ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Ακαδημαϊκό Έτος 2018-2019
1η Εργαστηριακή Άσκηση

Χρήστος Γκουρνέλος
ΑΜ : 5744
cgkournelos@ceid.upatras.gr

13 Ιανουαρίου 2019

Μέρος Α – Κωδικοποίηση *Huffman*

Στο πρώτο μέρος της εργασίας καλούμαστε να υλοποιήσουμε ένα σύστημα κωδικοποίησης/αποκωδικοποίησης μιας πηγής χαρακτήρων κειμένου βασισμένο στον κώδικα *Huffman* . Ειδικότερα ο κώδικας αυτός αποτελεί έναν βέλτιστο *(ελάχιστο μέσο μήκος κώδικα)* προθεματικό κώδικα και χρησιμοποιείται ευρέως για συμπίεση αρχείων ή κωδικοποίησή διαφόρων πηγών χωρίς απώλειες *(lossless)*. Μέτα το τέλος της υλοποίησης, θα εξετάσουμε την αποτελεσματικότητα του συστήματος τόσο στην χρήση «τεχνητών» πηγών (τυχαία παραγόμενοι χαρακτήρες) όσο και πραγματικών πηγών (αρχείου Αγγλικών λέξεων). *Το περιβάλλον υλοποίησής και δοκιμών είναι η* `MATLAB` .

Παρακάτω παρουσιάζονται αναλυτικά τα αποτελέσματα του κάθε υποερωτήματος:

– Υλοποίησή των συναρτήσεων για την κωδικοποίηση *Huffman*

Στο ερώτημα αυτό υλοποιήθηκαν 3 συναρτήσεις σχετικά με την κωδικοποίηση *Huffman* και κάποιες επιπλέον βοηθητικές συναρτήσεις. Αναλυτικά:

(α′) Η συνάρτηση για τον υπολογισμό των κωδικών λέξεων είναι η `myHuffmanDict`. Παίρνει σαν όρισμα 2 διανύσματα το αλφάβητο και το διάνυσμα με τις πιθανότητες εμφάνισης κάθε χαρακτήρα. Στην έξοδο επιστρέφει μια δομή (*"λεξικό"*) που περιέχει έναν πίνακα με τους χαρακτήρες του αλφαβήτου καθώς και τον πίνακα με τις αντίστοιχες κωδικές λέξεις.
Για την ταξινόμηση μιας δομής που περιέχει παραπάνω από έναν πίνακες υλοποιήθηκε η συνάρτηση `mySortStruct`.

(β′) Η συνάρτηση για την κωδικοποίηση είναι η `myHuffmanEnco`. Ως είσοδο αυτή η συνάρτηση παίρνει την συμβολοσειρά που πρέπει να κωδικοποίησή και το λεξικό που έχει παραχθεί από την προηγούμενη συνάρτηση. Σαν έξοδο έχει την κωδική ακολουθία από 0 και 1.

(γ′) Η συνάρτηση για την αποκωδικοποίηση είναι η `myHuffmanDeco`. Ως είσοδο αυτή η συνάρτηση παίρνει το κωδικοποιημένο σήμα και το λεξικό που δημιούργησε η `myHuffmanDict`. Στην έξοδο επιστρέφει την αποκωδικοποιημένη συμβολοσειρά στο κανονικό αλφάβητο.

Σημείωση: Για τις παραπάνω συναρτήσεις θα βρείτε το κώδικα στο Παράρτημα Α, καθώς και τα ".m" αρχεία στον φάκελο `"../matlab/huffman"`.

– Επαλήθευση κωδικοποίησης για τις πηγές Α και Β

Σε συνέχεια της υλοποίησης του συστήματος κωδικοποίησης *Huffman* εξετάζεται η ορθότητα του για 2 διαφορετικές πηγές **Α** και **Β**.

Γράμμα	Πιθανότητα εμφάνισης
a	0.0817
b	0.0149
c	0.0278
d	0.0425
e	0.1270
f	0.0223
g	0.0202
h	0.0609
i	0.0697
j	0.0015
k	0.0077
l	0.0403
m	0.0241
n	0.0675
o	0.0751
p	0.0193
q	0.0095
r	0.0599
s	0.0633
t	0.0906
u	0.0276
v	0.0097
w	0.0236
x	0.0015
y	0.0197
z	0.0074

Πίνακας 1: Πιθανότητές εμφάνισης γραμμάτων στην Αγγλική γλώσσα [1]

◇ Η πηγή **A** θεωρούμε πως είναι μια τεχνητή πηγή, και πιο συγκεκριμένα μια διακριτή πηγή χωρίς μνήμη που παράγει πεζούς χαρακτήρες του Αγγλικού αλφάβητου με βάση συγκεκριμένες πιθανότητες. Για τις πιθανότητες αυτές χρησιμοποιήθηκαν οι τιμές εμφάνισης βάση του Πίνακα 1. Με την χρήση της συνάρτησης `randsrc` που παρέχει η MATLAB δημιουργήθηκε τυχαία η πηγή των 10.000 χαρακτήρων. Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα για το Αγγλικό αλφάβητο με αυτές τις πιθανότητες ισούται με:
$$\overline{L}_A = 4,2050$$
2. Η μέγιστη εντροπία της πηγής με 26 χαρακτήρες ισούται με:
$$H(\cdot)_{max} = 4,7004$$
3. Η εντροπία της πηγής **A** με αυτές τις πιθανότητες ισούται με:
$$H(\cdot)_A = 4,1757$$
4. Η αποδοτικότητα του *Huffman* ισούται με:
$$\eta_A = \frac{H(\cdot)_A}{\overline{L}_A} = 0,9930$$
5. Η κωδικοποίηση και η αποκωδικοποίηση γίνεται σωστά, ελέγχοντας αν ισούται το αποτέλεσμα της `myHuffmanDeco` με την αρχική πηγή.

[1] Τα στοιχεία προέρχονται απ' τη Wikipedia.

6. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας[2] για την πηγή **A**:

Αρχικό Μέγεθος (kB) [3]	Μέγεθος κωδ/νης ακολουθίας (kB) [4]
10	5,23

7. Τέλος αυτό που παρατηρήθηκε είναι ότι αν στην `randsrc` δεν δοθεί σαν όρισμα ο πίνακας πιθανοτήτων, η παραγόμενη πηγή θα έχει ίδια πιθανότητα για όλα τα σύμβολά (1/26). Αυτή η διάφορα εκφράζεται στην κωδικοποίηση διότι πλέον έχουμε μέγεθος κωδικοποιημένης πηγής περίπου ίσο με $6,8$ kB.

◇ Η πηγή **B** είναι ένα αρχείο το οποίο δίνεται και το οποίο περιέχει 3.857 Αγγλικές λέξεις οι οποίες ξεκινούν από το χαρακτήρα κ. Εδώ να σημειωθεί ότι στο αρχείο αυτό υπήρχαν κάποια επιπλέον σύμβολα που αφαιρέθηκαν και κάποιοι κεφαλαίοι χαρακτήρες που αντικαταστάθηκαν με τους αντίστοιχους πεζούς.Επίσης το αρχείο μετατράπηκε σε μια ενιαία συμβολοσειρά χωρίς του χαρακτήρες αλλαγής γραμμής, ώστε να μπορεί να δοθεί σαν είσοδο στις συναρτήσεις για την κωδικοποίηση *Huffman* . Ο λόγος αυτής της τροποποίησης στην πηγή **B** είναι για να μπορεί να κωδικοποιηθεί με το ίδιο λεξικό που χρησιμοποιήθηκε για την πηγή **A**.
Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα, η εντροπία και η αποδοτικότητα είναι το ίδιο με πριν καθώς το αλφάβητο και οι πιθανότητες δεν άλλαξαν.
2. Η κωδικοποίηση και η αποκωδικοποίηση γίνεται σωστά, ελέγχοντας αν ισούται το αποτέλεσμα της `myHuffmanDeco` με την αρχική πηγή.
3. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας για την πηγή **B**:

Αρχικό Μέγεθος (kB)	Μέγεθος κωδ/νης ακολουθίας (kB)
29,11	16,93

Σημείωση: Για τις παραπάνω παρατηρήσεις χρησιμοποιήθηκαν τα scripts `random_source_test` *και* `kwords_source_test` *θα βρείτε το κώδικα στο Παράρτημα Α, καθώς και τα ".m" αρχεία στον φάκελο* `"../matlab/huffman/test"`.

– Κωδικοποίηση της πηγής B με ανανεωμένες πιθανότητες συμβολών

Προτείνεται να επανυπολογιστούν οι πιθανότητες του κάθε συμβόλου για την πηγή **B** με βάση το ίδιο το αρχείο kwords.txt.Περιμένουμε, το μέσο μήκος κωδικοποίησης να είναι πιο μικρό απ' ότι στο προηγούμενο ερώτημα, αλλά και η κωδικοποιημένη ακολουθία να έχει μικρότερο μέγεθος από πριν. Αυτό πρέπει να συμβεί, γιατί η λογική του κώδικα *Huffman* στηρίζεται στο ότι τα σύμβολα με μεγαλύτερη συχνότητα εμφάνισης θα έχουν μικρότερες κωδικές λέξεις έτσι ώστε μια ακολουθία να έχει όσο το δυνατόν μικρότερη κωδική απεικόνιση, αφού οι πιο συχνοί χαρακτήρες έχουν μικρότερες κωδικές απεικονίσεις απ' τους άλλους. Οι νέες πιθανότητες που προκύπτουν παρουσιάζονται στον Πίνακα 2. Όπως φαίνεται η διαφορά στη πιθανότητα για το γράμμα "k" είναι μεγάλη.
Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα με τις νέες πιθανότητες ισούται με:

$$\overline{L}_B = 4,0869$$

2. Η εντροπία της πηγής **B** με τις νέες πιθανότητες ισούται με:

$$H(\cdot)_B = 4,0482$$

[2] Μέσος όρος 10 δοκιμών με διαφορετικές τυχαίες πηγές
[3] Τα σύμβολα της πηγής αρχικά είναι τύπου `char` (1 Byte = 8 bits)
[4] Τα σύμβολα της κωδ/νης ακολουθίας καταλαμβάνουν 1bit ανά χαρακτήρα στην μνήμη

Γράμμα	Πιθανότητα εμφάνισης
a	0.0872
b	0.0147
c	0.0229
d	0.0227
e	0.1053
f	0.0068
g	0.0201
h	0.0278
i	0.0905
j	0.0020
k	0.1562
l	0.0500
m	0.0212
n	0.0715
o	0.0629
p	0.0189
q	0.0001
r	0.0514
s	0.0584
t	0.0517
u	0.0213
v	0.0044
w	0.0079
x	0.0008
y	0.0193
z	0.0041

Πίνακας 2: Πιθανότητές εμφάνισης γραμμάτων στο αρχείο kwords.txt

3. Η αποδοτικότητα του *Huffman* για την πηγή **B** ισούται με:

$$\eta_B = \frac{H(\cdot)_B}{\overline{L}_B} = 0,9905$$

4. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας για την πηγή **B**:

Αρχικό Μέγεθος (kB)	Μέγεθος κωδ/νης ακολουθίας (kB)
29.11	14.87

Παρόλο την μικρή μείωση που παρατηρείται στην αποδοτικότητα του *Huffman* για τις καινούργιες πιθανότητες, το μέγεθος της κωδικοποιημένης πηγής που προκύπτει είναι μικρότερο σε σχέση με πριν κατά 2,06 kB.

– Δεύτερης τάξης επέκταση της πηγής **A**

Στο υποερώτημα αυτό με τη βοήθεια της συνάρτησης `myNOrderSourceGen` (βλ. Παράρτημα Α), μετατρέψαμε την πηγή **A** σε 2ης τάξης. Πλέον αντί για 26 σύμβολα αλφάβητο, υπάρχουν 676 ζεύγη χαρακτήρων με νέες πιθανότητές. Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα για το Αγγλικό αλφάβητο με αυτές τις πιθανότητες ισούται με:

$$\overline{L}_{A^2} = 8,3819$$

2. Η εντροπία της πηγής **A** με αυτές τις πιθανότητες ισούται με:

$$H(\cdot)_{A^2} = 8,3516$$

3. Η αποδοτικότητα του *Huffman* ισούται με:

$$\eta_{A^2} = \frac{H(\cdot)_{A^2}}{\overline{L}_{A^2}} = 0,9964$$

4. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας για την δεύτερης τάξης επέκταση της πηγή **A**:

Αρχικό Μέγεθος (kB)	Μέγεθος κωδ/νης ακολουθίας (kB)
10	5.23

Σε σχέση με την 1ης τάξης πηγή είναι αναμενόμενο οι τιμές να είναι διπλάσιες καθώς επίσης από την θεωρία θα πρέπει να ισχύει η εξής ανισότητα επειδή η A είναι πηγή χωρίς μνήμη :

$$H(\cdot)_A \leq \frac{\overline{L}_{A^2}}{2} \leq H(\cdot)_A + \frac{1}{2} \Leftrightarrow$$
$$4,1757 \leq \frac{8,3819}{2} \leq 4,1757 + 0,5 \Leftrightarrow$$
$$4,1757 \leq 4.1910 \leq 4,6757$$

Επίσης βελτιώθηκε και η συμπίεση κατά 1,59 kB.

– Δεύτερης τάξης επέκταση της πηγής B

Όπως και στο προηγούμενο ερώτημα, μετατρέψαμε την πηγή **B** σε 2ης τάξης.

◇ Αρχικά η πηγή κωδικοποιήθηκε με τις πιθανότητες όπως έχουν υπολογισθεί για την πηγή **A**. Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα, η εντροπία και η αποδοτικότητα είναι το ίδιο με πριν καθώς το αλφάβητο και οι πιθανότητες δεν άλλαξαν.
2. Η κωδικοποίηση και η αποκωδικοποίηση γίνεται σωστά, ελέγχοντας αν ισούται το αποτέλεσμα της `myHuffmanDeco` με την αρχική πηγή.
3. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας για την δεύτερης τάξης επέκταση της πηγή **B**:

Αρχικό Μέγεθος (kB)	Μέγεθος κωδ/νης ακολουθίας (kB)
29,11	16,80

◇ Στην συνέχεια η πηγή κωδικοποιήθηκε χρησιμοποιώντας τις πιθανότητες των ζευγών χαρακτήρων απ' το αρχείο κωορδς. Το νέο αλφάβητο περιέχει μόνο τα ζευγάρια που υπάρχουν στο αρχείο συνοδευόμενα με τις πιθανότητες εμφάνισης τους. Παρατηρήσεις:

1. Το μέσο μήκος του κώδικα για το Αγγλικό αλφάβητο με αυτές τις πιθανότητες ισούται με:

$$\overline{L}_{B^2} = 7,5389$$

2. Η εντροπία της πηγής **A** με αυτές τις πιθανότητες ισούται με:

$$H(\cdot)_{B^2} = 7.5127$$

3. Η αποδοτικότητα του *Huffman* ισούται με:

$$\eta_{B^2} = \frac{H(\cdot)_{B^2}}{\overline{L}_{B^2}} = 0,9965$$

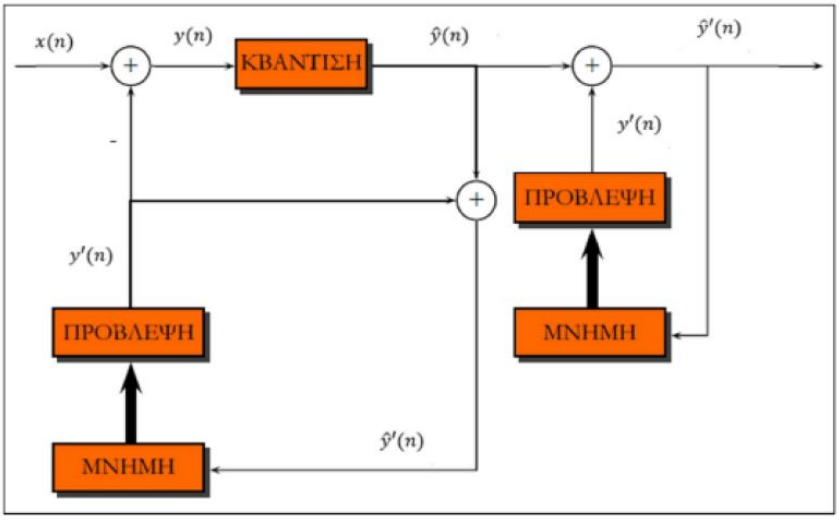
4. Μεγέθη αρχικής και κωδικοποιημένης ακολουθίας για την δεύτερης τάξης επέκταση της πηγή **B**:

Αρχικό Μέγεθος (kB)	Μέγεθος κωδ/νης ακολουθίας (kB)
29,11	13,71

Συγκρίνοντας τα στοιχεία απ' τους δύο τελευταίους πίνακες, κατ' αρχάς παρατηρούμε ότι το μέσο μήκος έχει μειωθεί, λόγω του ότι οι πιθανότητες προκύπτουν απ' το ίδιο το αρχείο. Στη συνέχεια βλέπουμε ότι η συμπίεση είναι καλύτερη στη δεύτερη περίπτωση με διαφορά 3,09 kB.

Μέρος Β – Κωδικοποίηση Διακριτής Πηγής με τη μέθοδο DPCM

Στο δεύτερο μέρος της εργασίας θα μελετήσουμε ένα σύστημα κωδικοποίησης *DPCM* όπως φαίνεται στο Σχήμα 1.



Σχήμα 1: Κωδικοποιητής και ο αποκωδικοποιητής ενός συστήματος *DPCM*

– Υλοποίηση συστήματος κωδικοποίησης/αποκωδικοποίησης *DPCM*

Η υλοποίηση έγινε, όπως και στο πρώτο μέρος στο περιβάλλον της `MATLAB` . Τα σχετικά αρχεία κώδικα παρουσιάζονται στο Παράρτημα Β και βρίσκονται στον φάκελο `"../matlab/dpcm"`. Αναλυτικά:

(α′) Για τον υπολογισμό των συντελεστών α_i του φίλτρου πρόβλεψης δημιουργήθηκε η συνάρτηση `my_predictor_factors`. Για το υπολογισμό αυτών χρησιμοποιήθηκε το κριτήριο ελαχιστοποίησης του μέσου τετραγωνικού σφάλματος ανάμεσα στο εκάστοτε τρέχον δείγμα εισόδου και την πρόβλεψή του.

(β′) Ο ομοιόμορφος Κβαντιστής υλοποιήθηκε με την συνάρτηση `my_quantizer`, όπως αναφέρεται στην εκφώνηση.

(γ′) Για την προσομοίωση του *DPCM* συστήματος (βλ. Σχήμα 1), φτιάχτηκαν δυο διαφόρετικες συναρτήσεις για τον πομπό (`my_dpcm_trans`) και για τον δέκτη (`my_dpcm_rec`). Κάποιες παραδοχές που αξίζει να σημειωθούν είναι οι εξής:

i. Για την αρχικοποίηση του συστήματος θεωρήθηκε ότι οι p πρώτες τιμές *(μέγεθος φίλτρου)* μεταδίδονται χωρίς σφάλματα.

ii. Οι συντελεστές στο φίλτρο πρόβλεψης θα κβαντίζονται στην δυναμική περιοχή $[-2, 2]$ σε 256 επίπεδα. Ο υπολογίσιμος και η κβάντιση γίνεται στον πομπό, και στη συνέχεια οι συντελεστές του φίλτρου πρόβλεψης κβαντίζονται και αποστέλλονται στο δέκτη.

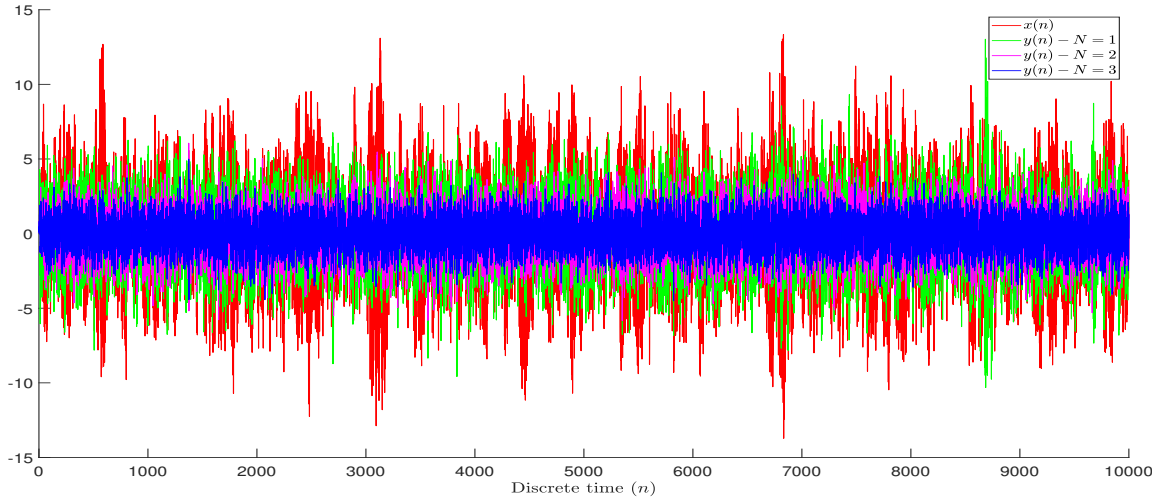
Και οι δυο συναρτήσεις επιστρέφουν ένα `struct` με τις μεταβλητές που συμμετέχουν στους υπολογισμούς. Ο Πικακάς 3 δείχνει τον συνδυασμό των μεταβλητών αυτών με την σημειογραφία της εκφώνησης και του Σχήματος 1.

Μεταβλητή MATLAB	**Μαθηματικός Συμβολισμός**
x	$x(n)$
y	$y(n)$
y_pred	$y'(n)$
y_quant	$\hat{y}(n)$
y_mem	$\hat{y}'(n)$
a_factors	a_i

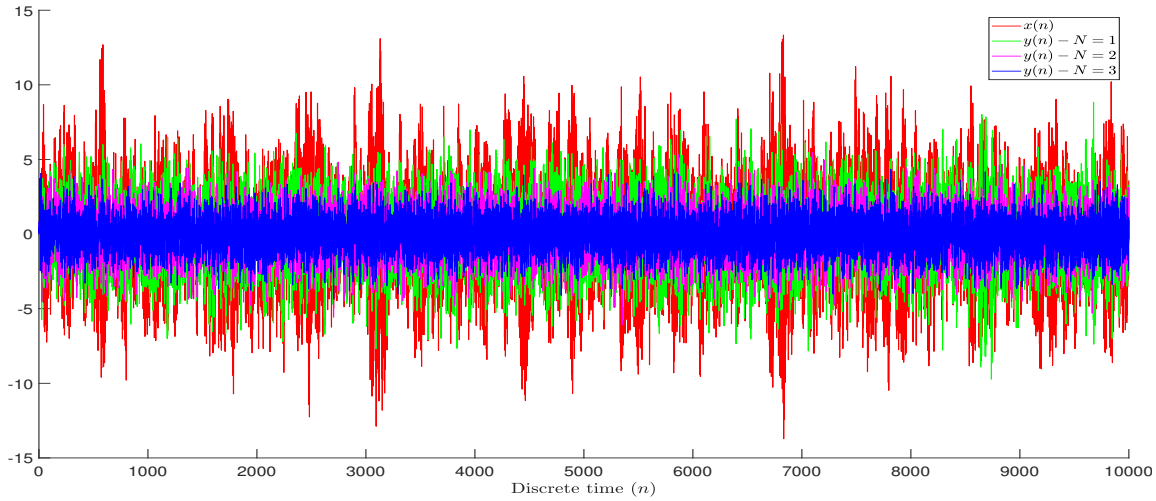
Πίνακας 3: Αντιστοίχηση συμβολισμών με τις συναρτήσεις MATLAB

– Διαγράμματα αρχικού σήματος και σφάλματος προβλέψεις

Για την αξιολόγηση της απόδοσης του προβλεπτή του συστήματος *DPCM* , δοκιμάστηκε για 2 τιμές του $p \geq 4$ και $N = 1, 2, 3$ bits. Συγκεκριμένα[5], επιλέχθηκαν οι τιμές $p = 4$ (βλ. Σχήμα 2) και $p = 24$ (βλ. Σχήμα 3).



Σχήμα 2: Διάγραμμα αρχικού σήματος και σφάλματος πρόβλεψης για $p = 4$ και $N = 1, 2, 3$ bits



Σχήμα 3: Διάγραμμα αρχικού σήματος και σφάλματος πρόβλεψης για $p = 24$ και $N = 1, 2, 3$ bits

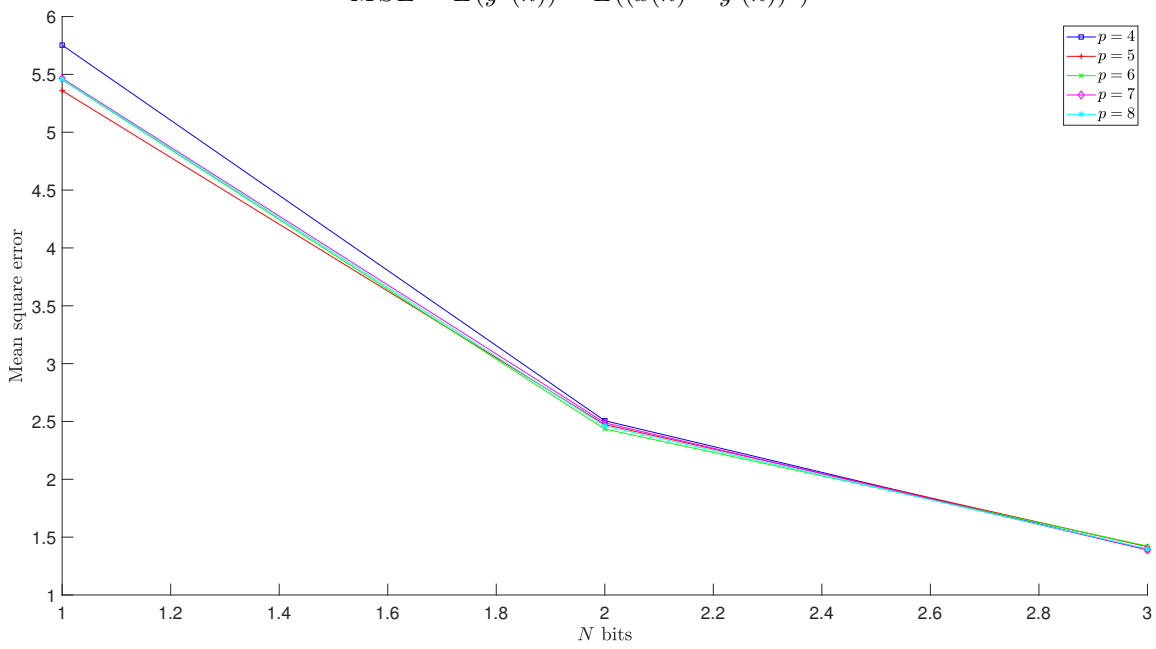
[5]Για την εξαγωγή των διαγραμμάτων εκτελέσαμε το script dpcm_predict_test.m

Όπως παρατηρείται εύκολα και στα δυο διαγράμματα, όσο μεγαλώνει το N το σφάλμα πρόβλεψης *"στενεύει"* δηλαδή μειώνεται. Αυτό σημαίνει ότι όσο πιο λεπτομερή κβάντιση κάνουμε τόσο ελαχιστοποιείται η διαφορά του σήματος εισόδου από την πρόβλεψη. Από την πλευρά η αύξηση του p κατά 20 τιμές βλέπουμε ότι δεν έχει σχεδόν καμία διαφορά και καμία επίδραση στο σφάλμα πρόβλεψης.

– Απόδοση *DPCM* και μέσο τετραγωνικό σφάλμα πρόβλεψης

Για την καλύτερη αξιολόγηση της απόδοσης του προβλέπτη, στο διάγραμμα[6]του στο Σχήματος , παρουσιάζεται το το μέσο τετραγωνικό σφάλμα πρόβλεψης ως προς το N και για διάφορες τιμές του p. Από το

$$MSE = E(y^2(n)) = E((x(n) - y'(n))^2)$$



Σχήμα 4: Διάγραμμα μέσου τετραγωνικού σφάλματος πρόβλεψης για ταξη προβλέπτη $p = 4 : 1 : 8$

παραπάνω γράφημα μπορούμε να συμπεράνουμε ότι η απόδοση του φίλτρου πρόβλεψης του σήματος επηρεάζεται σχεδόν αποκλειστικά από την ακρίβεια κβάντισης των δειγμάτων. Αυτο το συμπέρασμα προκύπτει απο το γεγονός ότι η γραφική παράσταση που σχεδιάστηκε είναι φθίνουσα σε όλες τις περιπτώσεις και κάθε φορά το σφάλμα σχεδόν υποδιπλασιάζεται με την αύξηση των bits. Οι συντελεστές του φίλτρου πρόβλεψης (α_i),

$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$
1.3887	1.3881	1.3880	1.3879	1.3878
-1.5212	-1.5185	-1.5191	-1.5193	-1.5192
1.2122	1.2087	1.2117	1.2109	1.2106
-0.3016	-0.2984	-0.3022	-0.2975	-0.2998
	-0.0025	0.0011	-0.0051	0.0050
		-0.0026	0.0033	-0.0095
			-0.0045	0.0073
				-0.0085

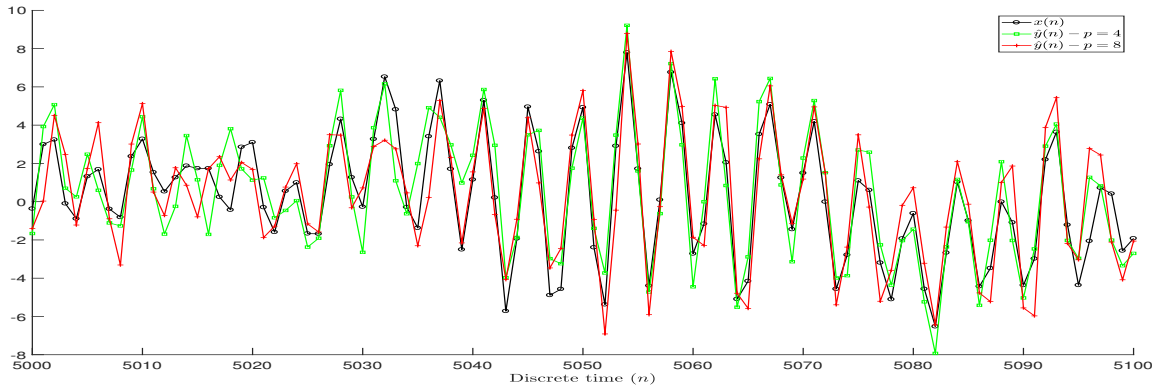
Πίνακας 4: Σεντλεστές πρόβλεψης α_i για κάθε τάξη p

παρουσιάζονται στον Πίνακα 4. Παρατηρούμε ότι κυμαίνονται γύρω από το 0 και ενώ οι αρχικοί είναι σχετικά μεγάλοι αριθμοί, όσο αυξάνεται το p και δημιουργούνται και άλλοι συντελεστές και σταθεροποιούνται σε αριθμό τάξης 10^{-3}.

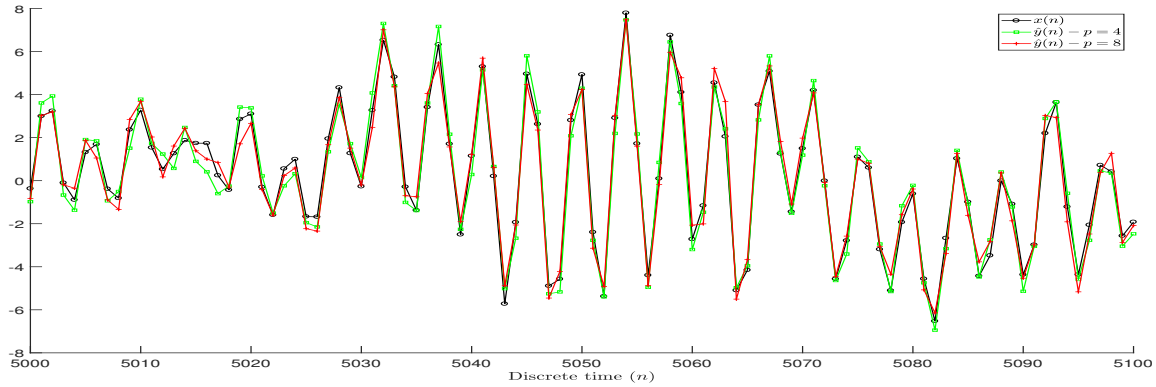
[6]Για την δημιουργία του διαγράμματος εκτελέσαμε το script `dpcm_mean_square_error.m`

– **Σύγκρισή αρχικού $x(n)$ και ανακατασκευασμένου $\hat{y}'(n)$ σήματος**

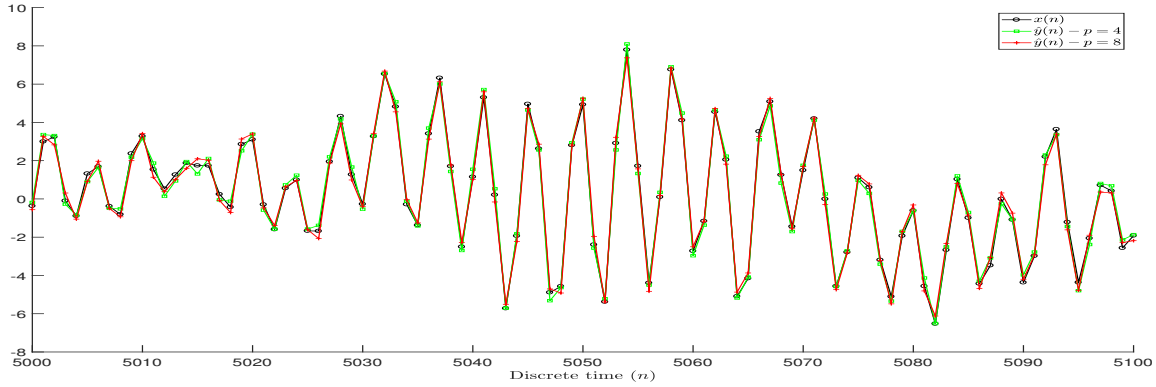
Στο ερώτημα αυτό γίνεται σύγκριση του αρχικού με το ανακατασκευασμένο σήμα. Παρακάτω παρουσιάζονται τα διαγράμματα[7]με το αρχικό και το ανακατασκευασμένο σήμα για $p = 4$ και $p = 8$. Κάθε σχήμα αφορά διαφορετικά bits κβάντισης $N = 1, 2, 3$ και για να μπορέσουμε να παρατηρήσουμε καλύτερα τις τιμές εχουμε πάρει τυχαια ενα παραθυρο 100 τιμών *(5000 - 5100)*.



Σχήμα 5: Διάγραμμα αρχικού και ανακατασκευασμένου σήματος και $N = 1$ bit



Σχήμα 6: Διάγραμμα αρχικού και ανακατασκευασμένου σήματος και $N = 2$ bit



Σχήμα 7: Διάγραμμα αρχικού και ανακατασκευασμένου σήματος και $N = 3$ bit

Παρατηρώντας τις μετρήσεις γίνεται προφανές ότι όσα περισσότερα βιτς χρησιμοποιούνται τόσο καλύτερη είναι η ανακατασκεή του αρχικού σήματος. Αυτό επιβεβαιώνει και τα συμπεράσματα απο τα προηγούμενα ερωτήματα, οτί μεγαλητερη ακρίβεια κβάντισης μικρότερο σφάλμα πρόβλεψης.

[7]Για την δημιουργία του διαγράμματος εκτελέσαμε το script `dpcm_reconstruct_signal.m`

Παράρτημα Α

Matlab scripts *Huffman* coding

- myHuffmanDict.m

```
% Author: Christos Gkournelos
% Date: 29/12/2018
%
% An custom implimentation of  Huffman dictionary generator
%
function [dict,avg_len] = myHuffmanDict(alphabet, probs, verbose)
  %
  %  Input error checking
  %
  if nargin < 2
    error('Wrong input. \nThe function requires 2 input vectors', -1)
  end
  if length(alphabet) ~= length(probs)
    error('Wrong input. \nThe length of alphabet and probalities does not match', -1)
  end
  if (~strcmp( class(probs), 'double') )
    error('Wrong input. \nThe probabilities vector must be of type double', -1)
  end
  if (min( probs ) < 0 )
    error('Wrong input. \nProbabilities of an input symbol cannot be negative', -1)
  end
  if (max( probs ) > 1 )
    error('Wrong input. \nProbabilities of an input symbol cannot be greater than ...
        1', -1)
  end
  %
  % Initialize  structs
  %
  for i = 1:length(alphabet)
    origin_index{i} = i;
    if isnumeric(alphabet(i))
      symbol{i} = num2str(alphabet(i));
    elseif (isstr(alphabet(i)) || ischar(alphabet(i)))
      symbol{i} = alphabet(i);
    elseif iscell(alphabet)
      symbol{i} = alphabet{i};
    else
      error('Wrong input. \nUnknown type of alpabet symbols. They must be chars ...
          vector or cell', -1)
    end % if
    code_table{i} = '';
  end %for
  dict.symbol = symbol;
  dict.code = code_table;
  dict.prob = probs;
  origin_struct = struct('symbol',{symbol}, 'prob', probs, 'ind', {origin_index});
  [sorted_struct index] = mySortStruct(origin_struct, 'prob');
  %
  % Main loop
  %
  while sorted_struct.prob ~= 1
    % Update
    updated_struct = myHuffMerge(sorted_struct, index);
    % Sort
    [sorted_struct index] = mySortStruct(updated_struct, 'prob');
  end % while
  avg_len = myHuffLength(dict);
  entro = myEntropy(dict.prob);
  if(verbose == 1)
    fprintf('Huffman coding atributes: \n\t - average length = %.4f \n\t - ...
        entropy = %.4f (max = %.4f) \n\t - efficiency = %.4f \n\n', avg_len ...
```

```
          ,entro, log2(length(alphabet)), (entro/avg_len));
59    end %if
60    %
61    % This function will update the symbols and the propabilities
62    %
63    function [updated_struct_] = myHuffMerge(s_, ¬)
64      first_symbol_ind_ = s_.ind{1};
65      dict.code = myHuffCodeUpdate(dict.code,first_symbol_ind_,'1');
66      second_symbol_ind_= s_.ind{2};
67      dict.code = myHuffCodeUpdate(dict.code,second_symbol_ind_,'0');
68      % Merge
69      merged_symbol_ = [s_.symbol{1} s_.symbol{2}];
70      merged_prob_   = s_.prob(1) + s_.prob(2);
71      merged_index_ = [s_.ind{1} s_.ind{2}];
72      % Clear
73      s_.symbol(1:2) = '';
74      s_.prob(1:2)   = '';
75      s_.ind(1:2)    = '';
76      % Update
77      s_.symbol = [s_.symbol merged_symbol_];
78      s_.prob = [s_.prob merged_prob_];
79      s_.ind = [s_.ind merged_index_];
80      updated_struct_ = s_;
81    end % fuction myHuffMerge
82    %
83    % This function will fill the code word on the dictionary
84    %
85    function [code_table_] = myHuffCodeUpdate(code_table_,input_ind_,new_code_)
86      for i = 1:length(input_ind_)
87          code_table_{input_ind_(i)} = strcat(new_code_, code_table_{input_ind_(i)});
88      end
89    end % function myHuffCodeUpdate
90    function [avg_len_] = myHuffLength(dict_)
91      avg_len_ = 0;
92      for i = 1:length(dict_.symbol)
93        avg_len_ = avg_len_ + dict_.prob(i)*length(dict_.code{i});
94      end % for
95    end % function myHuffLength
96    function entro = myEntropy(probs)
97      entro = sum(probs.*log2(1./probs));
98    end % myEntropy
99  end % funtion myHuffmanDict
```

- mySortStruct.m

```
1   % Author: Christos Gkournelos
2   % Date: 29/12/2018
3   %
4   % An custom implimentation for struct sorting
5   %
6   function [sorted_struct ind] = mySortStruct(s, field_name, direction)
7     %
8     %   ——— Input error checking ———
9     %
10    if ¬isstruct(s)
11      error('Wrong input: \nFirst input supplied is not a struct.', −1)
12    end % if
13    if ¬ischar(field_name) || ¬isfield(s, field_name)
14      error('Wrong input: \nSecond input is not a valid field_name.', −1)
15    end % if
16
17    % Direction by default is asceding
18    if nargin < 3
19      direction = 1;
20    elseif ¬isnumeric(direction) || numel(direction)>1 || ¬ismember(direction, [−1 1])
21      error('Wrong input. \nDirection must equal 1 for ascending order or −1 for ...
          descending order.')
22    end % if
```

```
    [dummy ind] = sort(s.(field_name));
    fields = fieldnames(s);
    for ii=1:length(fields)
      field_ = char(fields(ii));
      if length(ind) == length(s.(field_))
        sorted_struct.(field_) = s.(field_)(ind);
      end % if
    end % for ii
  end % mySortStruct
```

- myHuffmanEnco.m

```
% Author: Christos Gkournelos
% Date: 30/12/2018
%
% An custom implimentation of  Huffman encoding
%
function enco = myHuffmanEnco(sig , dict, verbose)
  %
  %   ---- Input error checking -----
  %
  if nargin < 2
    error('Wrong input. \nThe function requires 2 input vectors', -1)
  end
  [m,n] = size(sig);
  if ( m ≠ 1 && n ≠ 1)
      error('Wrong input. \nThe input signal must be a vector.', -1);
  end
  if ( ¬isstruct(dict) )
      error('Wrong input. \nThe input dictionary must be a struct.', -1);
  end
  %
  % Main loop
  %
  enco = [];
  for i = 1:length(sig) % signal iterator
    t_code = '';
    for j = 1:length(dict.symbol) % dictionary iterator
      if(strcmp(sig(i), dict.symbol{j}))
        t_code = dict.code{j};
        break;
      end % if
    end % for j
    if isempty(t_code)
        sig(i)
        error('One of the input signal characters is not incuded in Huffman ...
            dictionary.');
    else
        enco = [enco t_code];
    end % if
  end % for i
  if(verbose == 1)
    fprintf('Huffman encode atributes: \n\t- Input signal size = %.4f kB\n\t- ...
        Encoded signal size = %.4f kB\n\n', (length(sig)/1000), ...
        ((length(enco)/8)/1000));
  end %if
end % myHuffmanEnco
```

- myHuffmanDeco.m

```
% Author: Christos Gkournelos
% Date: 30/12/2018
%
% An custom implimentation of  Huffman decoding
%
function deco = myHuffmanDeco(comp , dict)
  %
```

```
8   %   ——— Input error checking ———
9   %
10  if nargin ≠ 2
11    error('Wrong input. \nThe function requires 2 input vectors', −1)
12  end
13  [m,n] = size(comp);
14  if ( m ≠ 1 && n ≠ 1)
15      error('Wrong input. \nThe input signal must be a vector.', −1);
16  end
17  if ( ¬isstruct(dict) )
18      error('Wrong input. \nThe input dictionary must be a struct.', −1);
19  end
20  %
21  % Main loop
22  %
23  deco = {};
24  code_pos = 1;
25  while (code_pos ≤ length(comp))
26    matches = dict;
27    temp_code = comp(code_pos);
28    not_found = 1;
29    while (not_found)
30      if (code_pos > length(comp))
31        matches = findMatches(temp_code, code_pos − 1, matches, 1);
32      else
33        matches = findMatches(temp_code, code_pos, matches);
34      end %if
35      if(length(matches.code) ≠ 1)
36        code_pos = code_pos + 1;
37        temp_code = comp(code_pos);
38      else
39        code_pos = 1;
40        found_symbol = matches.symbol;
41        not_found = 0;
42        comp = comp(length(matches.code{1})+1:end);
43      end % if
44    end % while not_found
45    deco = [deco found_symbol];
46  end % while lenght(comp)
47  deco = cell2mat(deco);
48  %
49  % This function will find the possible matches of the input signal with the dict
50  %
51  function matches_ = findMatches(code_, pos_, dict_ , code_len_)
52    if nargin == 3
53      code_len_ = 0;
54    end
55    matches_.symbol = {};
56    matches_.code = {};
57    j = 1;
58    for i = 1:length(dict_.code)
59      if (strcmp(dict_.code{i}(pos_), code_))
60        if(code_len_ == 0)
61          matches_.symbol(j) = dict_.symbol(i);
62          matches_.code(j) = dict_.code(i);
63          j = j + 1;
64        else
65          if(length(dict_.code{i}) == code_len_)
66            matches_.symbol(j) = dict_.symbol(i);
67            matches_.code(j) = dict_.code(i);
68            j = j + 1;
69          end %if
70        end %if
71      end % if
72    end %for
73  end % function findMatches
74 end % function myHuffmanDeco
```

Βοηθητικά Matlab scripts για την επαλήθευση των ερωτημάτων

- random_source_test.m

```
%
% This script is used for testing myHuffman implementation
% with a random english letters source
%
clear; clc;
load('alphabet_symbols.mat'); % alphabet : loads the letters of english alphabet
load('alphabet_probs.mat'); %probabilities : loads the probability of each letter ...
    based on wikepedia
letter_pos = 1:26; % index for letters in alphabet
%
% Genareate the random source_a
%
% random_pos_table = randsrc(10000,1,letter_pos);
random_pos_table = randsrc(10000,1,[letter_pos; cell2mat(probabilities)]);
source_a = char();
for i = 1:10000
  source_a(i) = alphabet(random_pos_table(i));
end % for
%
% Compute Hufmman dictionary
%
[dict, avg_code_len] = myHuffmanDict(alphabet, cell2mat(probabilities), 1);
%
% Encode source_a
%
enco = myHuffmanEnco(source_a, dict, 1);
%
% Decode again and check if the result is the same
%
deco = myHuffmanDeco(enco, dict);
if isequal(deco,source_a)
  disp('Perfect!! Our Huffman works fine with random sources.');
else
  error('Ooops maybe we missed something.');
end % if
```

- kwords_source_test.m

```
%
% This script is used for testing myHuffman implementation
% with kwords source
%
clear; clc;
load('alphabet_symbols.mat'); % alphabet : loads the letters of english alphabet
load('alphabet_probs.mat'); % probabilities : loads the probability of each ...
    letter based on wikepedia
load('kwords.mat'); % kwords : This table contains the letters from kwords.txt
%
% Fill the source_b vector
%
t_array = table2array(kwords);
source_b = char();
for i = 1:length(t_array)
  source_b = [source_b char(t_array(i))];
end % for
%
% Compute Hufmman dictionary
%
[dict, avg_code_len] = myHuffmanDict(alphabet,cell2mat(probabilities),1);
%
% Encode source_b
%
enco = myHuffmanEnco(source_b,dict,1);
```

```
%
% Decode again and check if the result is the same
%
deco = myHuffmanDeco(enco, dict);
if isequal(deco,source_b)
  disp('Perfect!! Our Huffman works fine with kwords source.');
else
  error('Ooops maybe we missed something.');
end % if
fprintf('————————————————\n');
%
% Update the probabilities for each symbol
%
[updated_alphabet updated_probs] = myFreqCompute(source_b);
%
% Compute new Hufmman dictionary
%
[updated_dict, updated_avg_code_len] = ...
    myHuffmanDict(updated_alphabet,cell2mat(updated_probs), 1);
%
% Encode source_b
%
updated_enco = myHuffmanEnco(source_b,updated_dict,1);
%
% Decode again and check if the result is the same
%
updated_deco = myHuffmanDeco(updated_enco, updated_dict);
if isequal(updated_deco,source_b)
  disp('Perfect!! Our updated Huffman works fine kwords source.');
else
  error('Ooops maybe we missed something.');
end % if
```

- myFreqCompute.m

```
% Author: Christos Gkournelos
% Date: 02/01/2019
%
% An custom implimentation for computing the frequency of
% each symbol in a char array
%
function [alphabet freq] = myFreqCompute(input_stream)
  % initialize the alphabet
  cell_flag = 0;
  if iscell(input_stream)
    cell_flag = 1;
    util_struct.symbol{1} = input_stream{1};
  else
    util_struct.symbol{1} = input_stream(1);
  end % if
  util_struct.appearance{1} = 1;
  for i = 2 : length(input_stream)
    allready_exist = 0;
    for j = 1 : length(util_struct.symbol)
      if(cell_flag == 1)
        if (isequal(input_stream{i}, util_struct.symbol{j} ))
          allready_exist = 1;
          util_struct.appearance{j} = util_struct.appearance{j} + 1;
        end % if
      else
        if(strcmp(input_stream(i), util_struct.symbol{j}))
          allready_exist = 1;
          util_struct.appearance{j} = util_struct.appearance{j} + 1;
        end % if
      end % if
    end % for
    if(allready_exist == 0)
      if (cell_flag == 1)
```

```
34          util_struct.symbol{length(util_struct.symbol)+1} =  input_stream{i};
35        else
36          util_struct.symbol = [util_struct.symbol input_stream(i)];
37        end % if
38        util_struct.appearance = [util_struct.appearance 1];
39      end %if
40    end % for
41    for j = 1 : length(util_struct.symbol)
42      util_struct.freq{j} = util_struct.appearance{j} / length(input_stream);
43    end %for
44    freq = util_struct.freq;
45    alphabet = util_struct.symbol;
46  end % myFreqCompute
```

- myNOrderSourceGen.m

```
1   % Author: Christos Gkournelos
2   % Date: 03/01/2019
3   %
4   % An custom implimentation for creating
5   % an N-order symbol source
6   %
7   function [new_source, new_probs] = myNOrderSourceGen(init_source, init_probs, N)
8     new_source = {};
9     new_probs = {};
10    if (N == 2)
11      for i = 1:length(init_source)
12        for j = 1:length(init_source)
13          new_source{(length(new_source)) + 1} = [init_source{i} init_source{j}];
14          new_probs{(length(new_probs)) + 1} = init_probs{i}*init_probs{j};
15        end % for j
16      end % for i
17    elseif (N > 2)
18      [source_, probs_] = myNOrderSourceGen(init_source, init_probs, N-1);
19      for i = 1:length(init_source)
20        for j = 1:length(source_)
21          new_source{(length(new_source)) + 1} = [init_source{i} source_{j}];
22          new_probs{(length(new_probs)) + 1} = init_probs{i}*probs_{j};
23        end % for j
24      end % for i
25    elseif (N < 2)
26      error('N-order must be higher to 2');
27    end % if
28  end % myN_OrderSourceGen
```

Παράρτημα Β

Matlab scripts *DPCM*

- my_predictor_factors.m

```
% Author: Christos Gkournelos
% Date: 05/01/2019
%
% A function for computing the factors of
% the p-linear predictor
%
function [a_factors] = my_predictor_factors(p, x)
  %
  %  Input error checking
  %
  if (nargin ~= 2)
    error('Wrong input. \nThe function requires 2 inputs', -1)
  end % if
  N = length(x);
  %
  % Compute autocorrelation vector r_vec
  %
  for i = 1 : p
    sum = 0;
    for j = p+1 : N
      sum = sum + x(j)*x(j-i);
    end %for
    r_vec(i) = sum / (N - p);
  end % for
  %
  % Compute autocorrelation matrix R_mat
  %
  for i = 1:p
    for j = 1 : p
      sum = 0;
      for k = p+1 : N
        sum = sum + x(k-i)*x(k-j);
      end % for k
      R_mat(i,j) = sum / (N - p);
    end %for j
  end % for i
  %
  % Compute a factors
  %
  a_factors = R_mat \ r_vec';
end % my_predictor_factors
```

- my_quantizer.m

```
% Author: Christos Gkournelos
% Date: 05/01/2019
%
% An custom implimentation of a N-bits quantizer
%
function [y_quant] = my_quantizer (y , N, min_value, max_value)
  %
  %  Input error checking
  %
  if (nargin ~= 4)
    error('Wrong input. \nThe function requires 4 inputs', -1)
  end % if
  if y < min_value
    y = min_value;
  elseif y > max_value
   y = max_value;
```

```
  end % if
  %
  % Compute centers vector
  %
  Δ = 2*max_value/(2^(N));
  centers(1) = max_value - (Δ / 2);
  for i = 2:(2^N)
    centers(i) = centers(i-1) - Δ;
  end % for
  %
  % Find y's quantized value
  %
  for i = 1 : (2^N)
    if ((y ≥ (centers(i) - Δ/2)) && (y ≤ (centers(i) + Δ/2)))
      y_quant = centers(i);
      break;
    end % if
  end % for
end % my_quantizer
```

- my_dpcm_trans.m

```
% Author: Christos Gkournelos
% Date: 05/01/2019
%
% A function which emulates a DPCM trasmitter system
%
function [trans_vars] = my_dpcm_trans(x, p, N)
  %
  %  Input error checking
  %
  if (nargin ≠ 3)
    error('Wrong input. \nThe function requires 3 inputs', -1)
  end % if
  %
  % Initialization
  %
  max_quant_val =  3.5;
  min_quant_val = -3.5;
  for i = 1 : p
    y(i) = x(i);
    y_pred(i) = x(i);
    y_quant(i) = my_quantizer(x(i), N, min_quant_val, max_quant_val);
    y_mem(i) = y_pred(i) + y_quant(i);
  end % for
  %
  % Find prediction filter factors and quantize them
  %
  a_factors = my_predictor_factors(p, x);
  for i = 1 : length(a_factors)
    a_quant(i) = my_quantizer(a_factors(i), 8, -2, 2);
  end % for
  %
  % Main DPCM encoding loop
  %
  for i = (p+1) : length(x)
    y_pred(i) = sum(a_quant .* fliplr(y_mem((i-p):(i-1))));
    y(i) = x(i) - y_pred(i);
    y_quant(i) = my_quantizer(y(i), N, min_quant_val, max_quant_val);
    y_mem(i) = y_pred(i) + y_quant(i);
  end % i
  %
  % Fill the return structure
  %
  trans_vars.y = y;
  trans_vars.y_pred = y_pred;
  trans_vars.y_quant = y_quant;
  trans_vars.y_mem = y_mem;
```

```
47    trans_vars.a_factors = a_factors;
48    trans_vars.a_quant = a_quant;
49  end % my_dpcm_impl
```

- my_dpcm_rec.m

```
1   % Author: Christos Gkournelos
2   % Date: 05/01/2019
3   %
4   % A function which emulates a DPCM receiver system
5   %
6   function [rec_vars] = my_dpcm_rec(y_quant, p, a_factors, init_vec )
7     %
8     %  Input error checking
9     %
10    if (nargin ~= 4)
11      error('Wrong input. \nThe function requires 3 inputs', -1)
12    end % if
13    if (length(init_vec) ~= p)
14      error('Wrong input. \nInitialization vector must be same size with p ', -1)
15    end
16    if (length(a_factors) ~= p)
17      error('Wrong input. \nPrediction factors vector must be same size with p ', -1)
18    end
19    %
20    % Initialization
21    %
22    y_pred = init_vec;
23    y_mem = init_vec;
24    %
25    % Main DPCM encoding loop
26    %
27    for i = (p+1):length(y_quant)
28      y_pred(i) = sum(a_factors .* fliplr(y_mem((i-p):(i-1))));
29      y_mem(i) = y_quant(i) + y_pred(i);
30    end % for i
31    %
32    % Fill the return structure
33    %
34    rec_vars.y_pred = y_pred;
35    rec_vars.y_mem = y_mem;
36  end % my_dpcm_rec
```

Βοηθητικά Matlab scripts για την επαλήθευση των ερωτημάτων

- dpcm_predict_test.m

```
1   %
2   % This script is used for testing my DPCM implementation
3   % and plot the signal with prediction noise
4   %
5   clear; clc;
6   load('source.mat') % x: input signal
7   %
8   % First test p = 4
9   %
10  p = 4;
11  N = 3;
12  for i = 1:N
13    dpcm_trans_N{i} = my_dpcm_trans(x, p, i);
14    dpcm_rec_N{i} = my_dpcm_rec(dpcm_trans_N{i}.y_quant, p, ...
          dpcm_trans_N{i}.a_quant, x(1:p)');
15  end % for
16  %
```

```
% Plot
%
figure
hold on
plot((1:length(x)),x,'r-')
plot((1:length(x)),dpcm_trans_N{1}.y,'g-')
plot((1:length(x)),dpcm_trans_N{2}.y,'m-')
plot((1:length(x)),dpcm_trans_N{3}.y,'b-')
legend({'$x(n)$','$y(n) - N=1$','$y(n) - N=2$' ,'${y(n) - N=3}$'}, ...
    'Interpreter','latex');
xlabel('Discrete time $(n)$','Interpreter','latex');
ac = gca;
ac.FontSize = 18;
hold off
%
% Second test p = 24
%
clear;
load('source.mat') % x: input signal
p = 24;
N = 3;
for i = 1:N
  dpcm_trans_N{i} = my_dpcm_trans(x, p, i);
  dpcm_rec_N{i} = my_dpcm_rec(dpcm_trans_N{i}.y_quant, p, ...
      dpcm_trans_N{i}.a_quant, x(1:p)');
end % for
%
% Plot
%
figure
hold on
plot((1:length(x)),x,'r-')
plot((1:length(x)),dpcm_trans_N{1}.y,'g-')
plot((1:length(x)),dpcm_trans_N{2}.y,'m-')
plot((1:length(x)),dpcm_trans_N{3}.y,'b-')
legend({'$x(n)$','$y(n) - N=1$','$y(n) - N=2$' ,'${y(n) - N=3}$'}, ...
    'Interpreter','latex');
xlabel('Discrete time $(n)$','Interpreter','latex');
ac = gca;
ac.FontSize = 18;
hold off
```

- dpcm_mean_square_error.m

```
%
% This script is used for testing my DPCM implementation
% and plot the mean square error of  prediction
%
clear; clc;
load('source.mat') % x: input signal
N = 3;
a_factors_p = zeros(8,5);
a_quant_p = zeros(8,5);
for p =  4:8
  for i = 1:N
    dpcm_trans_N{i} = my_dpcm_trans(x, p, i);
    dpcm_rec_N{i} = my_dpcm_rec(dpcm_trans_N{i}.y_quant, p, ...
        dpcm_trans_N{i}.a_quant, x(1:p)');
    mean_sq_error(i,p-3) = immse(x',dpcm_trans_N{i}.y_pred);
  end % for
  a_factors_p(1:p,p-3) = dpcm_trans_N{1}.a_factors;
  a_quant_p(1:p,p-3) = dpcm_trans_N{1}.a_quant;
end %for p
%
% Plot
%
figure
hold on
```

```
plot(mean_sq_error(:,1),'b-s')
plot(mean_sq_error(:,2),'r-+')
plot(mean_sq_error(:,3),'g-x')
plot(mean_sq_error(:,4),'m-d')
plot(mean_sq_error(:,5),'c-*')
legend({'$p=4$','$p=5$','$p=6$' ,'$p=7$','$p=8$'}, 'Interpreter','latex');
xlabel('$N$ bits','Interpreter','latex');
ylabel('Mean square error','Interpreter','latex');
% title('$MSE = E(y^2(n)) =  E((x(n)-y'(n))^2)$','Interpreter','latex')
ac = gca;
ac.FontSize = 18;
hold off
```

- dpcm_reconstruct_signal.m

```
%
% This script is used for testing my DPCM implementation
% and plot the reconstructed signal over the initial signal
%
clear; clc;
load('source.mat') % x: input signal
%
% First test p = 4
%
p = 4;
N = 3;
for i = 1:N
  dpcm_trans_N{1,i} = my_dpcm_trans(x, p, i);
  dpcm_rec_N{1,i} = my_dpcm_rec(dpcm_trans_N{1,i}.y_quant, p, ...
      dpcm_trans_N{1,i}.a_quant, x(1:p)');
end % for
p = 8;
for i = 1:N
  dpcm_trans_N{2,i} = my_dpcm_trans(x, p, i);
  dpcm_rec_N{2,i} = my_dpcm_rec(dpcm_trans_N{2,i}.y_quant, p, ...
      dpcm_trans_N{2,i}.a_quant, x(1:p)');
end % for
%
% Plot N=1
%
figure
hold on
x_ax = 5000:5100;
plot(x_ax,x(5000:5100),'k-o')
plot(x_ax,dpcm_rec_N{1,1}.y_mem(5000:5100),'g-s')
plot(x_ax,dpcm_rec_N{2,1}.y_mem(5000:5100),'r-+')
legend({'$x(n)$','$\hat{y}(n) - p=4$','$\hat{y}(n) - p=8$'}, 'Interpreter','latex');
xlabel('Discrete time $(n)$','Interpreter','latex');
ac = gca;
ac.FontSize = 18;
hold off
%
% Plot N=2
%
figure
hold on
x_ax = 5000:5100;
plot(x_ax,x(5000:5100),'k-o')
plot(x_ax,dpcm_rec_N{1,2}.y_mem(5000:5100),'g-s')
plot(x_ax,dpcm_rec_N{2,2}.y_mem(5000:5100),'r-+')
legend({'$x(n)$','$\hat{y}(n) - p=4$','$\hat{y}(n) - p=8$'}, 'Interpreter','latex');
xlabel('Discrete time $(n)$','Interpreter','latex');
ac = gca;
ac.FontSize = 18;
hold off
%
% Plot N=3
%
```

```
figure
hold on
x_ax = 5000:5100;
plot(x_ax,x(5000:5100),'k-o')
plot(x_ax,dpcm_rec_N{1,3}.y_mem(5000:5100),'g-s')
plot(x_ax,dpcm_rec_N{2,3}.y_mem(5000:5100),'r-+')
legend({'$x(n)$','$\hat{y}(n) - p=4$','$\hat{y}(n) - p=8$'}, 'Interpreter','latex');
xlabel('Discrete time $(n)$','Interpreter','latex');
ac = gca;
ac.FontSize = 18;
hold off
```