ,0.00035459  
255119 ,0.00028388  
255136 ,0.00030237  
255153 ,0.00021035  
255170 ,0.00012189  
255187 ,0.00010319  
255204 ,0.00010549  
255221 ,8.1153e-05  
255238 ,7.0082e-05  
255255 ,9.8178e-06

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

% Convert the pair keys to strings that can be used as symbols

pairKeys = keys(pairProbabilities);

symbols = 1:length(pairKeys); % Assigning an index to each unique pair

probabilities = cell2mat(values(pairProbabilities));

% Encode the second order image using Huffman coding

[numericDict, avglen] = huffmandict(symbols, probabilities);

% Map the numeric symbols back to the pairs

pairDict = cell(size(numericDict, 1), 2);

for i = 1:size(numericDict, 1)

pairDict(i, 1) = pairKeys(numericDict{i, 1});

pairDict(i, 2) = numericDict(i, 2);

end

disp('Second Order Huffman Encoding:');

disp(pairDict);

% Calculate entropy

second\_order\_entropy = -dot(probabilities, log2(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, numericDict(:, 2));

second\_order\_mean\_length = dot(huffman\_code\_lengths, 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: 5.7878

Mean Length of Second Order Huffman Encoding: 5.8126

Efficiency of Second Order Huffman Encoding: 0.99573

**Συμπέρασμα**

Παρατηρούμε ότι η απόδοση του κώδικα είναι ακόμα πιο κοντά στο 100% (99.57%), 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.** 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 Ν χωρίς μνήμη αποδεικνύεται ότι:

**H(XN) = NH(X) (ΙV)**

**(3) a.** Επομένως, για Ν = 2 η σχέση **(ΙV)** γίνεταιH(X2) = 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 **(ΙV)** δεν ισχύει στη προκειμένη περίπτωση.

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

**H(XN) LN H(XN) + 1 (V)**

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

* **Ν = 1**: H(X) L1 H(X) + 1 => **3.7831 L1 4.7831**
* **Ν = 2**: H(X2) L2 H(X2) + 1 => **7.5662 L2 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 = (VI)**

**(4)** Αρχικά κωδικοποιούμε την 1ης τάξης πηγή μας μέσω της συνάρτησης [[**code] = huffmanenco(sig,dict)**](https://www.mathworks.com/help/comm/ref/huffmanenco.html#d126e56456)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)**](https://www.mathworks.com/help/comm/ref/huffmandeco.html#d126e55758)πάνω στο encoded\_image και αποθηκεύσουμε το αποκωδικοποιημένο διάνυσμα στο *decompressed\_image\_vector* και μετά μέσω της **[**[**B] = reshape(A,sz)**](https://www.mathworks.com/help/matlab/ref/reshape.html#d126e1410905) 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:

Εικόνα που περιέχει πτηνό, παπαγάλος, ασπρόμαυρο, ράμφος

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει πτηνό, παπαγάλος, ασπρόμαυρο, ράμφος

Περιγραφή που δημιουργήθηκε αυτόματα**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(\_\_\_)**](https://www.mathworks.com/help/matlab/ref/double.numel.html#d126e1082414)παίρνουμε το πλήθος των δυαδικών ψηφίων της encoded\_image και του *image\_vector*, πάνω στο οποίο καλούμε βέβαια και την [[**binStr] = dec2bin(D)**](https://www.mathworks.com/help/matlab/ref/dec2bin.html#d126e353433)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:');

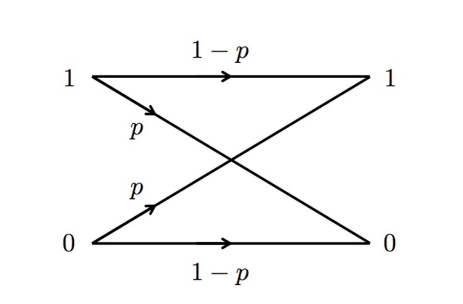
disp(num2str(j));

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.

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 to 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)** 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 Χ, 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(Xj) και συγκεκριμένα στο BSC ορίζεται ως:

**C(p) = 1 – Hb(p) (VIΙ)**

όπου: Hb(p) = –plog2p – (1 – p)log2(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:

**I(X; Y) = Η(Υ) – Η(Υ|Χ) (VIΙΙ)**

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

Η(Υ|Χ) 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));

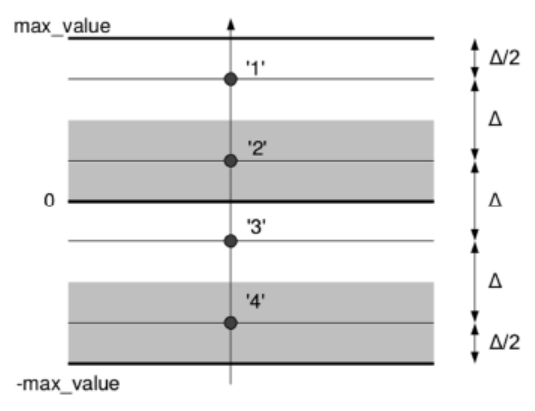
joint\_information = output\_entropy - conditional\_entropy;

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 2Ν 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 2Ν. Έπειτα, υπολογίζουμε τα κέντρα των επιπέδων, τα οποία θα είναι τοποθετημένα στο διάστημα [min\_value + , min\_value + , …, max\_value – ].

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. Έπειτα, αρχικοποιούμε το κβαντισμένο διάνυσμα *y\_hat* και για να «κόψουμε» τις εκτός εύρους τιμές αποθηκεύουμε στο *y\_clipped* το ελάχιστο των τιμών max\_value και max(y, min\_value) για κάθε δείγμα του *y*. Έτσι, για την εύρεση του κέντρου κβάντισης που είναι πλησιέστερο στην τιμή του κάθε δείγματος αποθηκεύουμε ως δείκτη για κάθε δείγμα την ελάχιστη από όλες τις αποστάσεις του *y\_clipped* από τα κέντρα του επιπέδου που ανήκει το δείγμα. Στη συνέχεια, τον αποθηκεύουμε στο διάνυσμα με τα κέντρα του επιπέδου και έπειτα όλο αυτό το αποτέλεσμα στο αντίστοιχο στοιχείο του *y\_hat.*

function [y\_hat] = my\_quantizer(y, N, min\_value, max\_value)

% Quantization step

delta = (max\_value - min\_value) / (2^N - 1);

% Calculation of centers of quantization area

centers = min\_value + delta/2 : delta : max\_value - delta/2;

% Pre-allocate y\_hat

y\_hat = zeros(size(y));

% Process each element in y

for i = 1:length(y)

% Limiting the dynamic range of the current element

y\_clipped = min(max(y(i), min\_value), max\_value);

% Calculating the area to which the current element belongs

[~, index] = min(abs(y\_clipped - centers));

% Mapping the current element to the center of the respective area

y\_hat(i) = centers(index);

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 ’(n) είναι η πρόβλεψη, τότε θα δίνεται από τη σχέση:

**’(n) = ’(n – i) (IX)**

όπου ai είναι οι συντελεστές κβάντισης.

Οι συντελεστές κβάντισης είναι το διάνυσμα *a* διάστασης px1 που προκύπτει λύνοντας το σύστημα εξισώσεων Yule – Walker **R*a* = *r* => *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);

% Create autocorrelation matrix R and autocorrelation vector r

for i = 1:p

for j = 1:p

for n = max(i, j):N

R(i, j) = R(i, j) + x(n) \* x(n - abs(i-j));

end

end

end

for k = 1:p

for n = k+1:N

r(k) = r(k) + x(n) \* x(n - k);

end

end

% Quantize coefficients a with N=8 bits and dynamic range [-2,2]

a = R\r;

a\_quantized = my\_quantizer(a, 8, -2, 2);

% Initialize y(n), y\_hat\_prediction(n) and y\_hat(n)

y = zeros(N, 1);

y\_hat = zeros(N, 1);

y\_hat\_prediction = zeros(N, 1);

% Calculate the first p values

for i = 1:p

y\_hat\_prediction(i) = x(i);

% Calculate the error and quantize it

y(i) = x(i) - y\_hat\_prediction(i);

y\_hat(i) = my\_quantizer(y(i), quantization\_bits, quantization\_min\_value, quantization\_max\_value);

end

% Calculate the next values from p+1 to N

for i = p+1:N

% Calculate the prediction y\_hat'(n) using previous y\_hat\_prediction values

y\_hat\_prediction(i) = a\_quantized.' \* y\_hat\_prediction(i-1:-1:i-p);

% Calculate the error and quantize it

y(i) = x(i) - y\_hat\_prediction(i);

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(i) = y\_hat\_prediction(i) + y\_hat(i);

end

end

Αρχικά, ορίζουμε τα μητρώο R και το διάνυσμα *r* με τις αντίστοιχες διαστάσεις. Μετά αποθηκεύουμε στο διάνυσμα *a* το γινόμενο R-1*r* και αφού το κβαντίσουμε με Ν = 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* και *y\_hat\_prediction* που αντιπροσωπεύουν το σήμα σφάλματος, το κβαντισμένο σήμα σφάλματος και την πρόβλεψη του σήματος *x* αντίστοιχα. Για τις πρώτες p τιμές η πρόβλεψη ισούται με τη τιμή του *x*, οπότε αποθηκεύουμε στο *y*(1:p) το σφάλμα πρόβλεψης και στο *y\_hat*(1:p) το κβαντισμένο σφάλμα πρόβλεψης. Για τις επόμενες τιμές από p+1 ως Ν υπολογίζουμε το εσωτερικό γινόμενο του *a\_quantized*(1:p) με το *y\_hat\_prediction*(i-1:-1:i-p), δηλαδή τους p συντελεστές κβάντισης με τις προηγούμενες p γνωστές τιμές, σύμφωνα με τον τύπο **(IX)**. Όμοια με πριν, υπολογίζουμε το σήμα σφάλματος, το κβαντίζουμε και τέλος ενημερώνουμε τη πρόβλεψη προσθέτοντας το κβαντισμένο αυτό σφάλμα στην ήδη υπάρχουσα πρόβλεψη. 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 ανακατασκευασμένουσήματος 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);

% Calculate the first p values

% This assumes that these values are directly transmitted without prediction.

for i = 1:p

x\_reconstructed(i) = y\_hat(i);

end

% Calculate the next values from p+1 to N

for i = p+1:N

% Calculate the prediction y\_hat\_prediction(n) using the quantized coefficients a\_quantized

% and the previously reconstructed values

y\_hat\_prediction = a\_quantized.' \* x\_reconstructed(i-1:-1:i-p);

% Add the prediction to the quantized error to reconstruct x

x\_reconstructed(i) = y\_hat\_prediction + y\_hat(i);

end

end

Ξεκινάμε αρχικοποιώντας το διάνυσμα *x\_recontructed* που αντιπροσωπεύει το ανακατασκευασμένο σήμα εισόδου *x*. Για τις πρώτες p τιμές η πρόβλεψη ισούται με τη τιμή του *y\_hat*, οπότε αποθηκεύουμε στο *x\_reconstructed*(1:p) το κβαντισμένο σφάλμα πρόβλεψης *y\_hat*(1:p). Για τις επόμενες τιμές από p+1 ως Ν όμοια με πριν υπολογίζουμε το εσωτερικό γινόμενο του *a\_quantized*(1:p) με το *x\_reconstructed*(i-1:-1:i-p), δηλαδή τους p συντελεστές κβάντισης με τις προηγούμενες p γνωστές τιμές. Τέλος, ενημερώνουμε το ανακατασκευασμένο σήμα προσθέτοντας τη πρόβλεψη του στο ήδη υπάρχον κβαντισμένο σφάλμα. 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 ≥ 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 τους συντελεστές κβάντισης, το σήμα σφάλματος, το κβαντισμένο σφάλμα και τη πρόβλεψη στα διανύσματα *a\_quantized*, *y*, *y\_hat* και *y\_hat\_prediction* αντίστοιχα. Με τις κατάλληλες συναρτήσεις της MATLAB σχεδιάζουμε στο ίδιο γράφημα τα αρχικό σήμα και το σήμα σφάλματος.

% Questions 2, 4

for p = 5:5:10

for N = 1:3

[a\_quantized, y, y\_hat, y\_hat\_prediction] = dpcm\_encoder(x, p, N, min\_value, max\_value);

figure

title(['Prediction error and reconstructed signal for p=', num2str(p), ' and N=', num2str(N)]);

subplot(3,1,1); % Create a subplot for signal y

plot(y);

xlabel('Prediction Error');

subplot(3,1,2); % Create a subplot for signal x

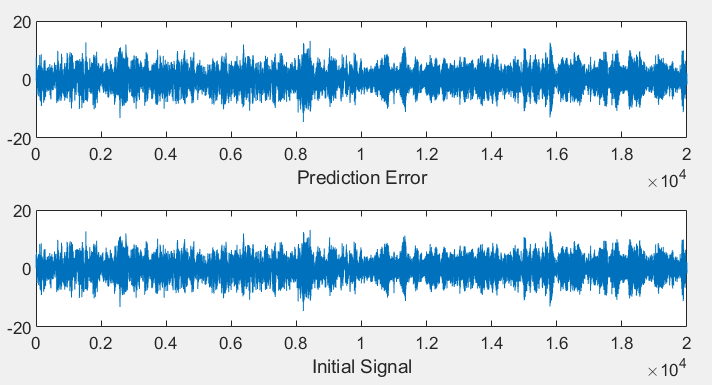
plot(x);

xlabel('Initial Signal');

end

end

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, γράφημα

Περιγραφή που δημιουργήθηκε αυτόματαΟι γραφικές παραστάσεις που λαμβάνουμε είναι οι εξής:

**Εικόνα 3**: p=5, N=1 **Εικόνα 4**: p=5, N=2

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:

**MSE = E[y2(n)] = E[(x(n) – ’(n))2] = E[(x(n) – ’(n – i))2] (X)**Αρχή φόρμας

Γενικά, η απόδοση της πρόβλεψης επηρεάζεται 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 απόδοσή 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 οποία 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.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, διάγραμμα, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα**Εικόνα** **9**: MSE για N = [1:3]

**Συμπέρασμα**

Παρατηρούμε όντως ότι το Μέσο Τετραγωνικό Σφάλμα ακολουθεί μια φθίνουσα πορεία καθώς το πλήθος Ν των δυαδικών ψηφίων κβάντισης αυξάνεται, ξεκινώντας από τη μέγιστη τιμή για Ν = 1 και τείνοντας στο 0 για πολύ μεγάλες τιμές του Ν. Αυτό επιβεβαιώνεται και από τις πιο κάτω γραφικές παραστάσεις όπου μελετάμε και περιπτώσεις με μεγαλύτερη τιμή Ν από 3.

Επιπλέον, παρατίθενται πιο κάτω και οι τιμές των συντελεστών κβάντισης του διανύσματος *a\_quantized* για κάθε τιμή του παράγοντα p:

**p =5**

1.2863

-1.5843

0.9882

-0.5490

-0.0314

**p =6**

1.2863

-1.5686

0.9412

-0.4706

-0.0784

0.0471

**p =7**

1.2706

-1.5216

1.1922

-0.9569

0.7059

-0.6118

0.5020

**p =8**

1.0824

-1.3020

0.9412

-0.6275

0.2980

-0.0784

0.0627

0.3451

**p =9**

1.1451

-1.3020

0.9255

-0.5804

0.1882

0.0784

-0.1412

0.5176

-0.1569

**p =10**

1.1137

-1.1765

0.8941

-0.5490

0.2353

-0.0627

0.0627

0.2196

0.1098

-0.2353