

ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Ακαδημαϊκό Έτος 2018-2019

1^η Εργαστηριακή Άσκηση

Όνοματεπώνυμο: Πετράκης Κωσταντίνος

AM: 5878

Μέρος Α

Κωδικοποίηση Huffman

(1)

Οι πιθανότητες που δίνονται στο link της Wikipedia έχουν άθροισμα 99.9990. Έτσι έχω κανονικοποιήσει το διάνυσμα πιθανοτήτων $p=p./99.9990$ ώστε το άθροισμα των πιθανοτήτων του κάθε συμβόλου να είναι 1.

Για την κωδικοποίηση της πηγής A επειδή η συνάρτηση randsrc απαιτεί αριθμητικούς πίνακες για ορίσματα έχω δημιουργήσει έναν πίνακα ακεραίων numbers από 1 έως 26(όσα και τα γράμματα του αλφαριθμητικού της πηγής A) για να παράξω ένα σήμα 10000 χαρακτήρων και στην συνέχεια μεταφράζω τους ακεραίους στο αντίστοιχο γράμμα. Στον πίνακα symbols κρατάω το αλφάριθμητο της πηγής A (a-z) και με την εντολή sig_letters=symbols(sig); μεταφράζω τους ακεραίους στο αντίστοιχο γράμμα. Δηλαδή ο πίνακας sig_letters είναι οι 10000 χαρακτήρες που παράγει η πηγή A. Από τη συνάρτηση κωδικοποιήσης για την πηγή A προκύπτει:

The screenshot shows a terminal window titled "Φωτογραφίες - Στεγμένο οθόνης (10).png". The command entered is "Εργασία γίγαντας της φωτογραφίας". The terminal output shows the creation of a dictionary named "dict" from a list of binary strings. The list contains 26 entries, each consisting of a number from 1 to 26 followed by a binary string of length 5. The strings represent the binary form of the Greek alphabet (α, β, γ, δ, ε, Ζ, Ζ).

```
>> dict
dict =
[ 1]  '1110'
[ 2]  '110000'
[ 3]  '01001'
[ 4]  '11111'
[ 5]  '100'
[ 6]  '00101'
[ 7]  '110011'
[ 8]  '0110'
[ 9]  '1011'
[10]  '001001011'
[11]  '0010011'
[12]  '11110'
[13]  '00111'
[14]  '1010'
[15]  '1101'
[16]  '110001'
[17]  '001001001'
[18]  '0101'
[19]  '0111'
[20]  '000'
[21]  '01000'
[22]  '001000'
[23]  '00110'
[24]  '001001010'
[25]  '110010'
[26]  '001001000'
```

(Όπου ο κάθε αριθμός αντιστοιχεί στο αντίστοιχο γράμμα a-z)

Για να μετατρέψω την ακολουθία χαρακτήρων που παράγει η πηγή A στη δυαδική αναπαράσταση ASCII του κάθε συμβόλου χρησιμοποιώ την εντολή `sig_binary=reshape(dec2bin(sig_letters).'-0',1,[]);` Όπου `sig_letters` ο πίνακας που κρατάω την ακολουθία 10000 χαρακτήρων που παρήγαγε η πηγή A και στον `sig_binary` είναι η αντίστοιχη ακολουθία στο δυαδικό(σύμφωνα με την κωδικοποίηση ASCII).

Παρακάτω φαίνεται ένα στιγμιότυπο που επιβεβαιώνει την ορθή κωδικοποίηση και αποκωδικοποίηση της πηγής A.

The screenshot shows a MATLAB command window titled "Φωτογραφίες - Στηγνότυπο οθόνης (13).png". The workspace pane on the right lists variables: `dsig_letters`, `numbers`, `p`, `sig`, `sig_binary`, `sig_letters`, and `symbols`. The command history pane contains the following code:

```

>> p=mp./99.9990;
>> numbers=l:26;
>> symbols='a':'z';
>> dict=my_huffmandict(numbers,p);
>> sig=randsrc(10000,1,[numbers; p]);
>> comp=my_huffmanenco(sig,dict);
>> sig_letters=symbols(sig);
>> sig_binary=reshape(dec2bin(sig_letters).'-0',1,[]);
>> whos sig_binary
  Name      Size            Bytes  Class       Attributes
  sig_binary    1x70000        560000  double
  ans =
  1
  Did you mean:
  >> isequal(sig_letters,dsig_letters)
  ans =
  1
  >> whos comp
  Name      Size            Bytes  Class       Attributes
  comp      1x41936         83872  char

```

Βλέπω ότι το μήκος της ακολουθίας 10000 χαρακτήρων σε bit είναι 70000 ενώ με την κωδικοποίηση Huffman παίρνω μήκος κώδικα 41936. Πετυχαίνω δηλαδή 40% οικονομικότερη αναπαράσταση σε σχέση με τη μη κωδικοποιημένη ακολουθία.

Η εντροπία της πηγής A σύμφωνα με τον τύπο $H(A) = -\sum_{i=1}^{26} p_i * \log p_i$ είναι 4,1758 bit/symbol.

Βλέπω ότι ικανοποιείται η σχέση $0 < H(A) < \log(26) = 4,7004$.

Για να υπολογίσω το μέσο μήκος κώδικα χρησιμοποίησα έναν προσωρινό πίνακα `leng` που κρατά το μήκος κώδικα για κάθε σύμβολο και στην συνέχεια εκτέλεσα `sum(p.*leng)`. Για τον υπολογισμό του `leng` έτρεξα

`for i=1:26`

`leng(i)=length(cell2mat(dict(i,2)));` % επειδή το πεδίο με την αναπαράσταση Huffman του συμβόλου είναι τύπου cell%

`end`

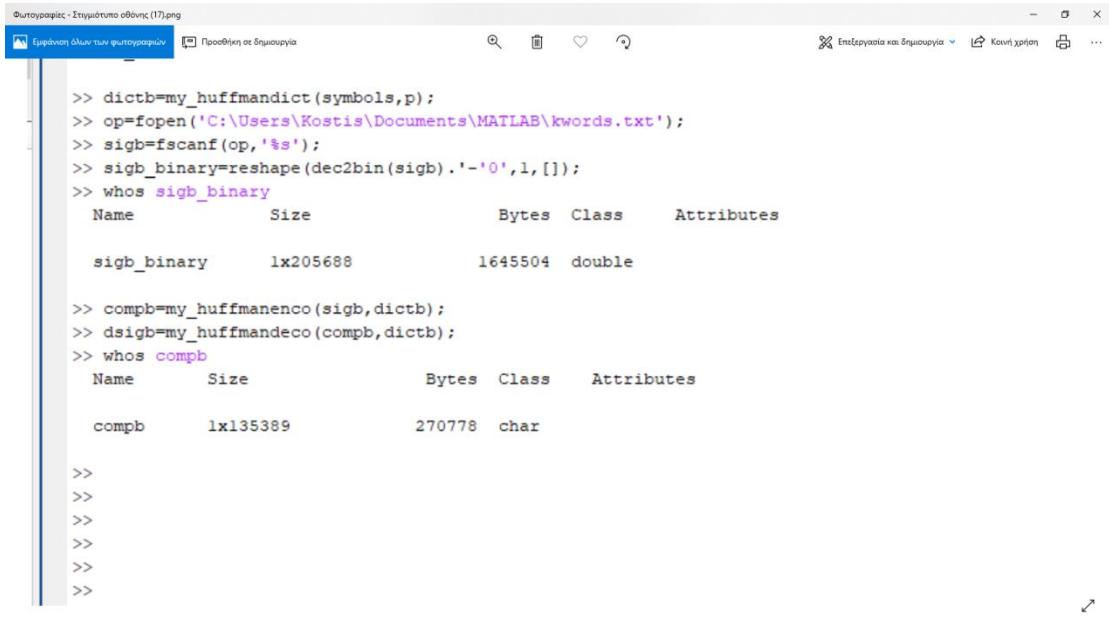
Το μέσο μήκος κώδικα που παράγει η συνάρτηση κωδικοποίησης my_huffmandict είναι $\bar{L} = \sum_{i=1}^{2^6} p(s_i) * l(s_i) = 4,2051 \text{ bit/symbol}$ και ικανοποιείται η σχέση $H(A) < \bar{L} < H(A) + 1$. Δηλαδή η αποδοτικότητα του κώδικα μου είναι $\eta = \frac{H(A)}{\bar{L}} = \frac{4.1758}{4.2051} = 0,993$. Παρατηρώ δηλαδή ότι είμαι πολύ κοντά στην εντροπία της πηγής A.

Για την πηγή B από την συνάρτηση κωδικοποίησης προκύπτει

αίστερ =	
'a'	'1110'
'b'	'110000'
'c'	'01001'
'd'	'11111'
'e'	'100'
'f'	'00101'
'g'	'110011'
'h'	'0110'
'i'	'1011'
'j'	'001001011'
'k'	'0010011'
'l'	'11110'
'm'	'00111'
'n'	'1010'
'o'	'1101'
'p'	'110001'
'q'	'001001001'
'r'	'0101'
's'	'0111'
't'	'000'
'u'	'01000'
'v'	'001000'
'w'	'00110'
'x'	'001001010'
'y'	'110010'
'z'	'001001000'

(Επισημαίνεται πως είναι ακριβώς η ίδια κωδικοποίηση με την A με την διαφορά ότι στην A χρησιμοποιήσα σαν ενδιάμεσο έναν πίνακα ακεραίων για την χρήση της συνάρτησης randsrc.)

Παρακάτω φαίνεται ένα στιγμιότυπο για την κωδικοποίηση της πηγής Β



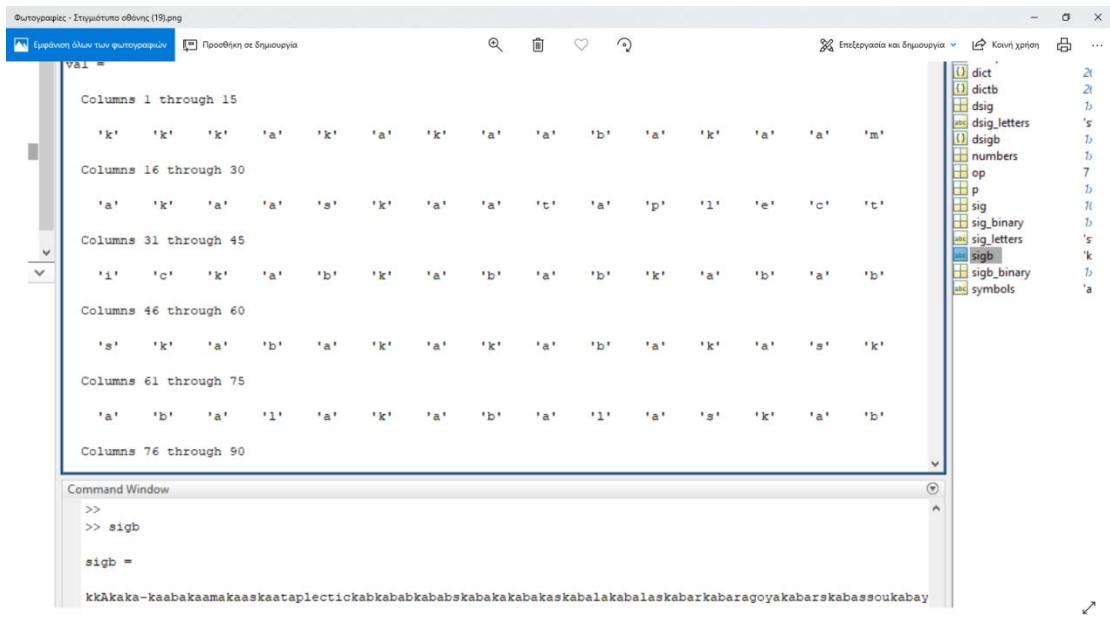
```
>> dictb=my_huffmandict(symbols,p);
>> op=fopen('C:\Users\Kostis\Documents\MATLAB\kwords.txt');
>> sigb=fscanf(op,'%s');
>> sigb_binary=reshape(dec2bin(sigb).'-0',1,[]);
>> whos sigb_binary
  Name      Size            Bytes  Class    Attributes
  sigb_binary    1x205688        1645504  double

>> compb=my_huffmanenco(sigb,dictb);
>> dsigb=my_huffmandeco(compb,dictb);
>> whos compb
  Name      Size            Bytes  Class    Attributes
  compb     1x135389        270778  char

>>
>>
>>
>>
>>
```

Για να μετατρέψω την ακολουθία χαρακτήρων της πηγής Β(δηλ. του αρχείου kwords.txt) στη δυαδική αναπαράσταση ASCII του κάθε συμβόλου χρησιμοποιώ την εντολή `sigb_binary=reshape(dec2bin(sigb).'-0',1,[]);` Όπου `sigb` είναι το αρχείο `kwords.txt` σαν πίνακας `string` και στον `sigb_binary` κρατάω την πηγή Β σαν δυαδική ακολουθία. Βλέπω λοιπόν ότι η πηγή Β απαιτεί 205.688 bit για την αναπραστασή της ενώ με την κωδικοποίηση huffman παίρνω μήκος κώδικα 135.389 bit. Πετυχαίνω δηλαδή 34% οικονομικότερη αναπαράσταση σε σχέση με την μη κωδικοποιημένη ακολουθία. Βλέπω ότι πετυχαίνω μικρότερη συμπίεση από ότι στην πηγή Α και αυτό γιατί δεν εκμεταλεύομαι την μεγάλη συχνότητα εμφάνισης του συμβόλου `k` (καθώς όλες οι λέξεις ξεκινούν με αυτό) στην πηγή Β.

Όσον αφορά την ορθή αποκωδικοποίηση της πηγής Β επειδή το αλφάριθμο που έχω κωδικοποίησει περιέχει μόνο τα πεζά γράμματα του αγγλικού αλφαριθμού ενώ το αρχείο `kwords.txt` εμπεριέχει και κάποιους επιπλέον χαρακτήρες (όπως το κεφαλαίο γράμμα `A` και την παύλα `-`) ηέξοδος την συνάρτησης αποκωδικοποίησης δεν είναι ακριβώς ίδια με τα περιεχόμενα του αρχείου `kwords.txt`. Όπως μπορούμε να δούμε από το παρακάτω στιγμιότυπο όμως η αποκωδικοποιημένη ακολουθία μου είναι ίδια με την έξοδο της πηγής Β για όλα τα σύμβολα που εμπεριέχονται στο αλφάριθμο.



(επάνω είναι η έξοδος της συνάρτησης αποκωδικοποίησης μου σε μορφή *cell array* και κάτω φαίνεται η πηγή *B* σε μορφή *string array*.)

Δηλαδή η κωδικοποίηση/αποκωδικοποίηση για την πηγή *B* έγιναν σωστά αλλά μόνο για τα σύμβολα που εμπερέχονται στο αλφάβητο(πεζοί χαρακτήρες). Τα υπόλοιπα άγνωστα στον κωδικοποιητή σύμβολα απλά αγνοήθηκαν.

Η εντροπία της πηγής *B* σύμφωνα με τις πιθανότητες που χρησιμοποιήσα και για την *A* είναι $H(B) = - \sum_{i=1}^{2^6} p_i * \log p_i = 4,1758 \text{ bit/symbol}$, ίση με την εντροπία της *A*

Το μέσο μήκος κώδικα που παράγει η συνάρτηση κωδικοποίησης *my_huffmandict* για την πηγή *B* είναι και πάλι $\bar{L} = \sum_{i=1}^{2^6} p(s_i) * l(s_i) = 4,2051 \text{ bits/symbol}$ και ικανοποιείται η σχέση $H(B) < \bar{L} < H(B) + 1$. Δηλαδή η αποδοτικότητα του κώδικα μου είναι $\eta = \frac{H(B)}{\bar{L}} = \frac{4,1758}{4,2051} = 0,993$.

(2)

Για να εκτιμήσω τις πιθανότητες των συμβόλων από το αρχείο *kwrods.txt* αρχικά προσθέτω στον πίνακα *symbols* τα σύμβολα 'Α' και '-' δηλαδή ο πίνακας *symbols* είναι πλέον *symbols=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','A','']*; Και στην συνέχεια υπολογίζω τις πιθανότητες κάθε συμβόλου όπως φαίνεται παρακάτω

```

Φωτογραφίες - Στιγμιότυπο οθόνης (21).png
Εμφάνιση όλων των φωτογραφιών | Προσθήτη σε δημιουργία | Επεξεργασία και δημιουργία | Κανβάρι χρήστη | ...
>> symbols=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u',
>> for i=1:28
freqb(i)=length(strfind(sigb,symbols(i)));
end
>> for i=1:28
p_b(i)=freqb(i)/length(sigb);
end
>> sum(p_b)

ans =
0.9992

>> p_b=p_b./0.9992;
>> sum(p_b)

ans =
1.0000

f1 >>

```

Επειδή όπως φαίνεται το άθροισμα των πιθανοτήτων είναι 0,9992 κανονικοποιώ το διάνυσμα ώστε να έχει άθροισμα 1. Και το διάνυσμα p_b περιέχει τώρα τις πιθανότητες των συμβόλων της πηγής B.

Και το νέο αλφάριθμο για την πηγή B τώρα είναι:

```

Φωτογραφίες - Στιγμιότυπο οθόνης (23).png
>> dictb2=my_huffmandict(symbols,p_b);
>> dictb2

dictb2 =
{'a' '1110'
'b' '011110'
'c' '111111'
'd' '111110'
'e' '010'
'f' '10111111'
'g' '101110'
'h' '01110'
'i' '000'
'j' '011111001'
'k' '110'
'l' '0010'
'm' '111100'
'n' '1010'
'o' '1001'
'p' '101100'
'q' '01111100000'
'r' '0011'
's' '1000'
't' '0110'
'u' '111101'
'v' '10111110'
'w' '011111'
'x' '0111110001'
'y' '101101'
'z' '01111101'
'A' '0111110001'
'-' '1011110'

available
ans
com
com
dict
dict
dict
dsig
dsig
dsig
freq
i
num
op
p
p_b
sig
sig_l
sig_l
sigb
sigb
sym

```

Ακολουθούν στιγμιότυπα για την κωδικοποίηση/αποκωδικοποίηση της πηγής B:

Όπου `compb2` το αρχείο `kwords.txt` κωδικοποιημένο με το νέο αλφάριθμο `dictb2`.

```

Φυτογραφίες - Στημμένο στο οθόνη (25).png
Εμφάνιση όλων των φωτογραφιών | Προσθήτη σε δημιουργία | Αναζήτηση | Καρδιά | Επεξεργασία και δημιουργία | Κανάλι χρήση | Εγκατάσταση | ...
'q'    '01111100000'
'r'    '0011'
's'    '1000'
't'    '0110'
'u'    '111101'
'v'    '10111110'
'w'    '0111111'
'x'    '0111110001'
'y'    '101101'
'z'    '01111101'
'A'    '01111100001'
'-'   '1011110'

>> compb2=my_huffmanenco(sigb,dictb2);
>> dsigb2=my_huffmandeco(compb2,dictb2);
>> whos compb2
  Name      Size            Bytes  Class       Attributes
  compb2    1x121090        242180  char

fx >>
<

```

Επάνω είναι η έξοδος της συνάρτησης αποκωδικοποίησης μου με το νέο αλφάβητο dictb2 σε μορφή cell array και κάτω φαίνεται η πηγή Β σε μορφή string array.

```

Φυτογραφίες - Στημμένο στο οθόνη (27).png
'k'    'k'    'A'    'k'    'a'    'k'    'a'    '-'   'k'    'a'    'a'    'b'    'a'    'k'    'a'
Columns 16 through 30
'a'    'm'    'a'    'k'    'a'    'a'    's'    'k'    'a'    'a'    't'    'a'    'p'    'l'    'e'
Columns 31 through 45
'c'    't'    'i'    'c'    'k'    'a'    'b'    'k'    'a'    'b'    'a'    'b'    'k'    'a'    'b'
Columns 46 through 60
'a'    'b'    's'    'k'    'a'    'b'    'a'    'k'    'a'    'k'    'a'    'b'    'a'    'k'    'a'
Columns 61 through 75
's'    'k'    'a'    'b'    'a'    'l'    'a'    'k'    'a'    'b'    'a'    'l'    'a'    's'    'k'
Command Window
compb2    1x121090        242180  char
>> sigb
sigb =
kkAkaka-kaabakaamakaaskaataaplectickbabkababskabakakabaskabalakabaskabarkabaragoyakabarskabassoukaba

```

Βλέπω λοιπόν ότι πλέον έχοντας συμπεριλάβει στο αλφάβητο εισόδου τα σύμβολα 'Α' και '-' η αποκωδικοποίηση είναι απόλυτα σωστή.

Επίσης το κωδικοποιημένο κατά Huffman διάνυσμα compb2 έχει τώρα μήκος 121.090 (Θυμίζω ότι πριν το αντίστοιχο διάνυσμα compb ήταν μήκους 135.389). Υπολογίζοντας τις πιθανότητες των συμβόλων από το αρχείο keywords.txt πετυχαίνω τώρα 41% οικονομικότερη αναπαράσταση με την κωδικοποίηση Huffman. Το αποτέλεσμα είναι αναμενόμενο καθώς η

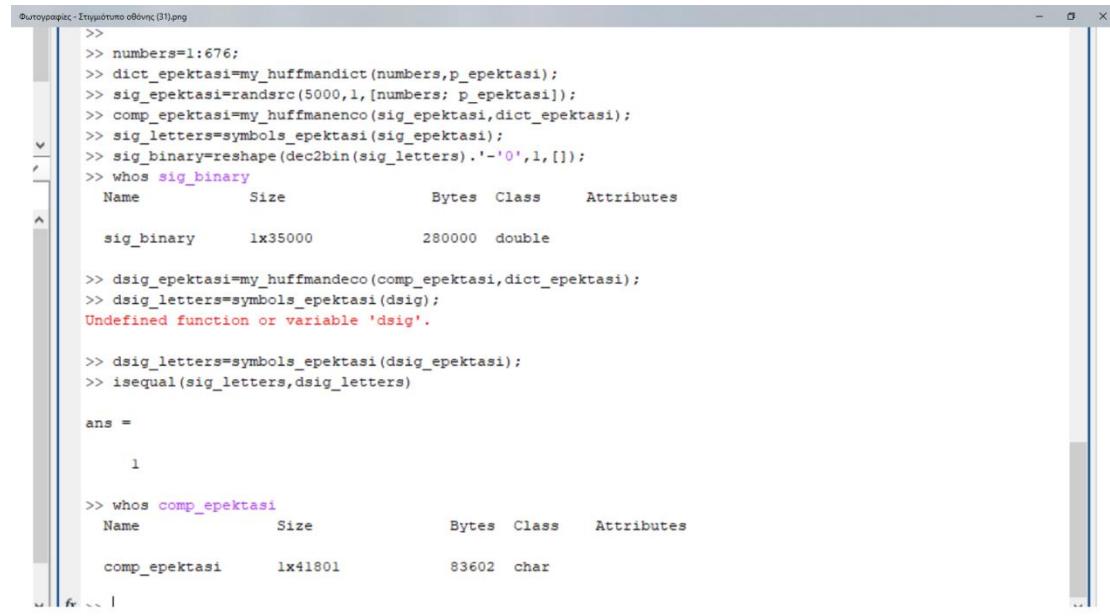
κωδικοποίηση μου τώρα συμπιέζει αποδοτικότερα την πηγή Β καθώς έχω λάβει υπ' όψιν μου ποια είναι τα πιο συχνά εμφανιζόμενα σύμβολα στην Β (όπως μπορούμε να δούμε και από το νέο αλφάριθμο πχ για το k που είναι το πιο συχνά εμφανιζόμενο γράμμα θέλω τώρα μόνο 3 bit). Επομένως είναι λογικό τώρα να πετυχαίνω καλύτερη συμπίεση για την πηγή Β.

(3)

Η δεύτερης τάξης επέκταση της πηγής Α θα αποτελείται από $26^2 = 676$ σύμβολα και πιο συγκεκριμένα το νέο αλφάριθμο θα είναι: {aa,ab,ac,...az,ba,bb,bc,...bz,....za,zb,zc,...zz}.

Αφού η αρχική πηγή πληροφορίας Α δεν έχει μνήμη (δηλαδή διαδοχικά σύμβολα που εκπέμπονται από την πηγή είναι στατιστικά ανεξάρτητα) οι πιθανότητες των ομαδοποιημένων συμβόλων θα δίνονται από το γινόμενο των πιθανοτήτων εμφάνισης των επιμέρους συμβόλων. Όπως και στο πρώτο ερώτημα για την κωδικοποίηση της πηγής Α επειδή η συνάρτηση randsrc απαιτεί αριθμητικούς πίνακες για ορίσματα έχω δημιουργήσει έναν πίνακα ακεραίων numbers από 1 έως 676 (όσα και τα σύμβολα του αλφαριθμού της δεύτερης τάξης επέκτασης της πηγής Α) για να παράξω ένα σήμα 5000 χαρακτήρων και στην συνέχεια μεταφράζω τους ακεραίους στο αντίστοιχο σύμβολο.

Ακολουθεί στιγμή του που επιβεβαιώνει την ορθή κωδικοποίηση και αποκωδικοποίηση της δεύτερης επέκτασης της πηγής Α



```

Φωτογραφίες - Σπηλιόπουλο Ιωάννης (31).png
>>
>> numbers=1:676;
>> dict_epektasi=my_huffmandict(numbers,p_epektasi);
>> sig_epektasi=randsrc(5000,1,[numbers; p_epektasi]);
>> comp_epektasi=my_huffmanenco(sig_epektasi,dict_epektasi);
>> sig_letters=symbols_epektasi(sig_epektasi);
>> sig_binary=reshape(dec2bin(sig_letters).-'0',1,[]);
>> whos sig_binary
  Name      Size            Bytes  Class       Attributes
  sig_binary    1x35000        280000  double
                                         ...
>> dsig_epektasi=my_huffmandeco(comp_epektasi,dict_epektasi);
>> dsig_letters=symbols_epektasi(dsig_epektasi);
Undefined function or variable 'dsig'.
>> dsig_letters=symbols_epektasi(dsig_epektasi);
>> isequal(sig_letters,dsig_letters)
ans =
  1
>> whos comp_epektasi
  Name      Size            Bytes  Class       Attributes
  comp_epektasi    1x41801        83602  char
                                         ...

```

(όπου dict_epektasi το αλφάριθμο της επεκταμένης πηγής , p_epektasi πίνακα 1x676 με τις πιθανότητες των συμβόλων που προκύπτουν)

Έχω μετατρέψει τους αριθμούς σε σύμβολα όπως και στα προηγούμενα ερωτήματα.

Εδώ παρατηρώ το εξής παράδοξο, ότι ενώ όπως θα δείξω παρακάτω η αποδοτικότητα του κώδικα μου είναι καλύτερη από πριν, με την κωδικοποίηση παίρνω ακολουθία μήκους 41801 σε σχέση με την ακολουθία 35000 χαρακτήρων(σε κωδικοποίηση ASCII) που παράγει η επεκταμένη πηγή Α. Η μόνη εξήγηση που μπορώ να δώσω είναι ότι έχει αυξηθεί το μέσο

μήκος κώδικα σε τιμή μεγαλύτερη του 7 (που είναι ο αριθμός των bit για την αναπαράσταση κάθε συμβόλου σε ASCII) και γι αυτό παίρνω συμπιεσμένη ακολουθία μεγαλύτερου μήκους (θα αποδειχθεί παρακάτω).

Ακολουθεί στιγμιότυπο από τον υπολογισμό της εντροπίας και του μέσου μήκους κώδικα:

```

Φωτογραφίες - Στηγμένο οθόνης (33).png
Clear Workspace Clear Commands Layout Add-Ins Help Request Support
LE CODE SIMULINK Parallel ENVIRONMENT RESOURCES
>> H_eppektasi=-sum(p_eppektasi.*log2(p_eppektasi))
H_eppektasi =
8.3516
>> for i=1:676
L(i)=p_eppektasi(i)*length(cell2mat(dict_eppektasi(i,2)));
end
>> sum(L)
ans =
8.3819
fxt >> |

```

Η εντροπία της επεκταμένης πηγής A σύμφωνα με τον τύπο $H(A^2) = -\sum_{i=1}^{676} p_i * \log p_i$ είναι 8,3516 bit/symbol. Βλέπω ότι ικανοποιείται η σχέση που συνδέει την εντροπία της πηγής H(A) με την εντροπία της 2ης τάξης επέκτασής της δηλαδή $H(A^2) = 2 * H(A) = 2 * 4,1758 = 8,3516$.

Το μέσο μήκος κώδικα που παράγει η συνάρτηση κωδικοποίησης my_huffmandict με το επεκταμένο αλφάβητο της πηγής A είναι $\bar{L}_2 = \sum_{i=1}^{676} p(s_i) * l(s_i) = 8,3819$ bit/symbol.

Από εδώ συμπεραίνω ότι είναι λογικό το κωδικοποιημένο διάνυσμα να έχει μεγαλύτερο μήκος από την ακολουθία που παράγει η πηγή καθώς όπως βλέπουμε το μέσο μήκος κώδικα είναι $8,3819 > 7$ bit/symbol που απαιτεί η κωδικοποίηση ASCII.

$$\text{Τελικά η απόδοση της κωδικοποίησης τώρα είναι } \eta = \frac{H(A^2)}{\bar{L}_2} = \frac{8,3516}{8,3819} = 0,9964.$$

Παρατηρούμε εδώ πως με τη δεύτερης τάξης επέκταση της πηγής πετυχαίνουμε καλύτερη απόδοση κωδικοποίησης από το ερώτημα (1).

Αυτό μπορούμε να το δείξουμε γενικά ξεκινώντας από το γνωστό τύπο:

$H(A) \leq \bar{L} < H(A) + 1$. Η εφαρμογή του τύπου αυτού στην 2ης τάξης επέκταση της πηγής δίνει:

$$H(A^2) \leq \bar{L}_2 < H(A^2) + 1 \Leftrightarrow 2 * H(A) \leq \bar{L}_2 < 2 * H(A) + 1$$

$$\Leftrightarrow H(A) \leq \frac{\bar{L}_2}{2} < H(A) + \frac{1}{2} \Leftrightarrow 4.1758 \leq \frac{8,3819}{2} < 4.1758 + \frac{1}{2} \text{ που προφανώς ισχύει.}$$

Γενικά αν είχαμε την η τάξης επέκταση της πηγής A, καθώς το η αυξάνεται τόσο περισσότερο το μήκος του προθεματικού κώδικα πλησιάζει την εντροπία της πηγής (=βέλτιστο μήκος κώδικα) και η απόδοση της κωδικοποίησης τείνει στο 1 καθώς το η τείνει στο άπειρο.

Σημείωση: Ο κώδικας για τις συναρτήσεις που ζητούνται παρατίθεται στο τέλος της αναφοράς.

Μέρος Β

Κωδικοποίηση

Διακριτής Πηγής με τη μέθοδο DPCM

(1)

Για την υλοποίηση του ζητούμενου συστήματος κωδικοποίησης/αποκωδικοποίησης DPCM αρχικά υπολογίζω το διάνυσμα συντελεστών του φίλτρου πρόβλεψης με βάση τις σχέσεις που δίνονται και στην συνέχεια χρησιμοποιώ την συνάρτηση μου `my_quantizer`, που προσομοιώνει την λειτουργία του κβαντιστή, για να πάρω τις κβαντισμένες τιμές(με N=8bits και δυναμική περιοχή [-2,2]) αυτού του διανύσματος. Και ο πομπός και ο δέκτης χρησιμοποιούν τις ίδιες κβαντισμένες τιμές του φίλτρου πρόβλεψης ώστε να λειτουργούν σε συμφωνία.

Έχω υλοποιήσει τη συνάρτηση `my_qyantizer`, που προσομοιώνει την λειτουργία του κβαντιστή, ώστε εκτός από την θέση στο διάνυσμα `centers` να επιστρέφει και το ίδιο το διάνυσμα `centers`(με την κατάλληλη κλήση) . Ο λόγος για αυτό είναι ότι χρειάζομαι το διάνυσμα `centers`, που περιέχει τα κέντρα των περιοχών κβάντισης, σαν είσοδο στις συναρτήσεις `my_dpcmenco()` και `my_dpcmdesco()` που προσομοιώνουν την λειτουργία του πομπού και του δέκτη αντίστοιχα.

Οι συντελεστές του φίλτρου πρόβλεψης αλλάζουν μόνο όταν αλλάζω την τάξη του φίλτρου πρόβλεψης `r`. Και κάθε φορά που αλλάζουν δημιουργώ το διάνυσμα α όπως εξηγείται παρακάτω και το κβαντίζω.

Στο παρακάτω στιγμιότυπο φαίνεται πως δημιουργώ τον πίνακα αυτοσυσχέτισης `R(Rxx` στον κώδικα) και το διάνυσμα αυτοσυσχέτισης `r(Rx` στον κώδικα) με βάση τις σχέσεις που δίνονται. (εδώ για `r=4` αλλά για όλες τις τιμές του `r` παρακάτω χρησιμοποιώ τον ίδιο τρόπο)

```

Фотографии - Στηματόποιο οθόνης (39).png
Εμφάνιση όλων των φωτογραφιών Προσθήκη σε δημοσιότητα
ents > MATLAB
Command Window
Undefined function or variable 'x'.
>> load('C:\Users\Kostis\AppData\Local\Temp\Rar$DRa0.966\Askhsh_1_18_19\Source_meros_b\source.mat')
>> for i=1:p
for n=p+1:length(x)
hold(n)=x(n)*x(n-i);
end
temp(i)=sum(hold);
end
>> for i=1:p
Rx(i)=(1/(length(x)-p))*temp(i);
end
>>
>> for i=1:p
for j=1:p
for n=p+1:length(x)
hold(n)=x(n-j)*x(n-i);
end
temp(i,j)=sum(hold);
end
end
>> for i=1:p
for j=1:p
Rxx(i,j)=(1/(length(x)-p))*temp(i,j);
end
end

```

Στη συνέχεια λύνω το γραμμικό σύστημα με την εντολή $a=Rxx \setminus Rx'$; για να προκύψει το διάνυσμα των συντελεστών του φίλτρου πρόβλεψης a .

Προσθέτω ένα μηδενικό σαν πρώτο στοιχείο του διανύσματος συντελεστών a έτσι ώστε αν η γραμμική συνάρτηση πρόβλεψης είναι

$y(k) = p(1)x(k-1) + p(2)x(k-2) + \dots + p(m-1)x(k-m+1) + p(m)x(k-m)$, τότε το αντίστοιχο διάνυσμα συντελεστών του φίλτρου πρόβλεψης να είναι

$a = [0 \ p(1) \ p(2) \ \dots \ p(m-1) \ p(m)]$. Έτσι παρακάτω χρησιμοποιώ την εντολή $a = [0; a]$; για να πετύχω αυτήν την συμπεριφορά και με τις εντολές

```

for i=1:length(a)

[y_out,centers]=my_quantizer(a(i),8,-2,2);

predictor(i)=centers(y_out);

end

```

παίρνω το κβαντισμένο διάνυσμα συντελεστών predictor , το οποίο και χρησιμοποιούν πομπός και δέκτης.

(2)

Στην συνέχεια δίνω ένα παράδειγμα του πως υλοποιώ το σύστημα κωδικοποίησης/αποκωδικοποίησης DPCM για $p=4$ και $N=3\text{bits}$ και $N=4\text{bits}$ και πως φτιάχνω τα γραφήματα που ζητούνται.

Φωτογραφίες - Στιγμιότυπο οθόνης (73).png

Εμφάνιση όλων των φωτογραφιών Προσθήτη σε δημοσιότητα

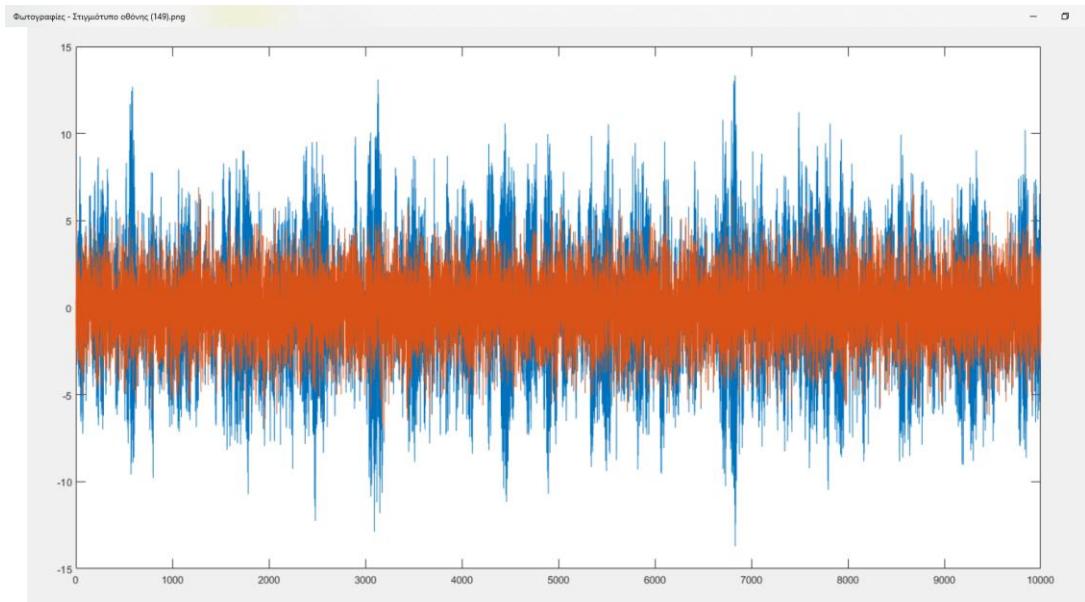
```

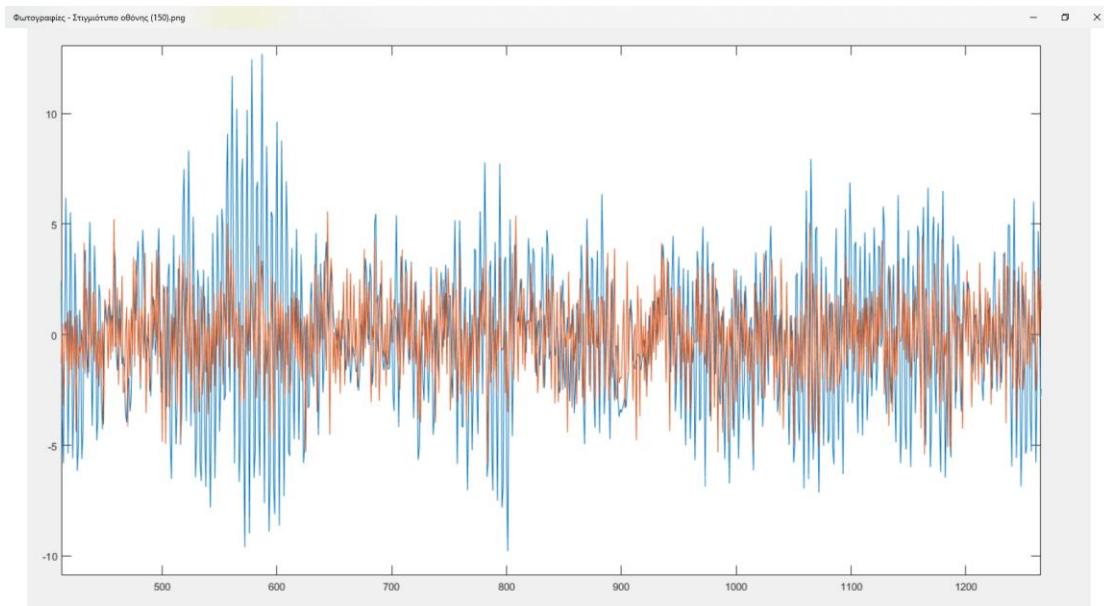
for j=1:p
Rxx(i,j)=(1/(length(x)-p))*temp(i,j);
end
end
>> a=Rxx\Rxx';
>> a=[0; a];
>> for i=1:length(a)
[y_out,centers]=my_quantizer(a(i),8,-2,2);
my_predictor(i)=centers(y_out);
end
>> predictor=my_predictor;
>> [y_out,centers]=my_quantizer(x(1),1,-3.5,3.5);
>> partition=[-3.5:3.5];
>> [indx,quanter]=my_dpcmenco(1,x,centers,partition,predictor);
[decodedx,dec_quanter]=my_dpcmdeco(indx,centers,predictor);
>> plot(t,x,t,quanter)
>> [y_out,centers]=my_quantizer(x(1),2,-3.5,3.5);
>> partition=[-3.5:1.75:3.5];
>> [indx,quanter]=my_dpcmenco(2,x,centers,partition,predictor);
[decodedx,dec_quanter]=my_dpcmdeco(indx,centers,predictor);
>> plot(t,x,t,quanter)
>> [y_out,centers]=my_quantizer(x(1),3,-3.5,3.5);
>> partition=[-3.5:0.875:3.5];
>> [indx,quanter]=my_dpcmenco(3,x,centers,partition,predictor);
[decodedx,dec_quanter]=my_dpcmdeco(indx,centers,predictor);
>> plot(t,x,t,quanter)
fk >> |
```

Για λόγους πληρότητας της αναφοράς παραθέτω γραφήματα του αρχικού σήματος με το σφάλμα πρόβλεψης πριν και μετά την κβάντιση του. Και στις 2 περιπτώσεις δίνονται 2 στιγμιότυπα του ίδιου γραφήματος, η κανονική μορφή και μια μεγενθυμένη.

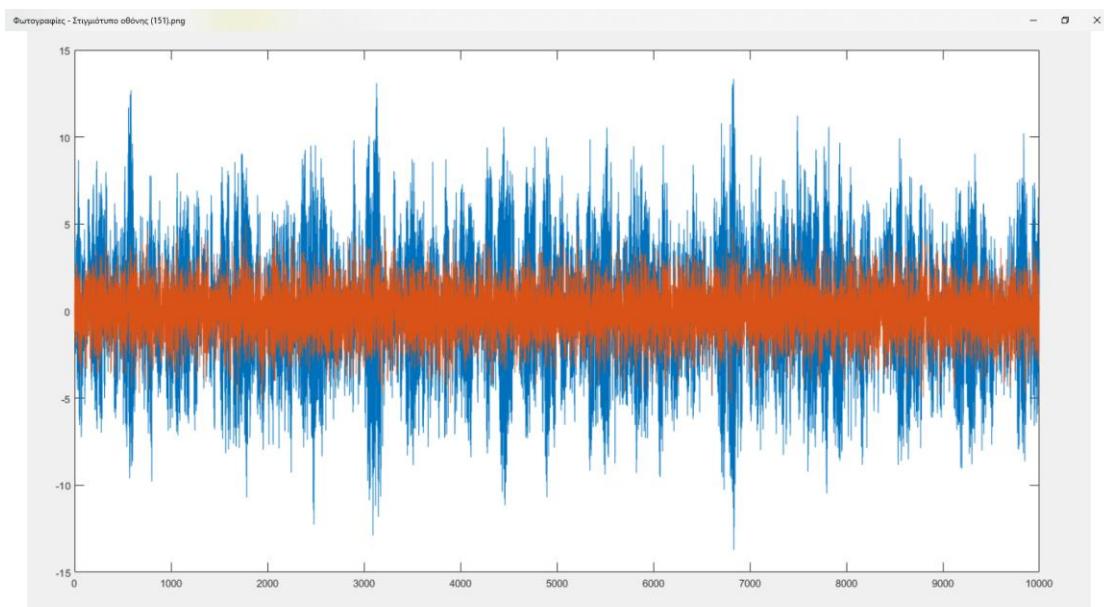
Στα παρακάτω στιγμιότυπα φαίνεται το γράφημα του αρχικού σήματος και το σφάλμα πρόβλεψης για τις αντίστοιχες παραμέτρους:

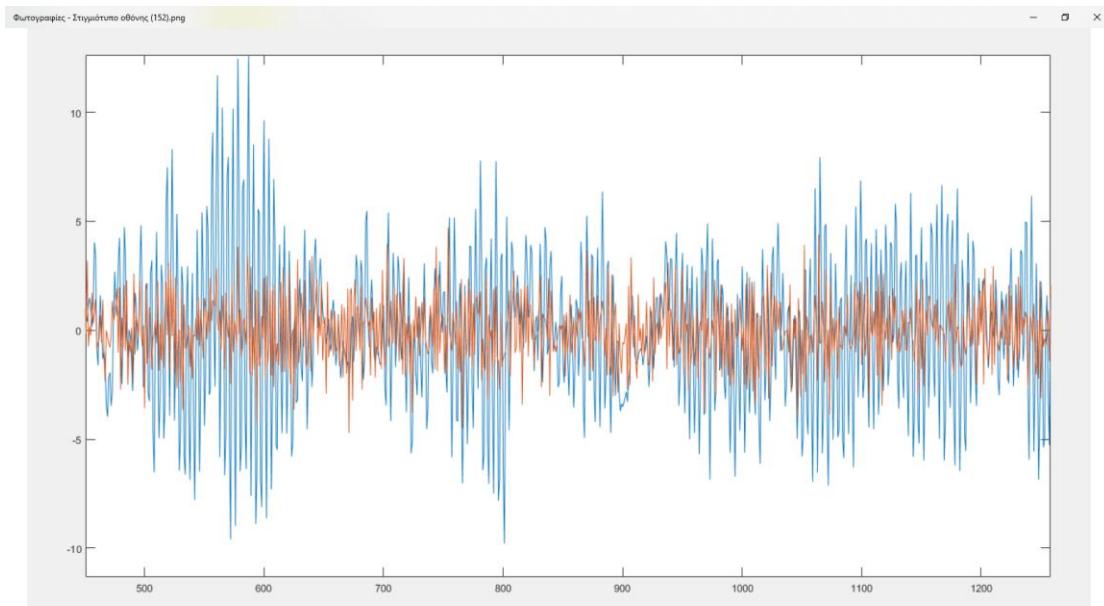
N=1bit και τάξη προβλέπτη p=4



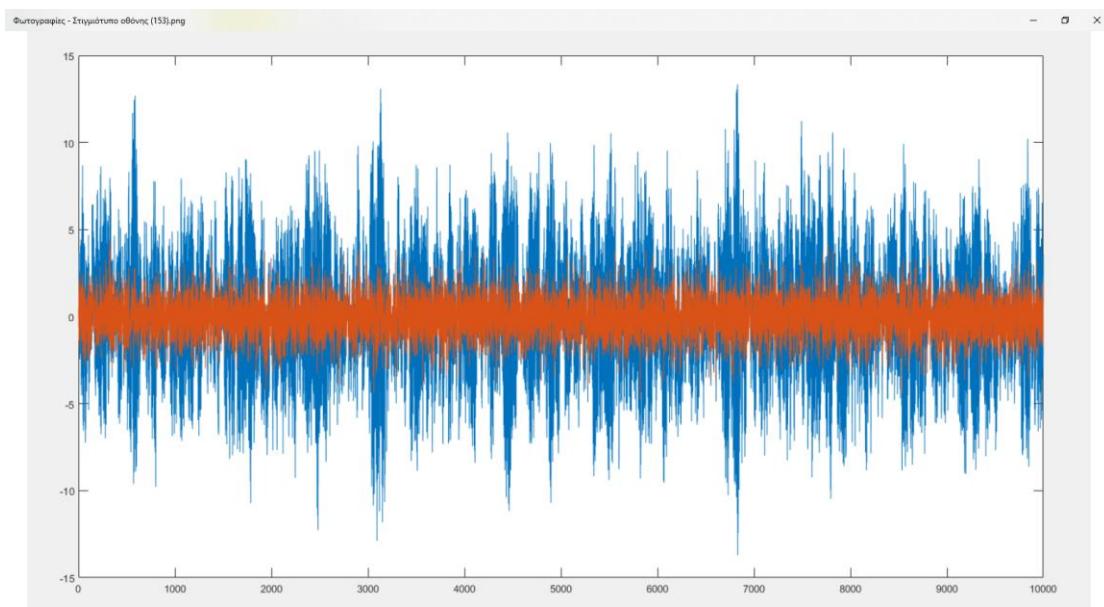


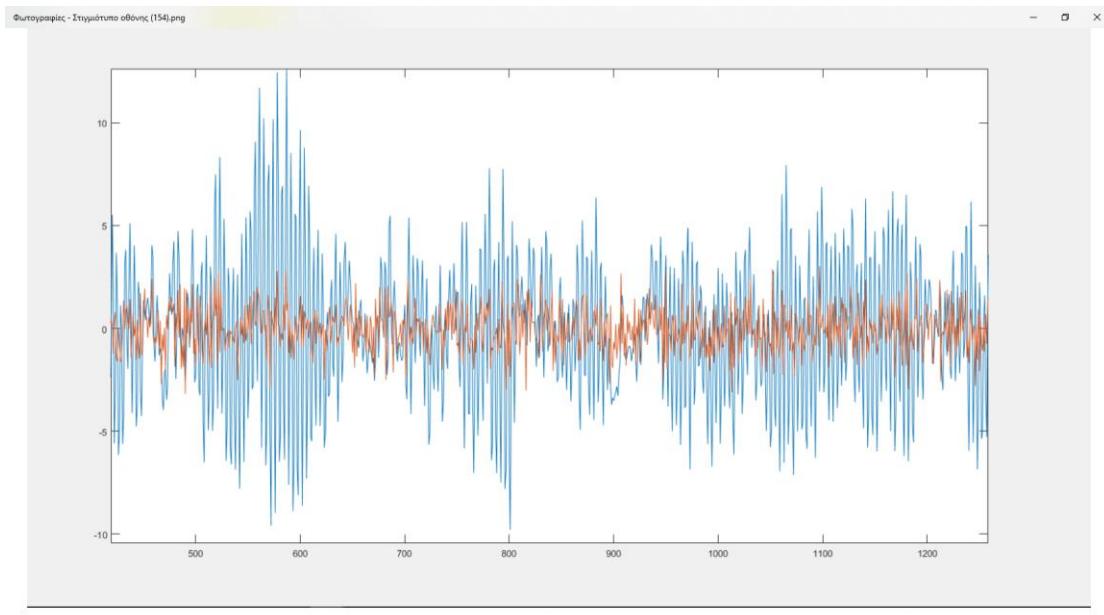
N=2bits και τάξη προβλέπτη p=4



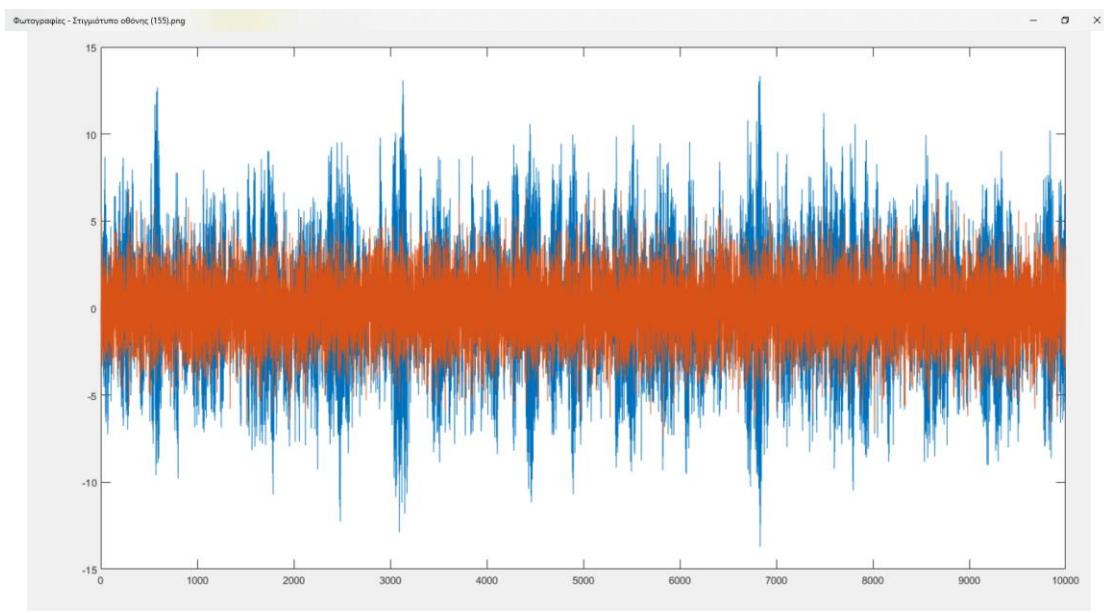


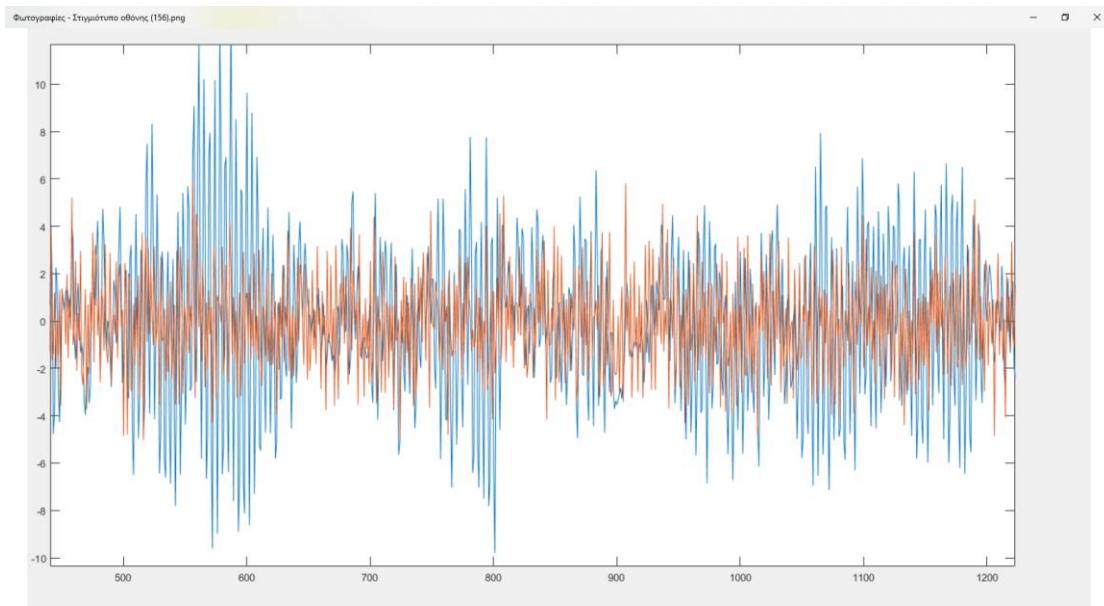
N=3bits και τάξη προβλέπτη p=4



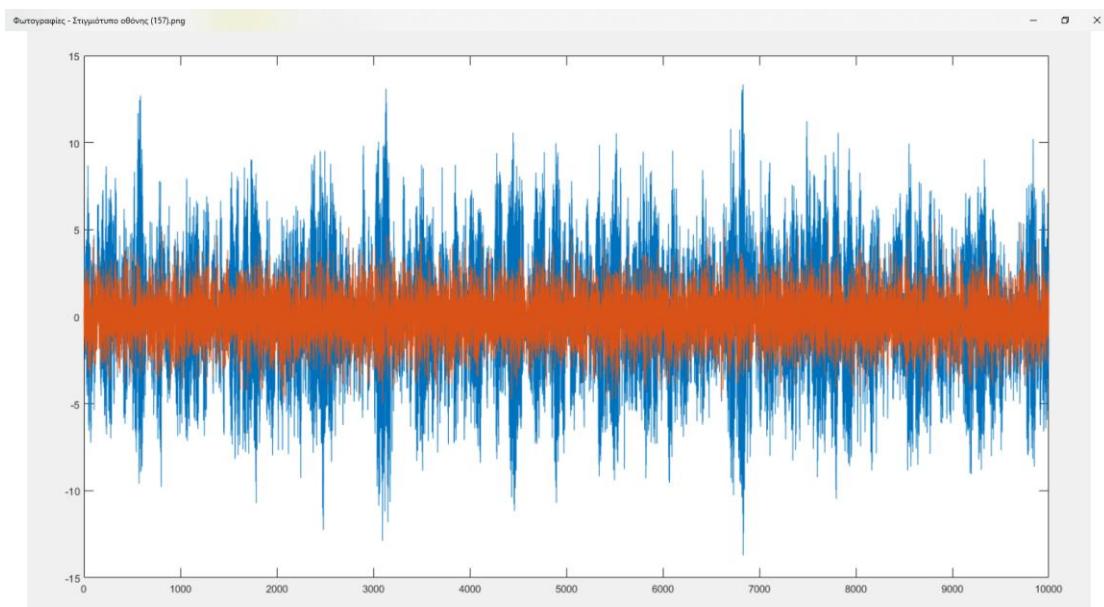


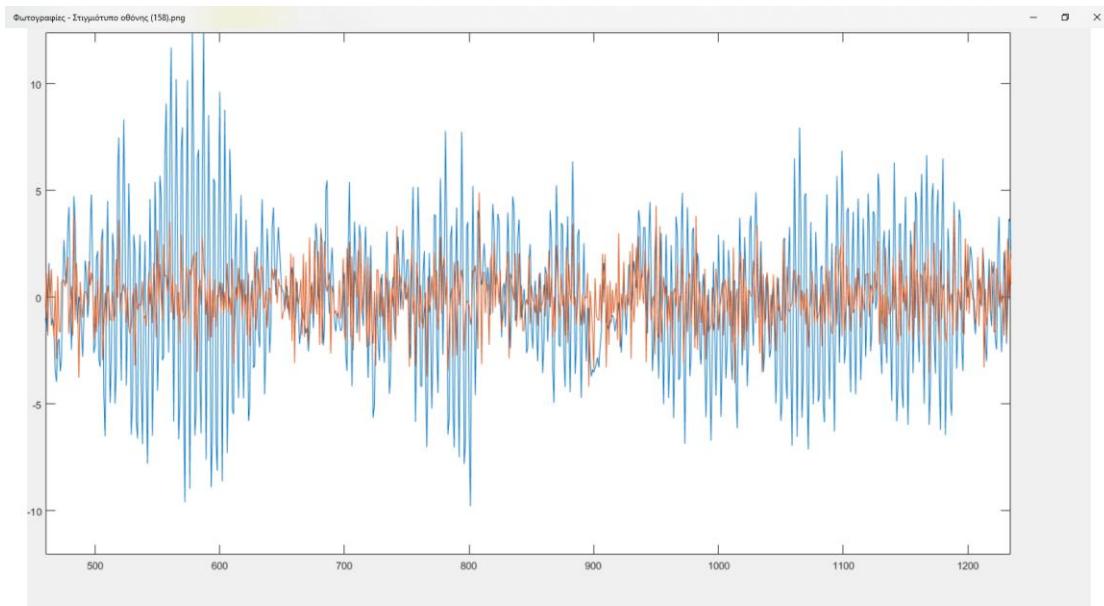
N=1bits και τάξη προβλέπτη p=8



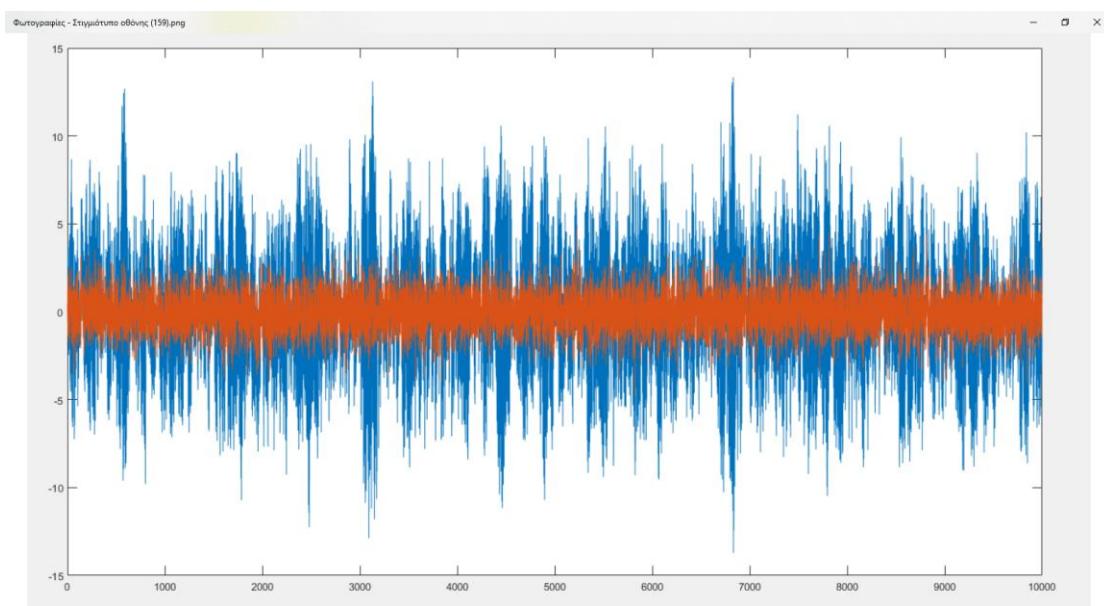


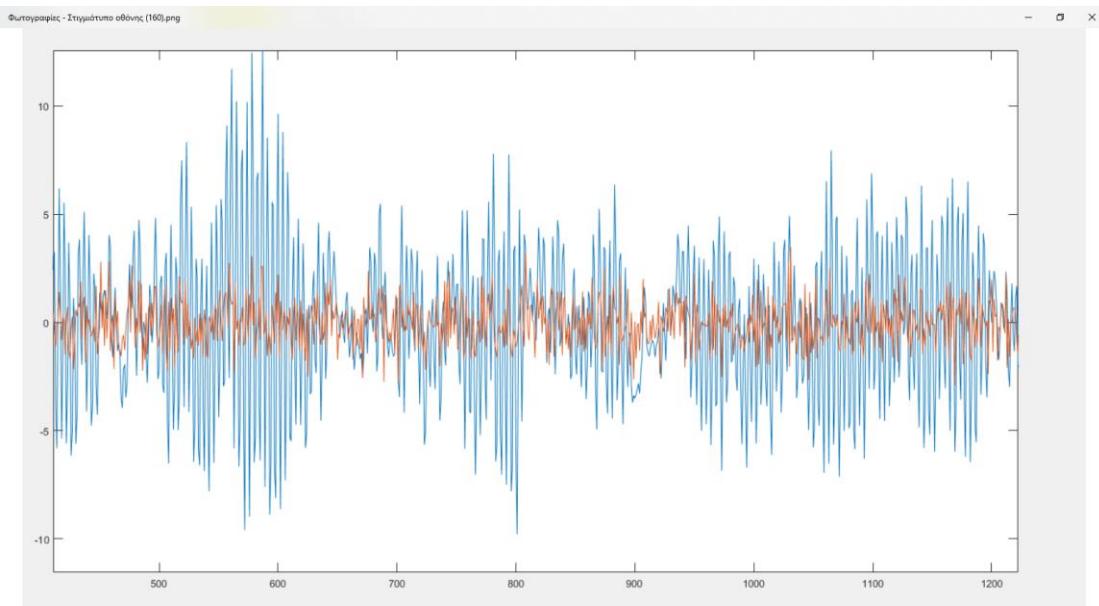
N=2bits και τάξη προβλέπτη p=8





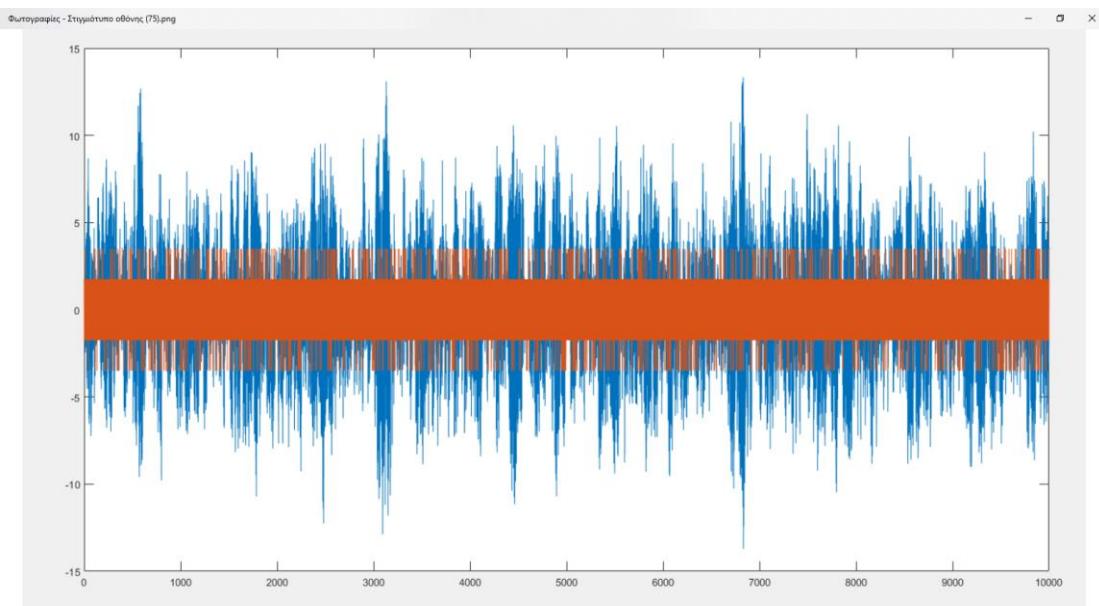
N=3bits και τάξη προβλέπτη p=8

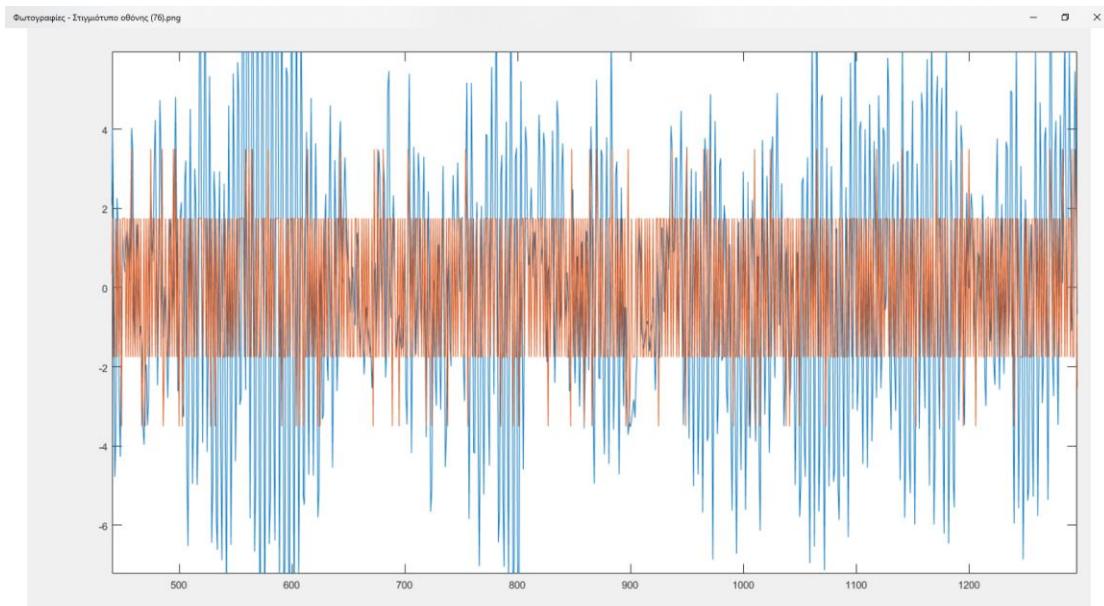




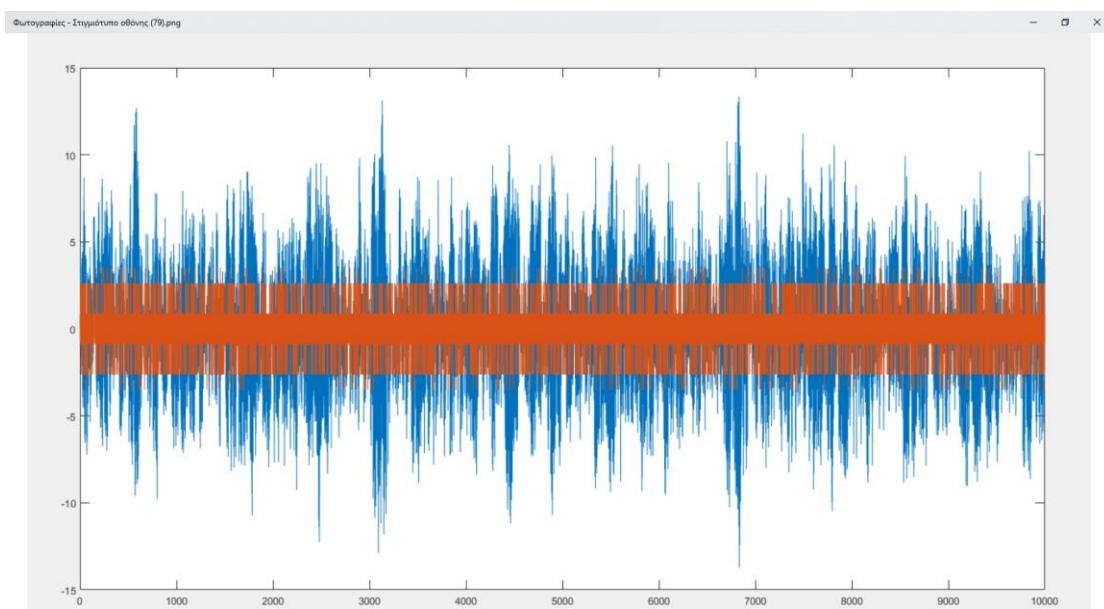
Στα παρακάτω στιγμιότυπα φαίνεται το γράφημα του αρχικού σήματος και το κβαντισμένο σφάλμα πρόβλεψης για τις αντίστοιχες παραμέτρους:

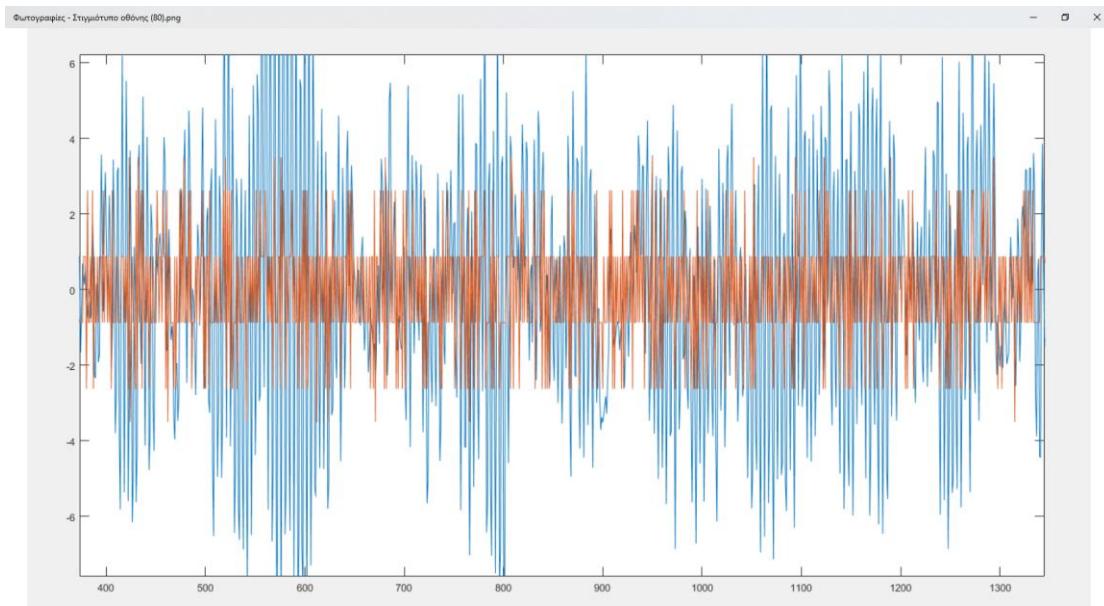
N=1bit και τάξη προβλέπτη p=4



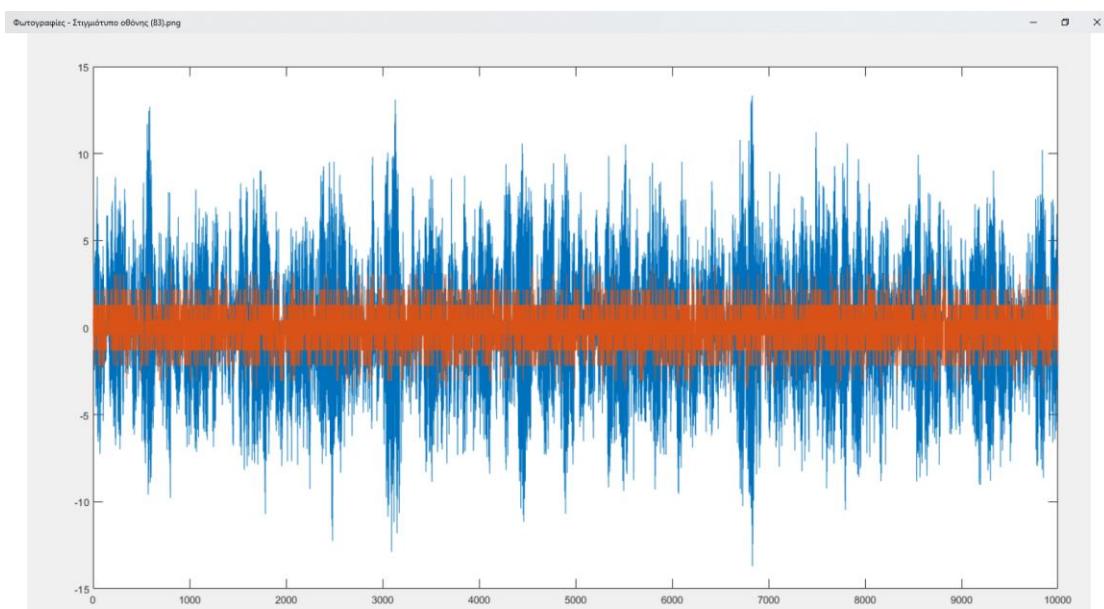


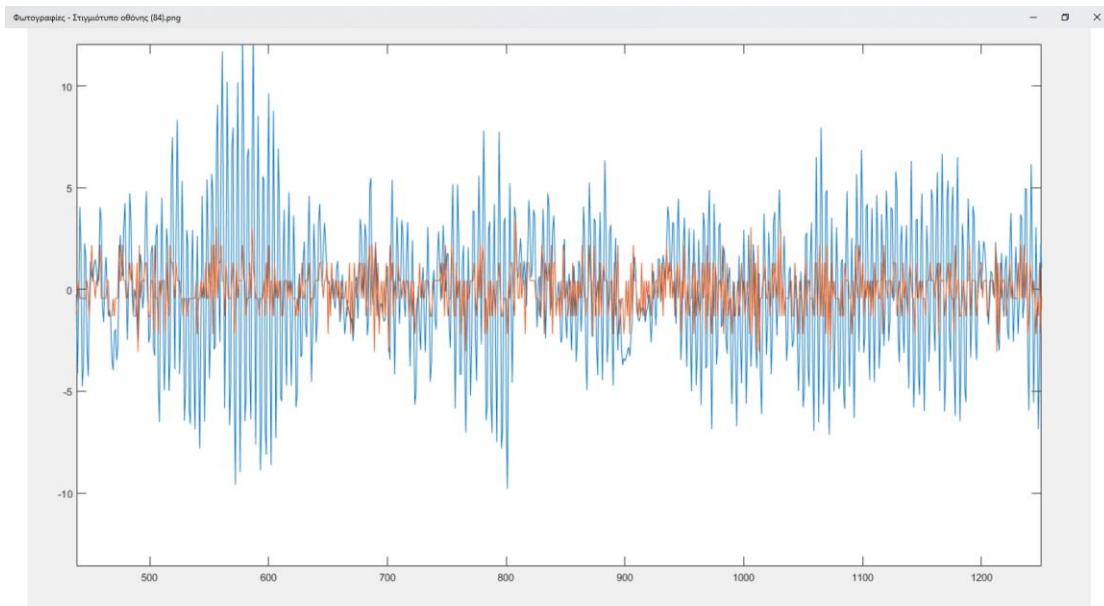
N=2bits και τάξη προβλέπτη p=4



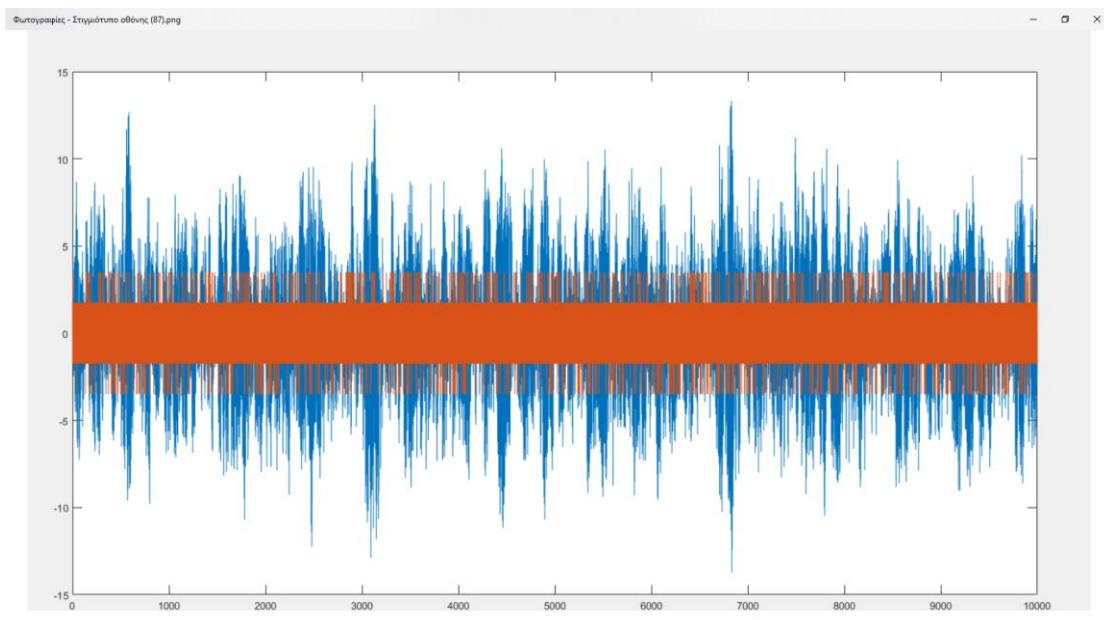


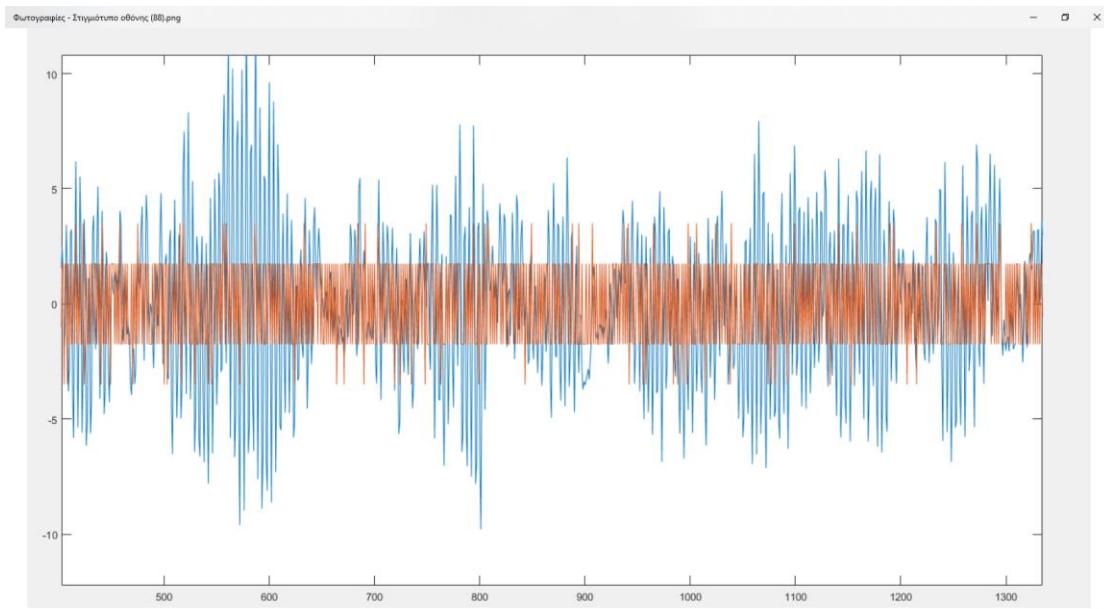
N=3bits και τάξη προβλέπτη p=4



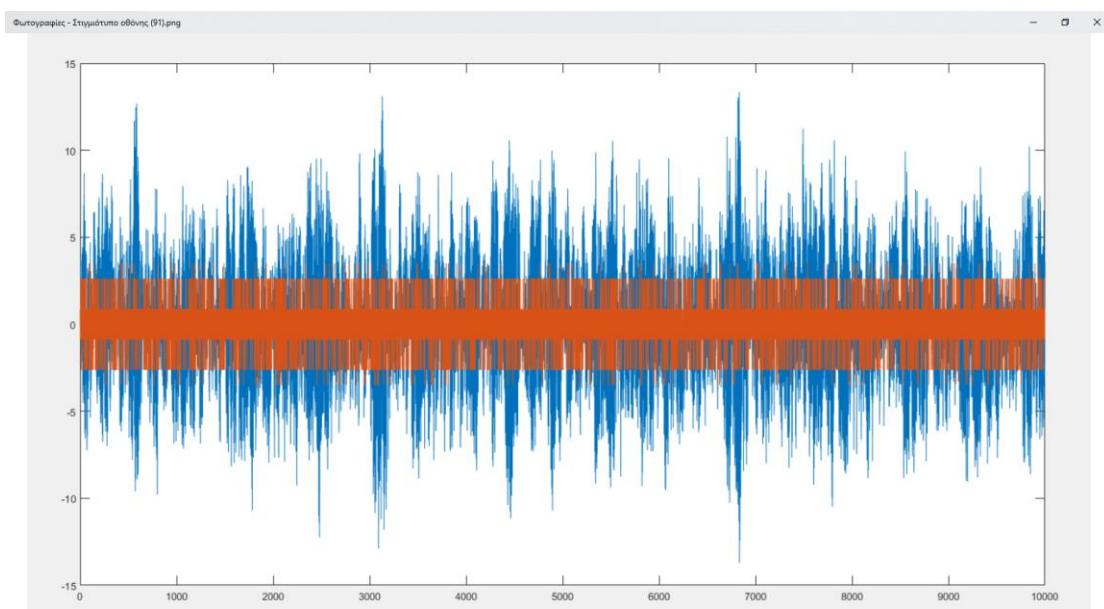


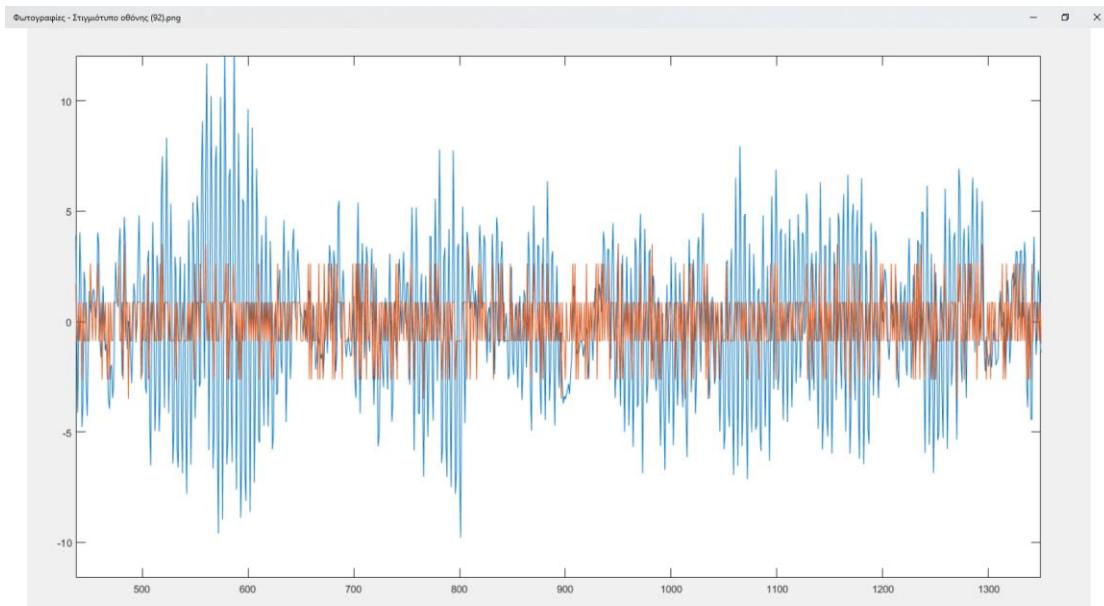
N=1bit και τάξη προβλέπτη p=8



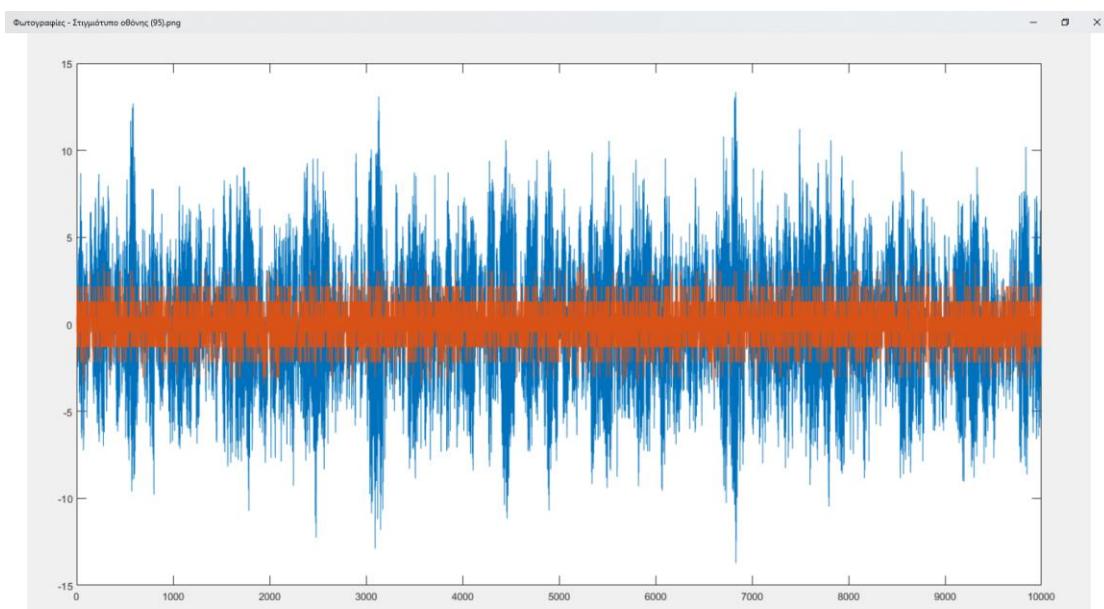


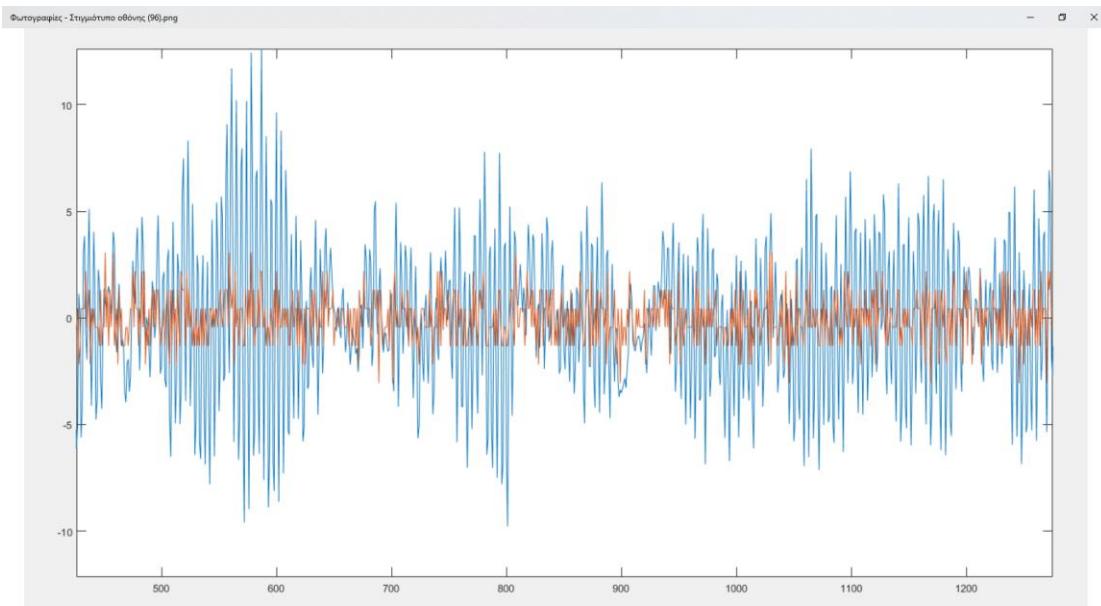
N=2bit και τάξη προβλέπτη p=8





N=3bit και τάξη προβλέπτη p=8





Αρχικά αυτό που παρατηρούμε είναι ότι το σφάλμα πρόβλεψης έχει πολύ μικρότερη δυναμική περιοχή από το σήμα εισόδου. Αυτός είναι και ο σκοπός μας από την θεωρία εξάλλου, να ελαχιστοποιήσουμε την διασπορά του σήματος σφάλματος έτσι ώστε αυτό να παρουσιάζει μικρή δυναμική περιοχή και να μπορεί να περιγραφεί ικανοποιητικά από μικρό αριθμό δυαδικών ψηφίων. Άλλα εδώ το επιβεβαιώνουμε και πειραματικά.

Επίσης είναι ξεκάθαρο πως με την χρήση περισσότερων bit για την κωδικοποίηση των δειγμάτων (δηλαδή με αύξηση του αριθμού N) το σήμα σφάλματος έχει καλύτερη συμπεριφορά. Αυτό είναι αναμενόμενο αν σκεφτούμε πως στην κωδικοποίηση DPCM για την πρόβλεψη του δείγματος χρησιμοποιούμε τις r (όπου r η τάξη του προβλέπτη) προηγούμενες κβαντισμένες τιμές του σήματος εισόδου και όχι τις τιμές του αυτές καθαυτές. Είναι επομένως λογικό όταν ο κβαντιστής χρησιμοποιεί περισσότερα bits(δηλ. έχει περισσότερες στάθμες) να μικραίνει το σφάλμα κβάντισης($\hat{y}(n) - y(n)$) , άρα να μικραίνει και η απόκλιση της πρόβλεψης μας από την πραγματική τιμή του σήματος εισόδου.

Το περίεργο εδώ είναι ότι η αύξηση του αριθμού των συντελεστών του διανύσματος του φίλτρου πρόβλεψης(αριθμός r) δεν έχει καμία επίπτωση στο σφάλμα πρόβλεψης. Δηλαδή ενώ θα περιμέναμε ότι λαμβάνοντας υπ' όψin περισσότερες τιμές του σήματος εισόδου η πρόβλεψη μας για την επόμενη τιμή θα ήταν καλύτερη εδώ βλέπουμε ότι είτε πάρουμε τις 4 προηγούμενες τιμές είτε τις 8 προηγούμενες η πρόβλεψη μας θα είναι το ίδιο κοντά στην πραγματική τιμή του σήματος εισόδου. Εδώ θα δώσω μια εξήγηση γιατί συμβαίνει αυτό και θα το επιβεβαιώσουμε παρακάτω στο ερώτημα (3) από τους συντελεστές του προγνώστη για $r > 4$.

Εφόσον μας δίνετε ότι το σήμα εισόδου έχει παραχθεί από μια Auto Regressive διαδικασία τάξης 4 υποθέτω ότι κάθε δείγμα του σήματος θα έχει μεγάλη συσχέτιση με τα 4 προηγούμενα του. Επομένως για να πάρω μια καλή πρόβλεψη χρειάζομαι τουλάχιστον τις 4 προηγούμενες τιμές του κάθε δείγματος πολλαπλασιασμένες με τους αντίστοιχους συντελεστές(αυτό το επιβεβαίωσα και πειραματικά παρότι δεν φαίνεται πουθενά στην

αναφορά, ότι δηλαδή θέλω τουλάχιστον 4 τιμές για καλή πρόβλεψη, π.χ με $p=3$ το μέσο τετραγωνικό σφάλμα πρόβλεψης με $N=1$ bit είναι 13.5 ενώ με $p=4$ είναι 4.5). Αφού λοιπόν δεν βλέπω βελτίωση στο σφάλμα πρόβλεψης για $p>4$ συμπεραίνω ότι τα δείγματα του σήματος μου έχουν ικανοποιητική αυτοσυσχέτιση με τα 4 προηγούμενα τους και μόνο. Άρα λαμβάνοντας υπ' όψιν στην πρόβλεψη μας $p>4$ προηγούμενες τιμές του σήματος δεν μας επιφέρει καλύτερη πρόβλεψη.

Σημείωση: Για να λάβω το μη κβαντισμένο σφάλμα πρόβλεψης τροποποιώ την συνάρτηση `my_dpcmenco` ώστε να επιστρέφει και την μη-κβαντισμένη τιμή του σφάλματος πρόβλεψης για το συγκεκριμένο ερώτημα και μόνο. Στην κανονική της μορφή όπως δίνεται στο τέλος της αναφοράς δεν φαίνεται αυτή η συμπεριφορά.

(3)

Για τάξη προβλέπτη $p=4:8$ βρίσκω το μέσο τετραγωνικό σφάλμα για αριθμό bits $N=1,2,3$ με τις εντολές (με την τροποποιημένη συνάρτηση `my_dpcmenco` όπως σημειώνεται και πιο πάνω)

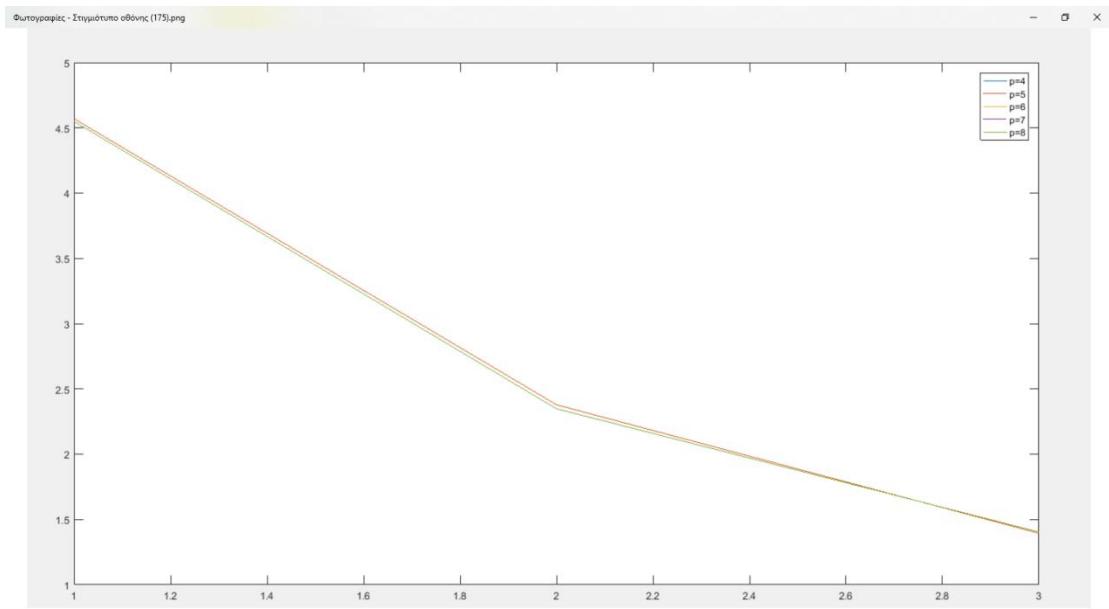
```
[indx,quanterr,error]=my_dpcmenco(2,x,centers,partition,predictor);
```

```
[decodedx,dec_quanterr]=my_dpcmdeco(indx,centers,predictor);
```

```
sum(error.^2)/length(error)
```

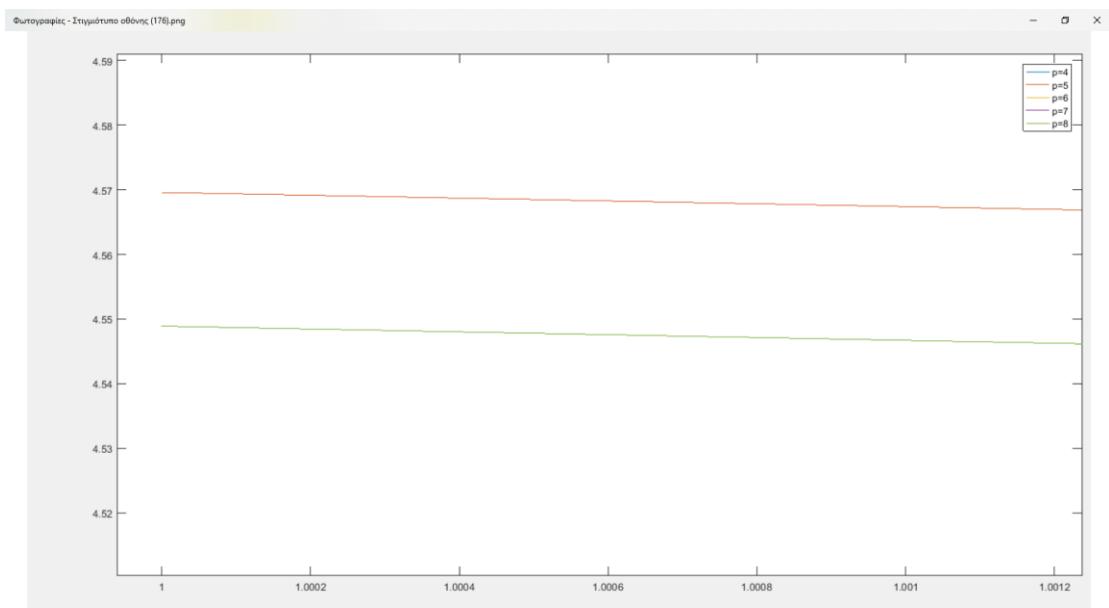
και για κάθε p φτιάχνω ένα διάνυσμα με τις τιμές αυτές.

Ακολουθεί το γράφημα που δείχνει το μέσο τετραγωνικό σφάλμα του συστήματος DPCM συναρτήσει του N (αριθμός bits του κβαντιστή) για $p=4:8$.

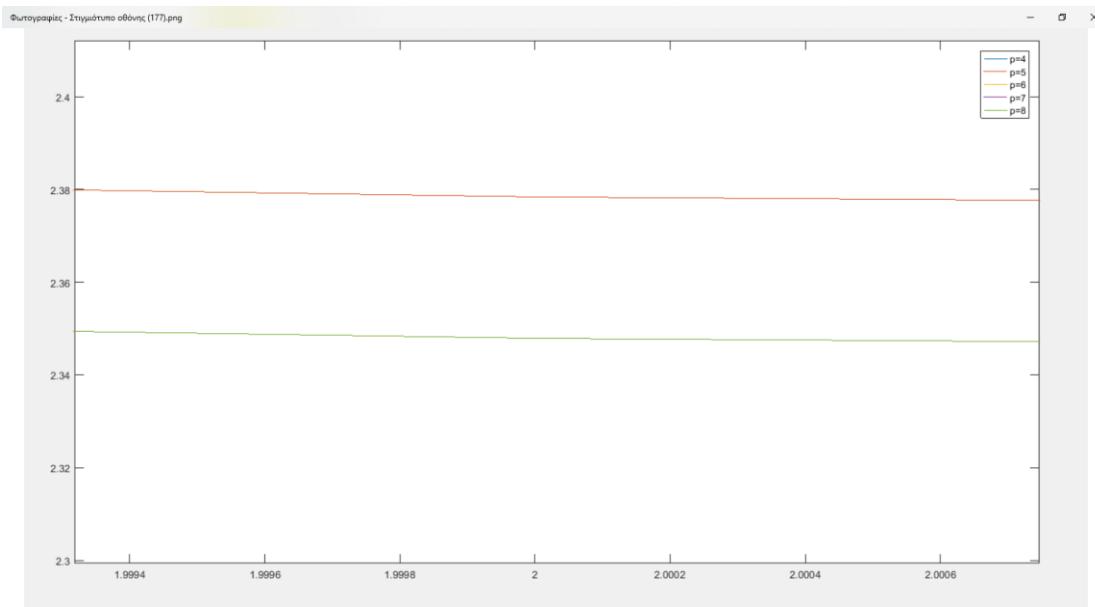


Επειδή οι καμπύλες διαφέρουν ελάχιστα παραθέτω το ίδιο γράφημα μεγενθυμένο στις τιμές του N=1, 2 ,3 bit.

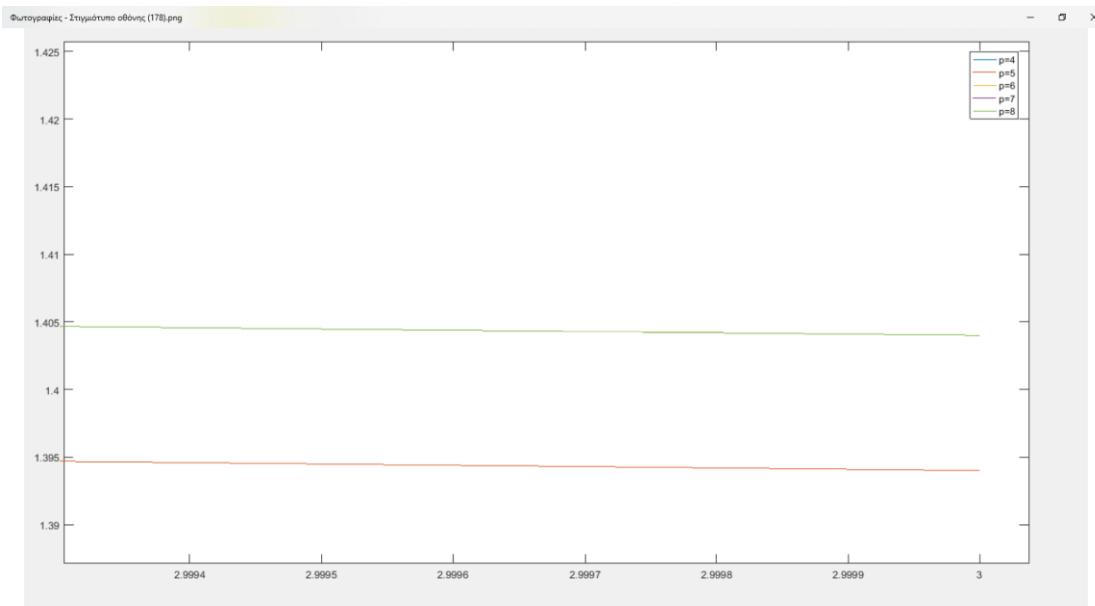
N=1bit



N=2bit



N=3bit



Βλέπω ότι οι καμπύλες συμπίπτουν για $p=4$ και $p=5$ ενώ υπάρχει μικρή διαφορά με $p=6,7,8$ των οποίων οι καμπύλες και πάλι συμπίπτουν. Παρατηρώ ότι έχω σημαντική βελτίωση του μέσου τετραγωνικού σφάλματος με την αύξηση του αριθμού των bit N αλλά και πάλι φαίνεται ότι η τάξη του προγνώστη p έχει ελάχιστη συνεισφορά στο σφάλμα πρόβλεψης.

Ακόμη χρησιμοποιώντας τον τύπο $SQNR = 3 * 4^N * E[\bar{X}^2]$, όπου $E[\bar{X}^2] = \frac{E[X^2]}{x_{max}^2}$ παίρνω τις τιμές για τον λόγο σήματος προς θόρυβο κράντισης όπως φαίνεται στα παρακάτω στιγμιότυπα

Φωτογραφίες - Στηγμένο οθόνη (183).png

```

>> (sum(x.^2)/length(x)) / (max(x)^2)
ans =
0.0735

>> sqnr=3*4*0.0735
sqnr =
0.8820

>> sqnr_db=10*log10(0.8820)
sqnr_db =
-0.5453

>> sqnr=3*(4^2)*0.0735
sqnr =
3.5280

>> sqnr_db=10*log10(3.5280)
sqnr_db =
5.4753

```

Φωτογραφίες - Στηγμένο οθόνη (184).png

```

3.5280
>> sqnr_db=10*log10(3.5280)
sqnr_db =
5.4753

>> sqnr=3*(4^3)*0.0735
sqnr =
14.1120

>> sqnr_db=10*log10(14.1120)
sqnr_db =
11.4959

```

Και όπως βλέπουμε από τα αποτελέσματα επιβεβαιώνεται και πειραματικά αυτό που γνωρίζουμε από τη θεωρία ότι κάθε επιπλέον bit αυξάνει το SQNR κατά 6dB (ή αλλιώς η παραμόρφωση μειώνεται κατά 6dB για κάθε αύξηση κατά 1bit του κώδικα αναπαράστασης).

Παρακάτω φαίνονται οι τιμές των συντελεστών του προβλέπτη για $p=4:8$. Το διάνυσμα a αντιστοιχεί στις πραγματικές τιμές του προγνώστη ενώ παραθέτω και το διάνυσμα predictor που περιέχει τις κβαντισμένες τιμές των συντελεστών (με $N=8$ bit και δυναμική περιοχή [-2,2]) και είναι το διάνυσμα συντελεστών που χρησιμοποιούν πομπός και δέκτης (για να λειτουργούν σε συμφωνία).

Επισημαίνεται πως προσθέτω την τιμή 0 στην αρχή του διανύσματος συντελεστών a και το χρησιμοποιώ όπως περιγράφω στο ερώτημα (1).

Για p=4

$$a = \begin{pmatrix} 0 \\ 1,3887 \\ -1,5212 \\ 1,2112 \\ -0,3016 \end{pmatrix} \text{ και predictor} = \begin{pmatrix} -0,0078 \\ 1,3828 \\ -1,5234 \\ 1,2109 \\ -0,3047 \end{pmatrix}$$

Για p=5

$$a = \begin{pmatrix} 0 \\ 1,3881 \\ -1,5185 \\ 1,2087 \\ -0,2984 \\ -0,0025 \end{pmatrix} \text{ και predictor} = \begin{pmatrix} -0,0078 \\ 1,3828 \\ -1,5234 \\ 1,2109 \\ -0,3047 \\ -0,0078 \end{pmatrix}$$

Για p=6

$$a = \begin{pmatrix} 0 \\ 1,3881 \\ -1,5191 \\ 1,2117 \\ -0,3022 \\ 0,0011 \\ -0,0026 \end{pmatrix} \text{ και predictor} = \begin{pmatrix} -0,0078 \\ 1,3828 \\ -1,5234 \\ 1,2109 \\ -0,3047 \\ 0,0078 \\ -0,0078 \end{pmatrix}$$

Για p=7

$$a = \begin{pmatrix} 0 \\ 1,3880 \\ -1,5193 \\ 1,2109 \\ -0,2975 \\ -0,0051 \\ 0,0033 \\ -0,0045 \end{pmatrix} \text{ και predictor} = \begin{pmatrix} -0,0078 \\ 1,3828 \\ -1,5234 \\ 1,2109 \\ -0,3047 \\ -0,0078 \\ 0,0078 \\ -0,0078 \end{pmatrix}$$

Για p=8

$$a = \begin{pmatrix} 0 \\ 1,3878 \\ -1,5192 \\ 1,2106 \\ -0,2998 \\ 0,0050 \\ -0,0095 \\ 0,0073 \\ -0,0085 \end{pmatrix} \text{ και predictor} = \begin{pmatrix} -0,0078 \\ 1,3828 \\ -1,5234 \\ 1,2109 \\ -0,3047 \\ 0,0078 \\ -0,0078 \\ 0,0078 \\ -0,0078 \end{pmatrix}$$

Το διάνυσμα συντελεστών a προκύπτει από την λύση του γραμμικού συστήματος $Ra=r$, όπου R : ο πίνακας αυτοσυσχέτισης διάστασης pxp του οποίου το (i,j) στοιχείο είναι το $Rx(i-j)$ και r : διάνυσμα αυτοσυσχέτισης διάστασης $px1$ με στοιχείο i το $Rx(i)$. Επομένως οι

συντελεστές του διανύσματος $a(i+1)$ εκφράζουν το 'βάρος' με το οποίο συνεισφέρει η $i_{οστή}$ προηγούμενη τιμή του σήματος εισόδου στην πρόβλεψη της τρέχουσας τιμής.

Αυτό που παρατηρούμε εδώ από τα διανύσματα συντελεστών του προβλέπτη είναι ότι για $r>4$ οι συντελεστές που αντιστοιχούν στις προηγούμενες 5,6,7 και 8 τιμές του σήματος εισόδου είναι πάρα πολύ κοντά στο 0. Από αυτό συμπεραίνουμε ότι μια τιμή του σήματος εισόδου έχει αξιόλογη (χρήσιμη για να έχουμε καλή πρόβλεψη) αυτοσυσχέτιση με τις 4 προηγούμενες τιμές από αυτήν.

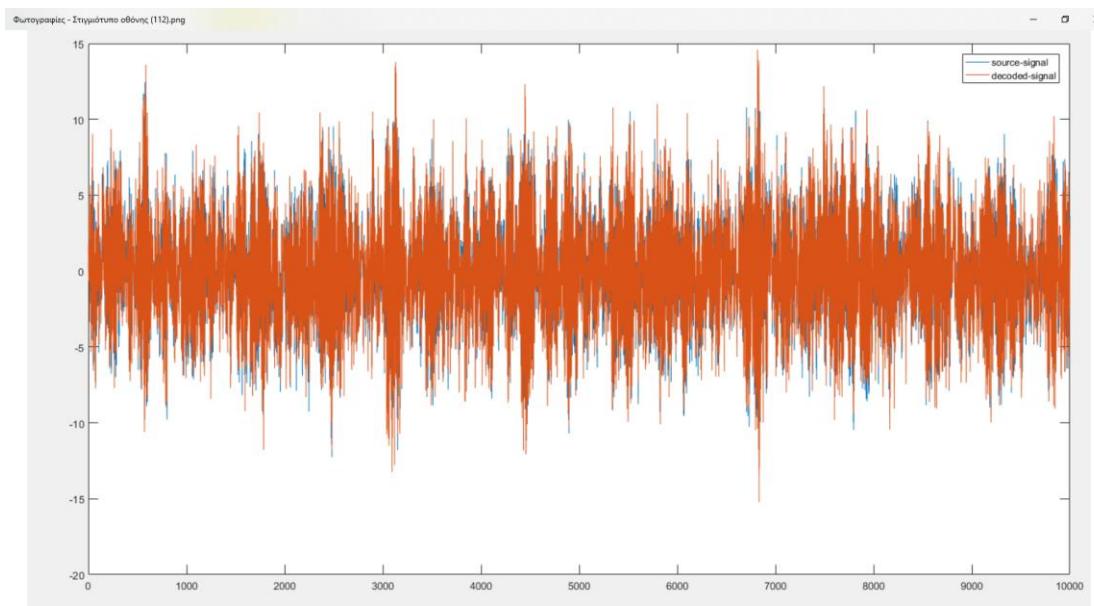
Επομένως για να προβλέψουμε ικανοποιητικά κάθε τιμή του σήματος εισόδου έχει νόημα να λάβουμε υπ' όψιν μας τις 4 προηγούμενες τιμές από αυτήν. Αν συμπεριλάβουμε και παλαιότερες τιμές ($r>4$) τότε όπως είδαμε από τους συντελεστές των διανυσμάτων πιο πάνω αυτές δεν προσφέρουν καμία επιπλέον πληροφορία στην πρόβλεψη.

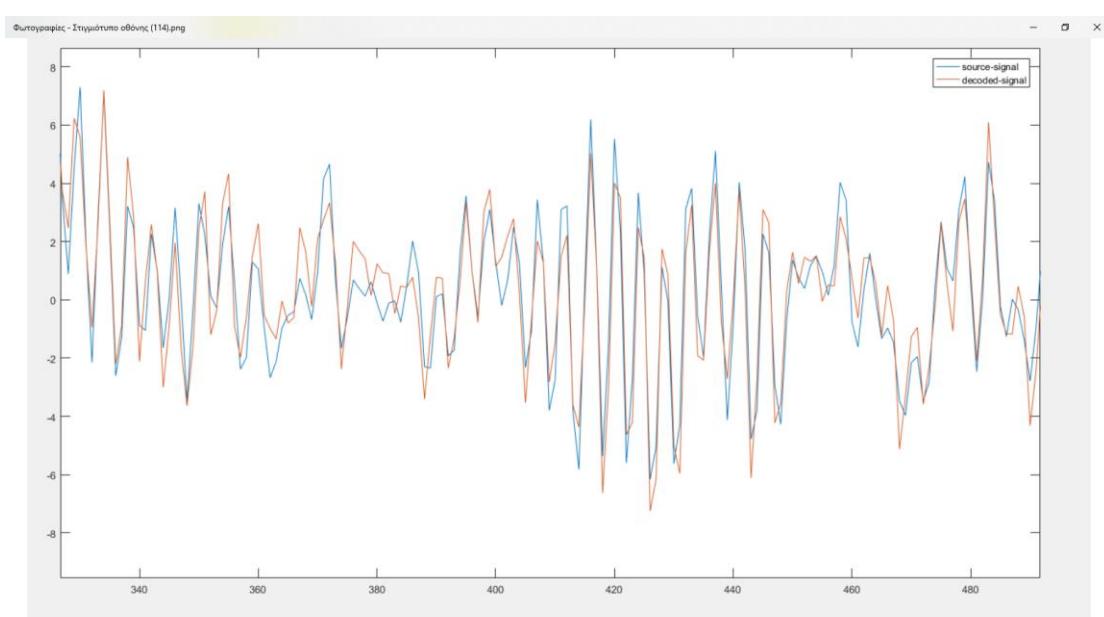
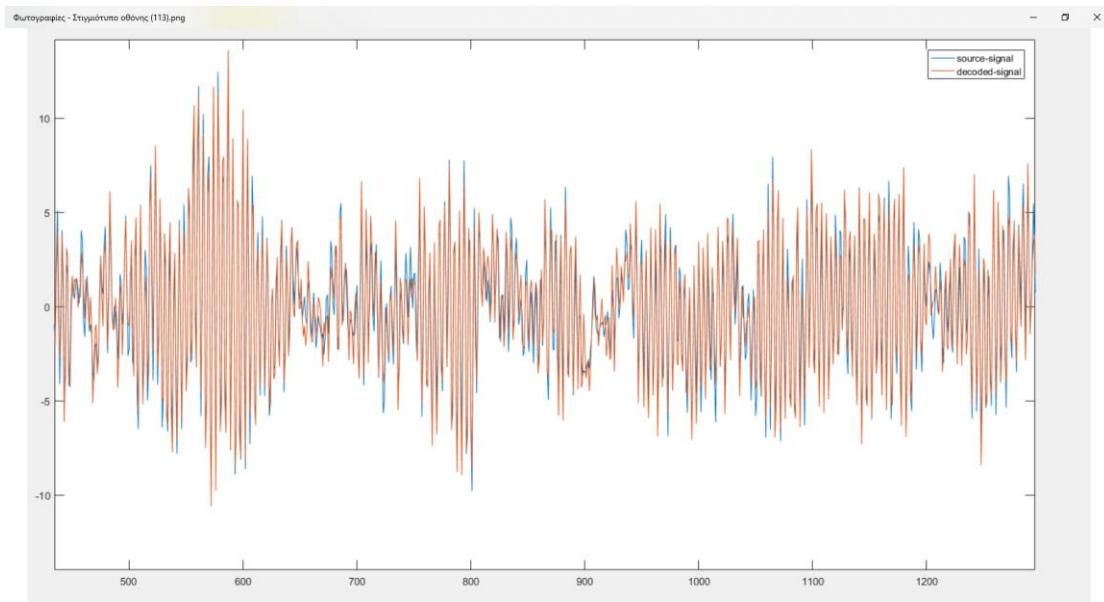
Εδώ επιβεβαιώνεται και η εξήγηση που είχα δώσει στο ερώτημα (2) γιατί δεν βελτιώνεται το σήμα σφάλματος με την αύξηση των συντελεστών του προβλέπτη (διότι οι συντελεστές που προστίθενται είναι πολύ κοντά στο 0).

(4)

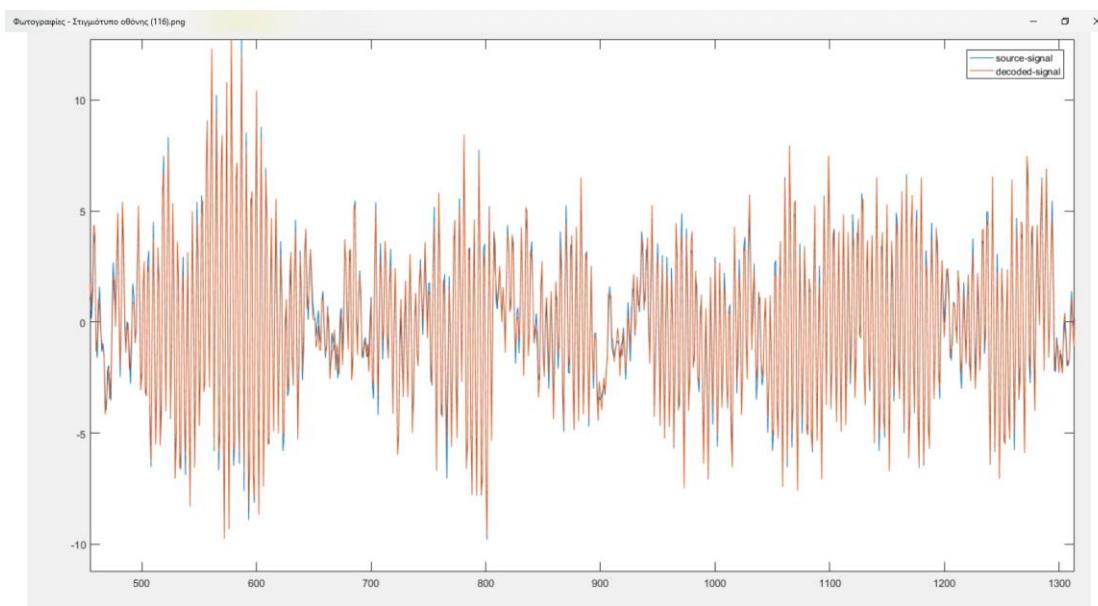
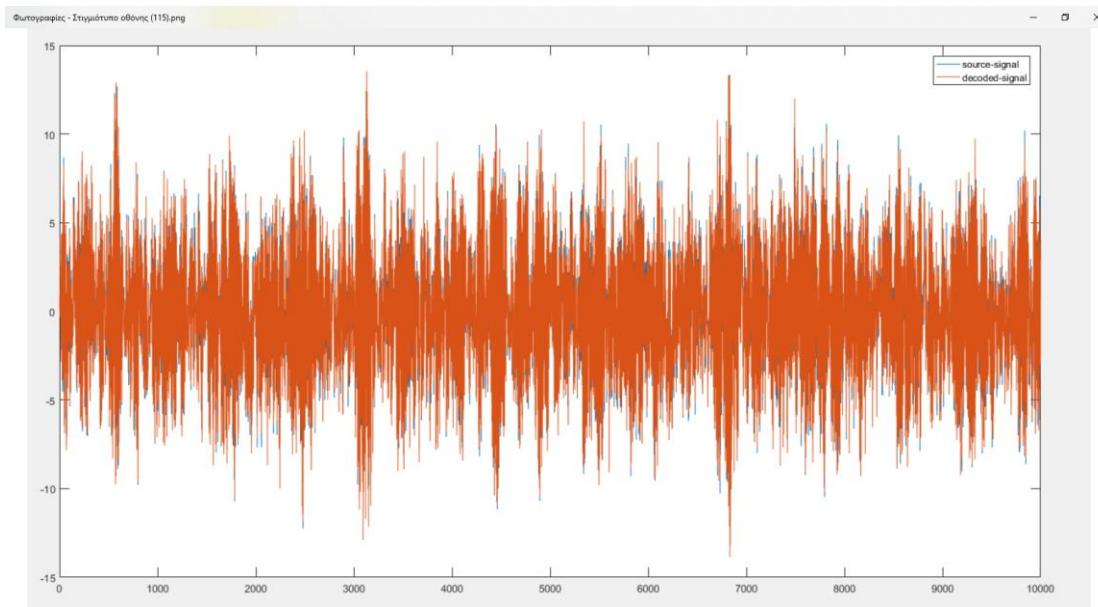
Ακολουθούν τα διαγράμματα του σήματος και της ανακατασκευής του στον δέκτη. Για κάθε περίπτωση παραθέτω 3 στιγμιότυπα του γραφήματος, με διαφορετική μεγέθυνση το καθένα ώστε να παρατηρήσουμε λεπτομερέστερα τις διαφορές. Να σημειώσω εδώ ότι για όλες τις περιπτώσεις η μεγέθυνση έχει γίνει στον ίδιο σημείο ώστε να φανούν οι διαφορές ανάμεσα σε κάθε συνδυασμό τιμών r και N .

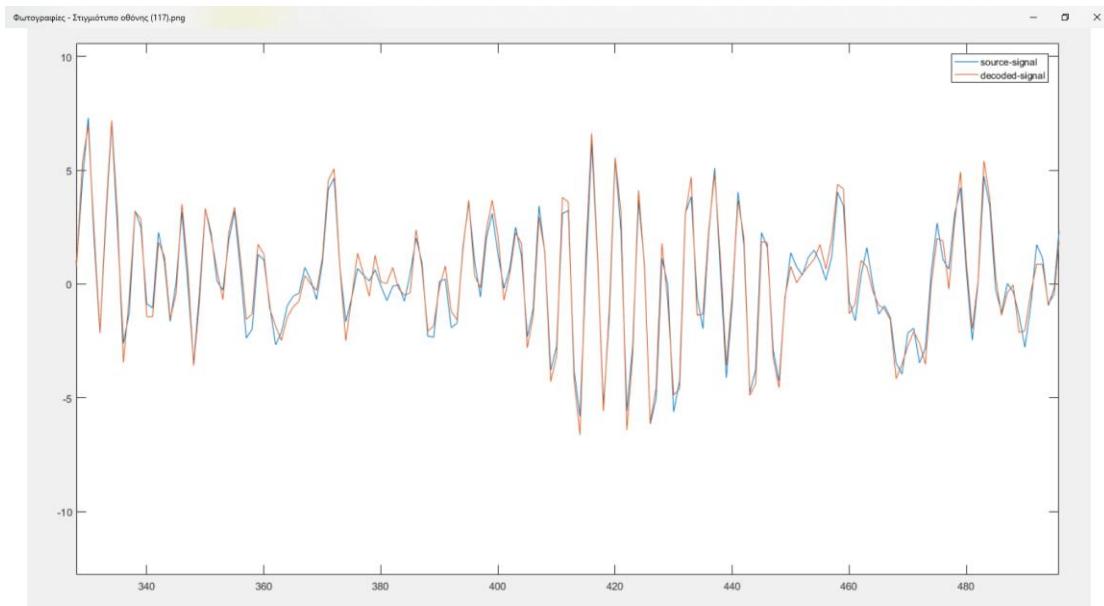
Τάξη φίλτρου πρόβλεψης $r=4$ και $N=1$ bit



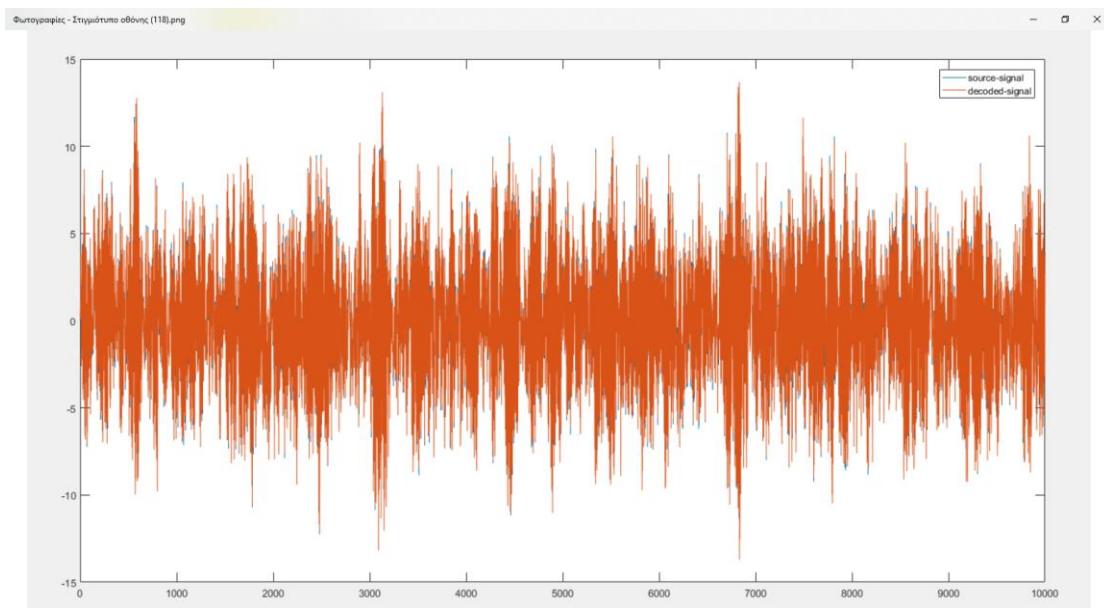


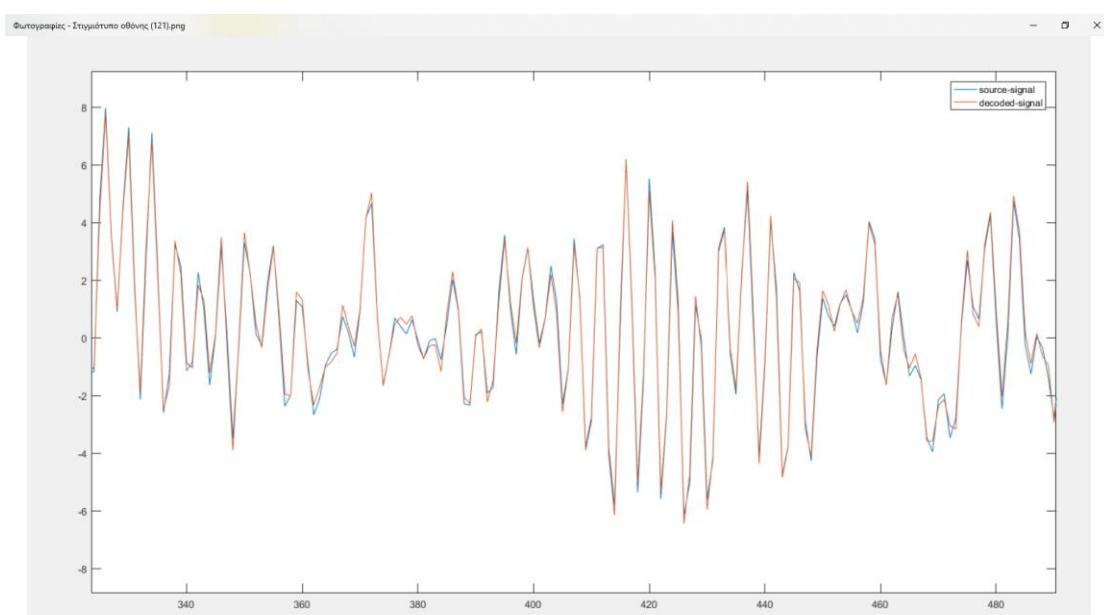
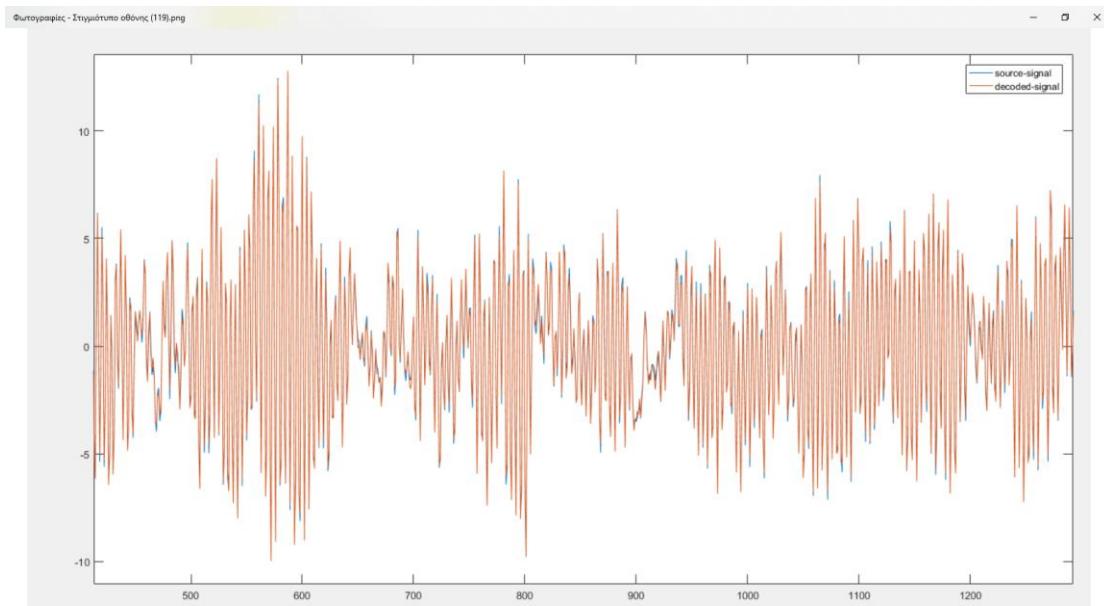
Τάξη φίλτρου πρόβλεψης $p=4$ και $N=2$ bit



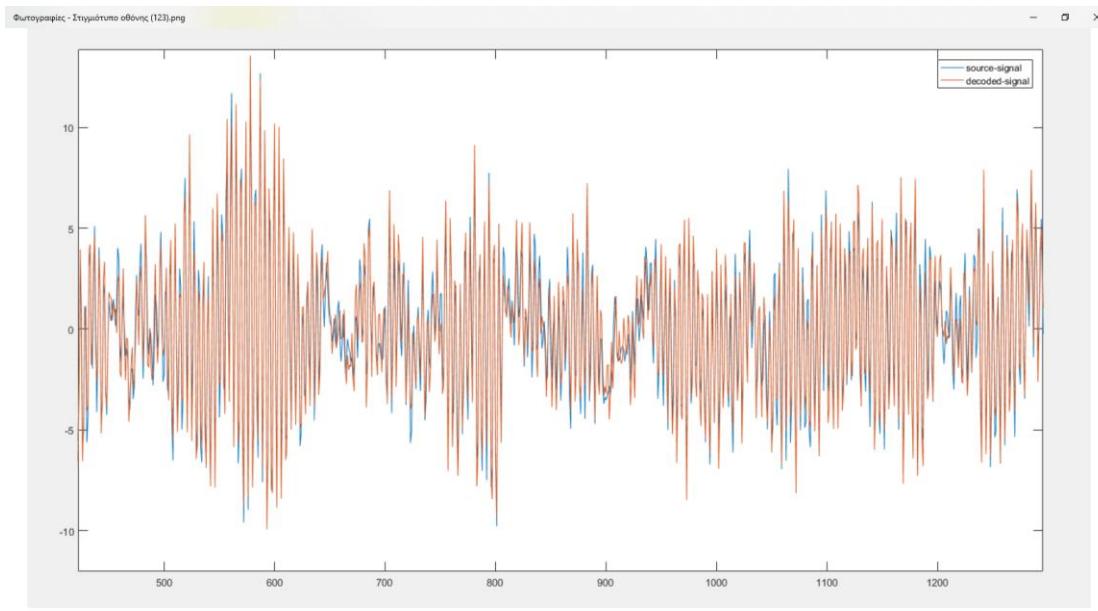
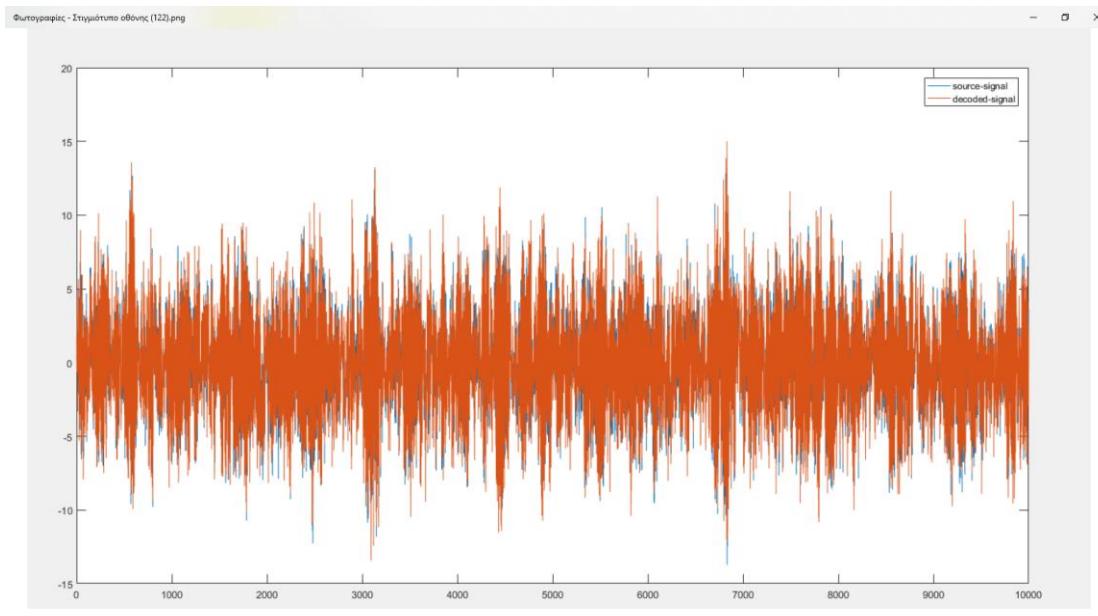


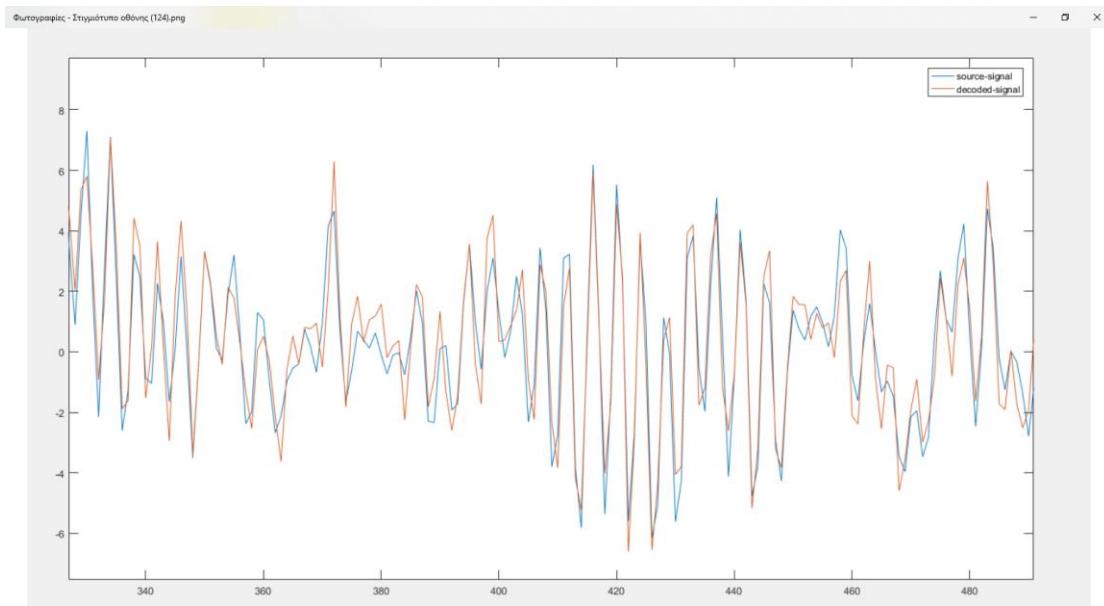
Τάξη φίλτρου πρόβλεψης $p=4$ και $N=3$ bit



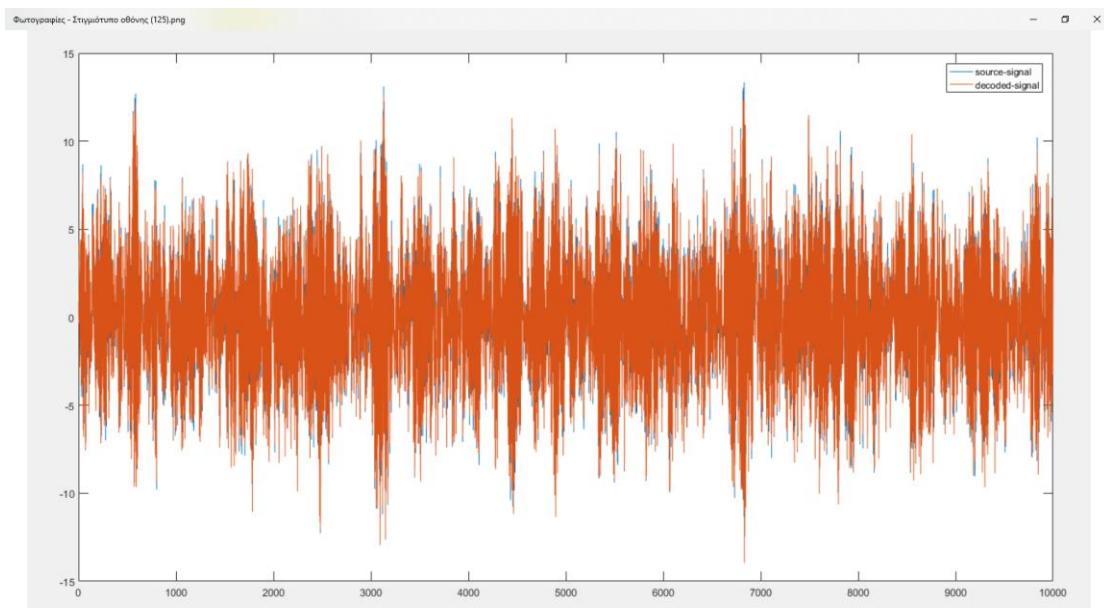


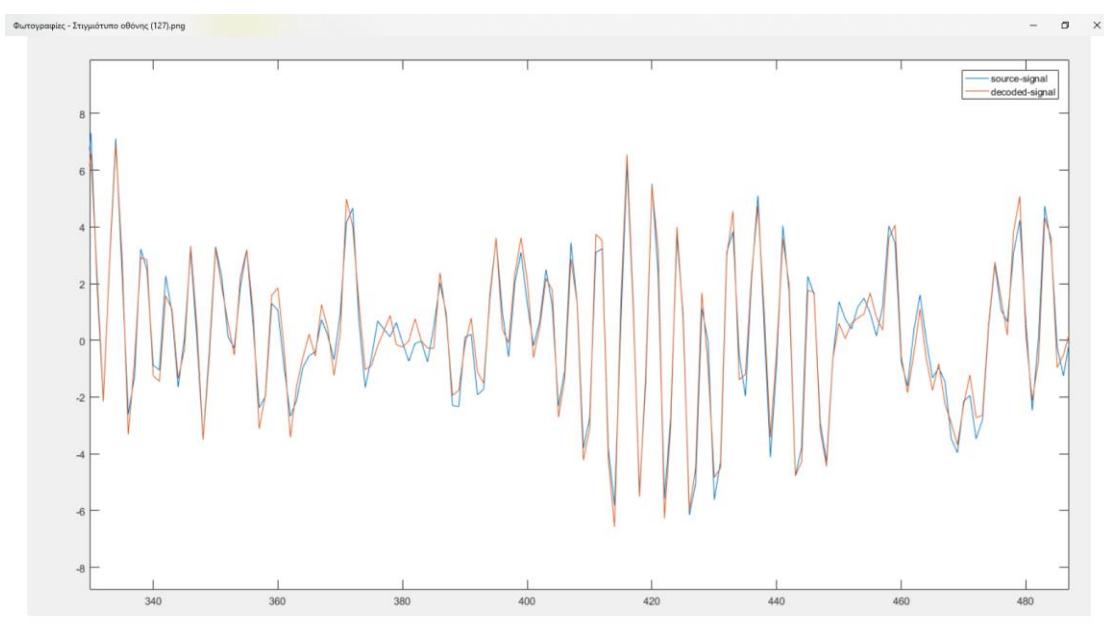
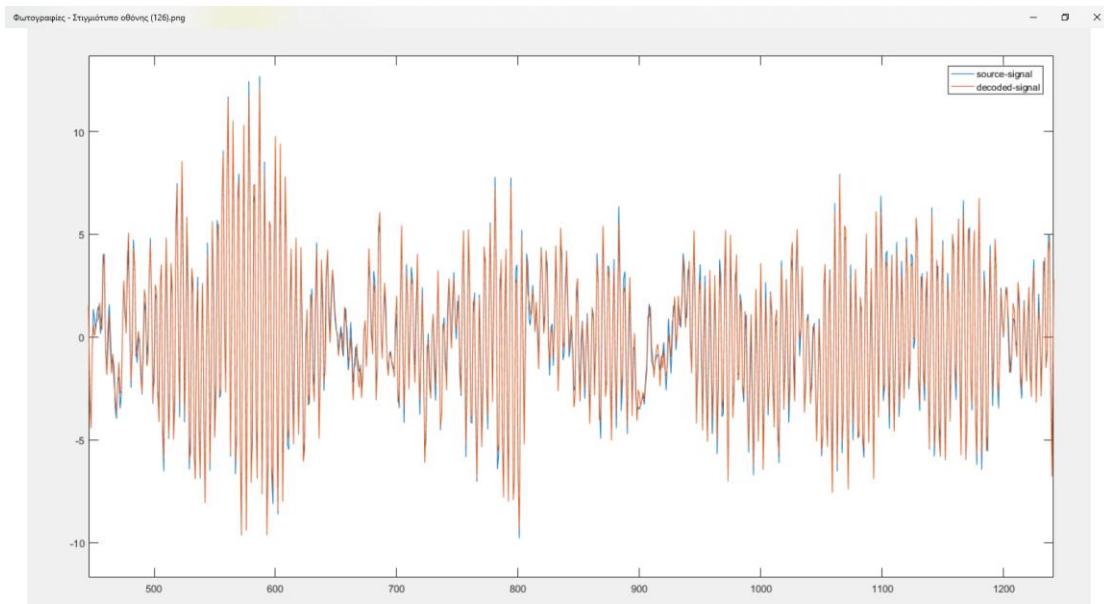
Τάξη φίλτρου πρόβλεψης $p=8$ και $N=1$ bit



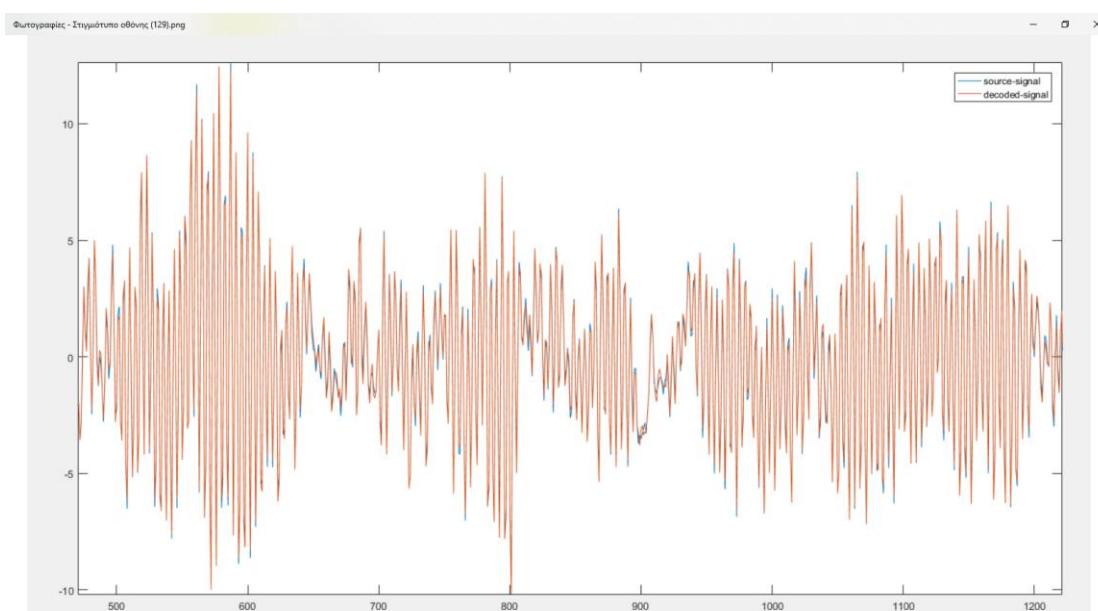
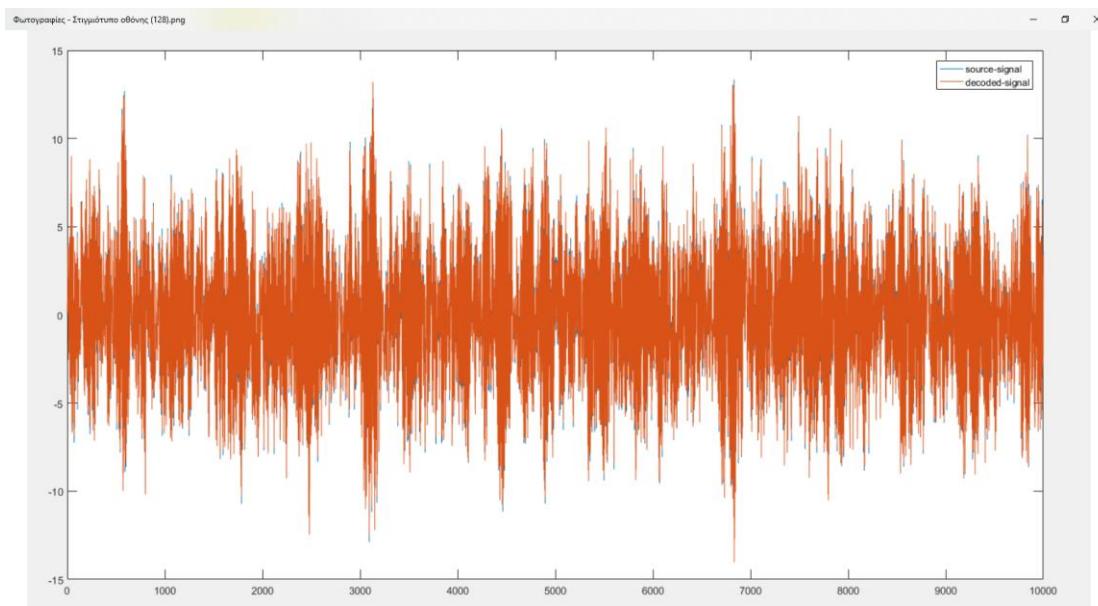


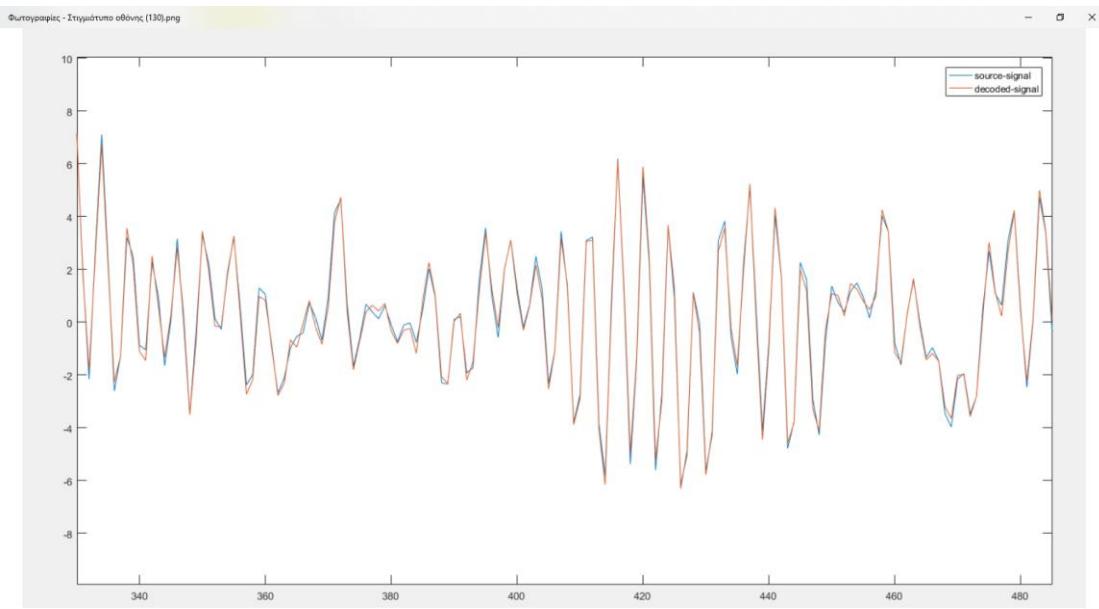
Τάξη φίλτρου πρόβλεψης $p=8$ και $N=2$ bit





Τάξη φίλτρου πρόβλεψης $p=8$ και $N=3$ bit

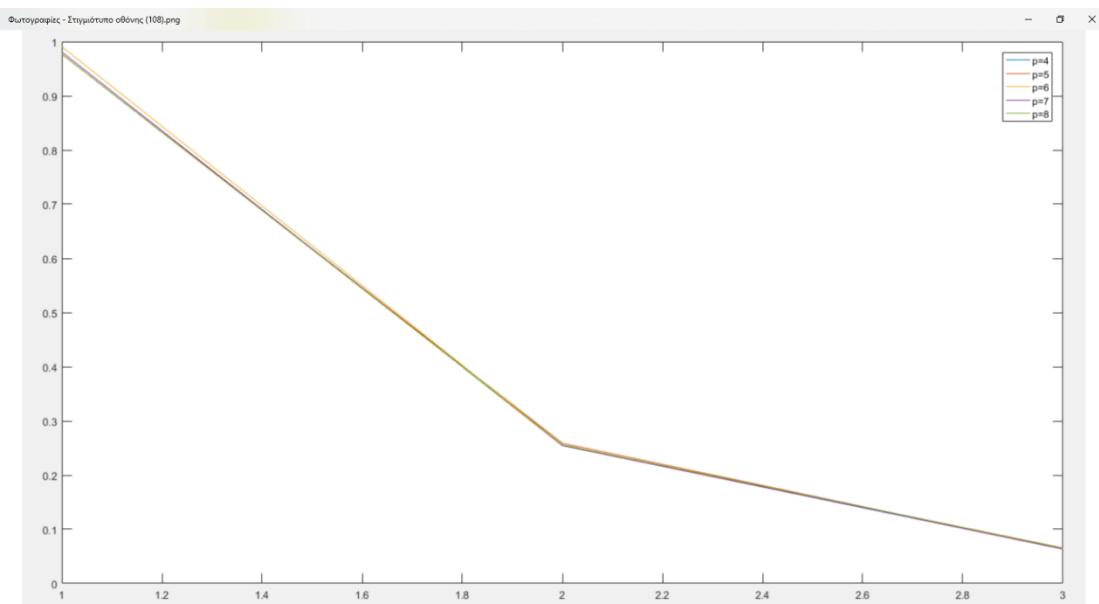




Εδώ βλέπουμε πως όσο αυξάνεται ο αριθμός των δυαδικών ψηφίων (αριθμός N) που χρησιμοποιούμε τόσο πιο κοντά είναι το ανακατασκευασμένο σήμα στο αρχικό. Το αναμέναμε καθώς όπως είδαμε από τα προηγούμενα ερωτήματα με την αύξηση του N μειώνεται το σφάλμα πρόβλεψης άρα και η ανακατασκευή του σήματος στον δέκτη θα είναι πιο κοντά στο αρχικό μου σήμα. Κι εδώ βλέπουμε πως για $p > 4$ δεν έχω κάποια ουσιαστική βελτίωση στο ανακατασκευασμένο σήμα.

Ακολουθεί ένα γράφημα που δείχνει την μέση τετραγωνική απόκλιση του αρχικού σήματος από την ανακατασκευή του συναρτήσει του αριθμού των δυαδικών ψηφίων N , για όλα τα p . Όπου τη μέση τετραγωνική απόκλιση την υπολογίζω με την εντολή

```
distor=sum((x-decodedx).^2)/length(x);
```



Βλέπουμε ότι με N=1bit η μέση τετραγωνική απόκλιση είναι κοντά στο 1 ενώ με N=3 bit η τιμή της είναι κοντά στο 0,05. Δηλαδή με αύξηση του N μόλις κατά 2 bit το ανακατασκευασμένο σήμα προσεγγίζει το αρχικό 20 φορές καλύτερα.

Ακολουθούν οι κώδικες για τις συναρτήσεις που υλοποιήθηκαν

ΜΕΡΟΣ Α

my_hyffmandict:

```
function [code] = my_huffmandict(signal,p)

[p,s]=sort(p);           %sortarw ton pinaka pithanotitwn p kai krataw to
index
for i=1:length(p)
    c{i}=[i];           %index gia kathe pithanotita
    code{i}='';          %codes gia diaforetikes pithanotites
end
while size(c,2)-1
    [p,i]=sort(p);
    c=c(i);
    for loop1 = [c{1}]      %vazw 0 sto elaxisto klado
        code{loop1} = ['0' code{loop1}];
    end
    for loop1 = [c{2}]      %vazw 1 sto 2o elaxisto klado
        code{loop1} = ['1' code{loop1}];
    end
    c{2} = [c{1} c{2}]; c(1) = []; %sigxwneusi
    p(2) = p(1) + p(2); p(1)=[]; %sigxwneusi
end
[~, s] = sort(s);       %epanaferw to arxiko sorting
code=code(s);           %epanaferw to sorting me vasi tin arxiki eisodo
%signal=signal(s);

for i=1:length(signal)
    code(2,i)={signal(i)};
end
code([1,2],:) = code([2,1],:);
code=code';

```

my_huffmanenco:

```
function enco = huffmanenco(sig, dict)

if (~iscell(sig) )
    [m,n] = size(sig);
    sig = mat2cell(sig, ones(1,m), ones(1,n) );
end
dictLength = size(dict,1);

idxCode = 1;
for i = 1 : length(sig)
```

```

% gia kathe timi, psaxnw diadoxika sto leksiko
% gia na vrw ton antistoixo kwdiko
tempcode = [];
for j = 1 : dictLength
    if( sig{i} == dict{j,1} )
        tempcode = dict{j,2};
        break;
    end
end

lenCode = length(tempcode);
enco(idxCode : idxCode+lenCode-1) = tempcode;
idxCode = idxCode + lenCode;
end

my_huffmandeco:

function deco = huffmandeco(comp, dict)

[m,n] = size(comp);
isSigNonNumeric = max(cellfun('isclass', {dict(:,1)}, 'char') );
deco = {};

i = 1;
while(i <= length(comp))
    tempcode = comp(i);
    found_code = is_a_valid_code(tempcode, dict);
    while(isempty(found_code) && i < length(comp))
        i = i+1;
        tempcode = [tempcode, comp(i)];
        found_code = is_a_valid_code(tempcode, dict);
    end
    if( i == length(comp) && isempty(found_code) )
        error(message('comm:huffmandeco:CodeNotFound'));
    end
    deco{end+1} = found_code;
    i=i+1;
end

if( n == 1 )           % ean i eisodos einai stili
    deco = deco';       % kanw kai tin eksodo dianisma stili
end
if ( ~isSigNonNumeric )
    decoMat = zeros(size(deco));
    decoMat = feval(class(dict{1,1}), decoMat); % douleuei se single
precision
    for i = 1 : length(decoMat)
        decoMat(i) = deco{i};
    end
    deco = decoMat;
end

%-----
% antistrofi anazitisi gia ena simvolo
% sigkrinw to code me ta stoixeia tou codebook kai epistrefw to
simvolo
% an vrethei
function found_code = is_a_valid_code(code, dict)

```

```

found_code = [];
m = size(dict);
for i=1:m(1)
    if ( isequal(code, dict{i,2}) )
        found_code = dict{i,1};
        return;
    end
end

```

ΜΕΡΟΣ Β

my_quantizer:

```

function [ y_out,centers ] = my_quantizer( y,N,min_value,max_value )

levels=2^N;
D=(2*max_value)/levels;
partition=[min_value:D:max_value];
centers=[min_value+D/2:D:max_value];
centers=[min_value centers max_value];
y_out=sum(partition<y)+1;
%centers;

end

```

my_dpcmenco:

```

function [ indx, quanterr ] = my_dpcmenco( N,sig, centers, partition,
predictor )

if nargin < 4
    error(message('comm:dpcmenco:NotEnoughInputs'));
end

if (length(centers)-1) ~= length(partition)
    error(message('comm:dpcmenco:InvalidPartitionSize'));
end;

if length(predictor) < 2
    error(message('comm:dpcmenco:InvalidPredictorSize'));
end
% To sistima DPCM ston pompo exei ti morfi:
%           e               quanterr
% Input signal -->+----Quantization-----|-----+
%                   ^_                         V
%                   |----->+
%           out |<---Predictor<-----| inp

len_predictor = length(predictor) - 1;
predictor = predictor(2:len_predictor+1);
predictor = predictor(:)';
len_sig = length(sig);

x = zeros(len_predictor, 1);
for i = 1 : len_sig;
    out = predictor * x;

```

```

e = sig(i) - out;
% index
indx(i) = my_quantizer(e,N,-3.5,3.5);
quanterr(i) = centers(indx(i));
inp = quanterr(i) + out;
% renew the estimated output
x = [inp; x(1:len_predictor-1)];
end;

end

my_dpcmdeco:

function [ sig,quanterr ] = my_dpcmdeco( indx, centers, predictor )

if nargin < 3
    error(message('comm:dpcmdeco:NotEnoughInputs'));
end;

% To sistima DPCM sto dekti exei ti morfi:
%           e               quanterr
%   INDX  -->-----Quantization-----+-----| -----
>sig
%                                     ^
%                                     v
%           out |-----Predictor-----| inp

len_predictor = length(predictor) - 1;
predictor = predictor(2:len_predictor+1);
predictor = predictor(:)';
len_sig = length(indx);

quanterr = indx;
quanterr = centers(indx);

x = zeros(len_predictor, 1);
for i = 1 : len_sig;
    out = predictor * x;
    sig(i) = quanterr(i) + out;
    % renew the estimated output
    x = [sig(i); x(1:len_predictor-1)];
end;

end

```