Ψηφιακές Τηλεπικοινωνίες

1η Εργαστηριακή Άσκηση 2018-2019

Ελευθέριος Μαντάς 1047128

ΜΕΡΟΣ Α

Κωδικοποίηση Huffman

Η κωδικοποίηση πηγής ασχολείται με την αποδοτική αναπαράσταση των δεδομένων που εξάγει μια πηγή πληροφορίας. Στη γενική περίπτωση αποδοτικές αναπαραστάσεις επιτυγχάνονται με Κώδικες Μεταβλητού Μήκους, ενώ ένα πολύ σημαντικό για εμάς ζήτημα είναι το πόσο μπορούμε να συμπιέσουμε μια πηγή, ερώτημα στο οποίο απαντάει το θεώρημα του Shannon.

Όσον αφορά την κωδικοποίση Huffman μπλοκ συμβόλων σταθερού μήκους από τη έξοδο της πηγής απεικονίζονται σε μεταβλητού μήκους μπλοκ δυαδικών συμβόλων.

Οι συχνότερα εμφανιζόμενες ακολουθίες εξόδου αντιστοιχούνται σε βραχύτερες δυαδικές ακολουθίες. Στην κωδικοποίηση μεταβλητού μήκους πρέπει να υπάρχει ένας και μοναδικός τρόπος για να διαχωρίζουμε τη λαμβανόμενη δυαδική ακολουθία σε κωδικές λέξεις.

Ο αλγόριθμος του κώδικα Huffman απατίζεται από τα εξής βήματα:

1. Διάταξη της εξόδου της πηγής κατά φθίνουσα σειρά πιθανοτήτων.
2. Συγχώνευση των δύο λιγότερο πιθανών εξόδων σε μία έξοδο, δηλαδή ένα σύμβολο και ανάθεση ως πιθανότητα της το άθροισμα των δύο πιθανοτήτων.
3. Αυθαίρετη ανάθεση των τιμών 0 και 1 ως κωδικές λέξεις για τις εξόδους που απομένουν. Σε κάθε βήμα του αλγορίθμου διατηρούμε αυτή την ανάθεση ίδια.
4. Ταξινόμηση και πάλι της εξόδου της πηγής κατά φθίνουσα σειρά πιθανοτήτων.
5. Επανάληψη των βημάτων 1-4 έως ότου συγχωνευτούν όλα τα σύμβολα.

Ερώτημα 1)

Υλοποιήθηκαν οι τρείς συναρτήσεις που ζητούνται με τη σειρά που δίνονται στην εκφώνηση.

huffmandict_func.m

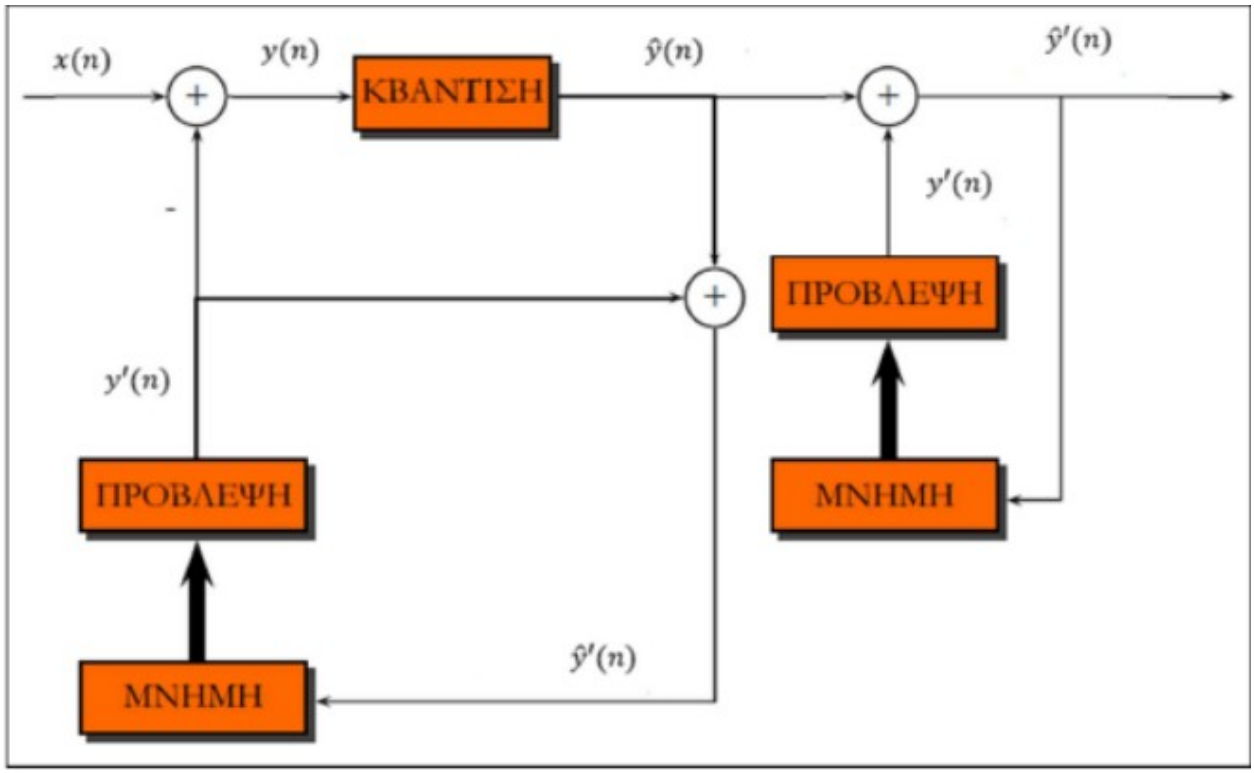
huffmanenco_func.m

huffmandeco_func.m

Ο κώδικάς τους παρατίθεται στο τέλος της αναφοράς.

ΜΕΡΟΣ Β

Κωδικοποίηση Διακριτής Πηγής με τη μέθοδο DPCM



Ερώτημα 1) Οι συναρτήσεις που ακολουθούν ,σε συνδυασμό με μια σύντομη περιγραφή της λειτουργίας τους, είναι αυτές που φτιάχτηκαν στο περιβάλλον της MATLAB για την υλοποίηση του συστήματος κωδικοποίησης/αποκωδικοποίησης DPCM .

Ακολουθήθηκε σειρά υλοποίησης ίδια με τη σειρά επεξήγησης των στοιχείων του συστήματος στην εκφώνηση:

encoder.m

Ο encoder (κωδικοποιητής) δέχεται το αρχικό σήμα εισόδου στην είσοδο του και το κβαντίζει. Κάθε φορά πριν γίνει η κβάντιση προστίθενται η πρόβλεψη από τον predictor (προβλέπτη). Η έξοδος, δηλαδή το κβαντισμένο σήμα οδηγείται σαν είσοδος στον decoder.

decoder.m

Ο decoder (αποκωδικοποιητής) δέχεται το κβαντισμένο σήμα από τον encoder (κωδικοποιητή) και αφού κάνει πρόβλεψη γι’ αυτόν επιστρέφει στην έξοδο την τιμή του δείγματος.

my_quantizer.m

Προκειται για τον κβαντιστή του συστήματος. Το σήμα κβαντίζεται με ομοιόμορφο κβαντιστή N bits με ακραίες τιμές min_value = -3.5 και max_value = 3.5.

calculation_of_a.m

Υπολογίζεται το μητρώο αυτοσυσχέτισης σύμφωνα με τους αριθμητικούς τύπους που βρίσκονται στην εκφώνηση της άσκησης.

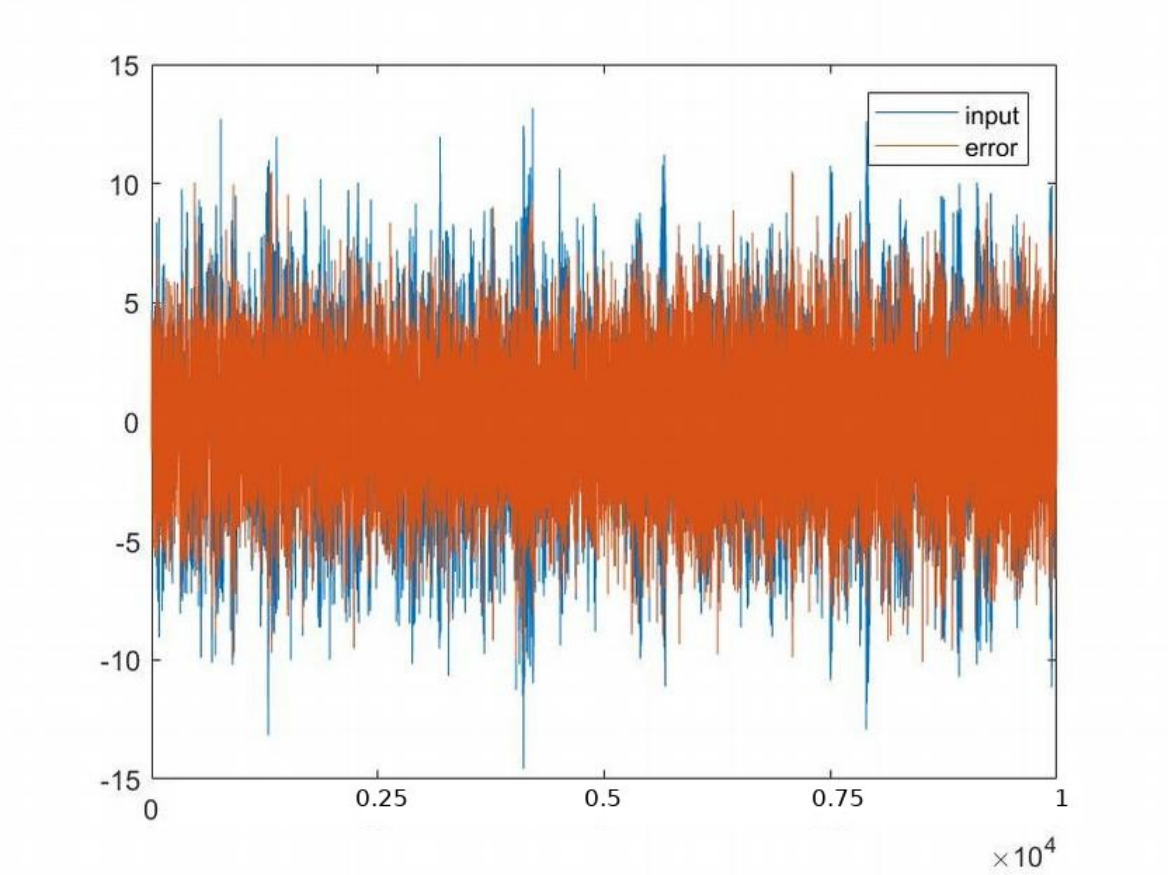
predictor.m

Υπολογίζει τις τιμές του a με χρήση των προηγούμενων τιμών που βρίσκονται ήδη στο buffer.

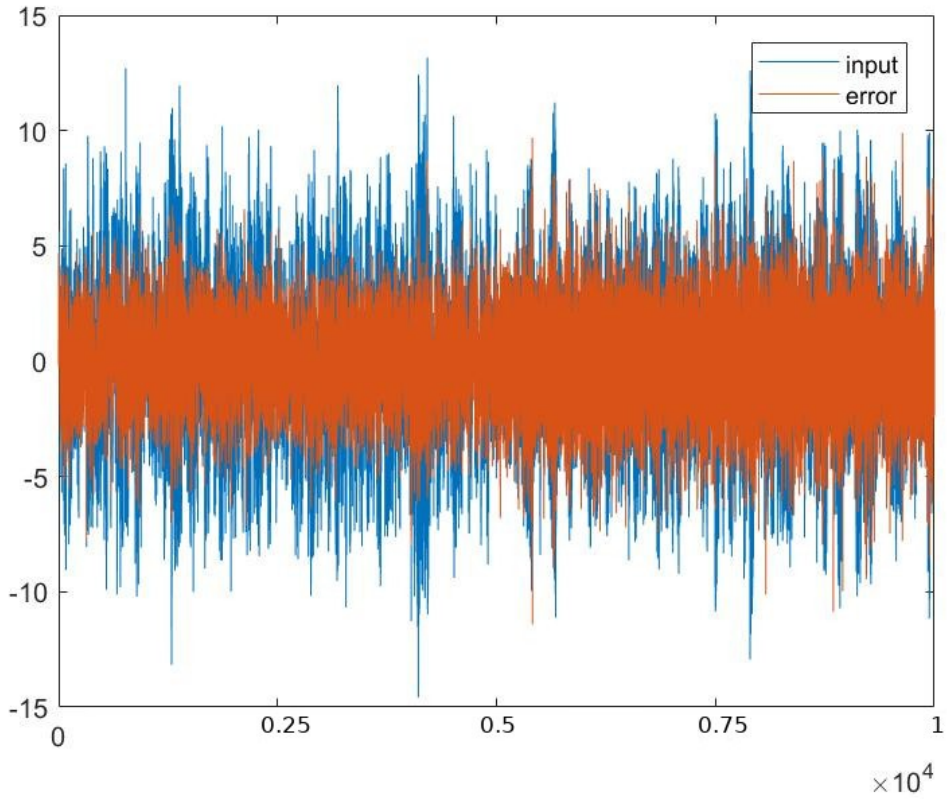
Ερώτημα 2)Για την υλοποίηση αυτού του ζητούμενου πήρα μετρήσεις για p=5 και p=8 για όλο το δείγμα.

Για δύο διαφορετικές τιμές του p παίρνουμε τα παρακάτω αποτελέσματα:

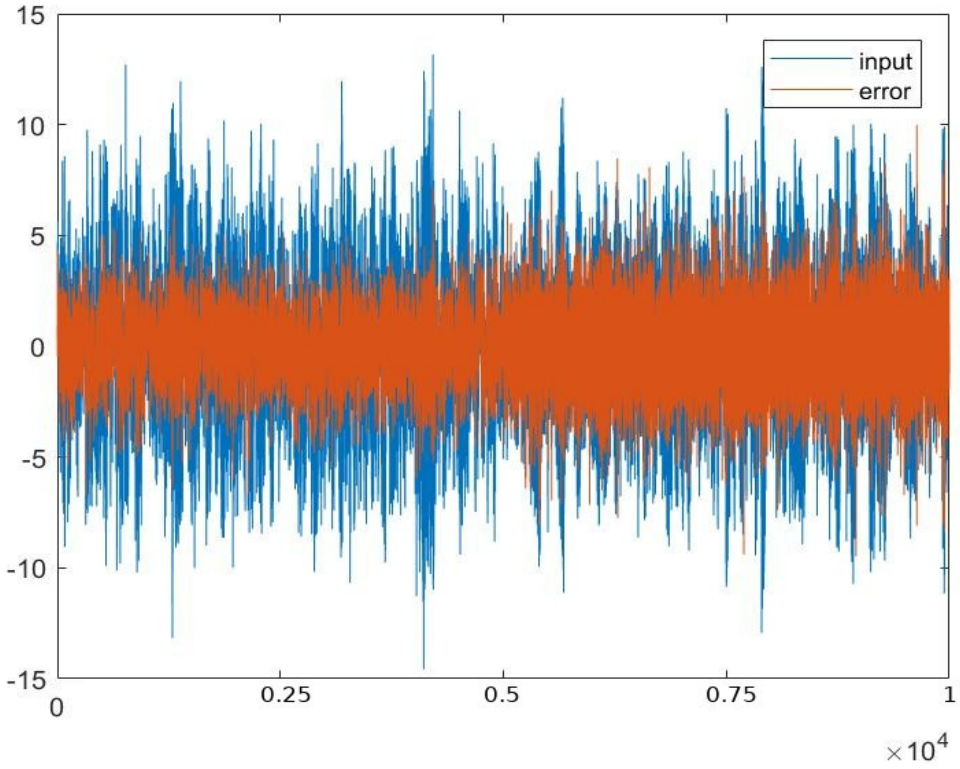
Δείγμα: 10.000 - p= 5 - N= 1



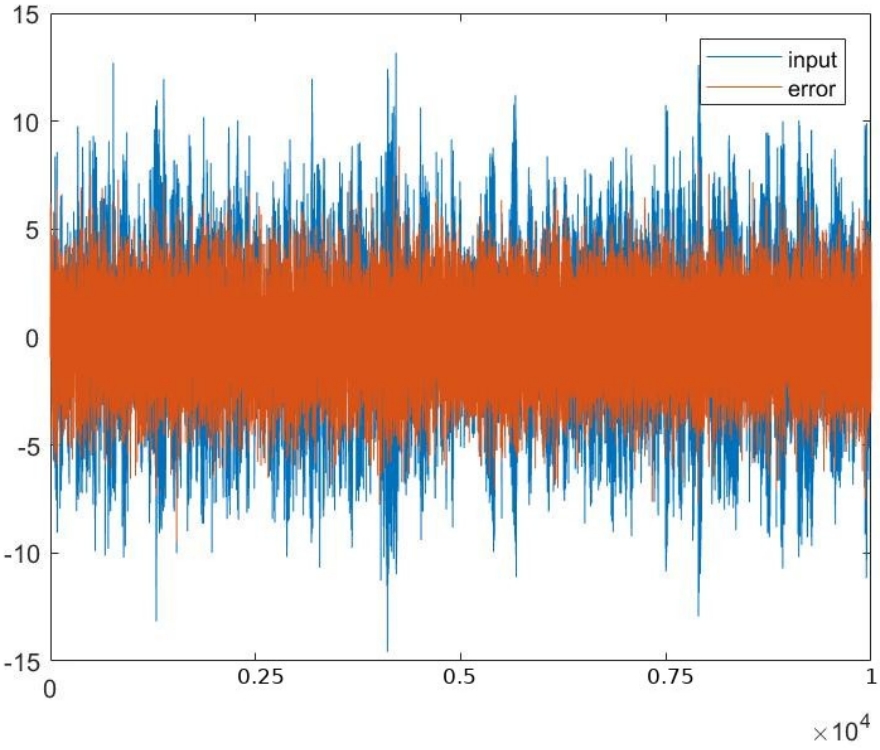
Δείγμα: 10.000 - p= 5 - N= 2



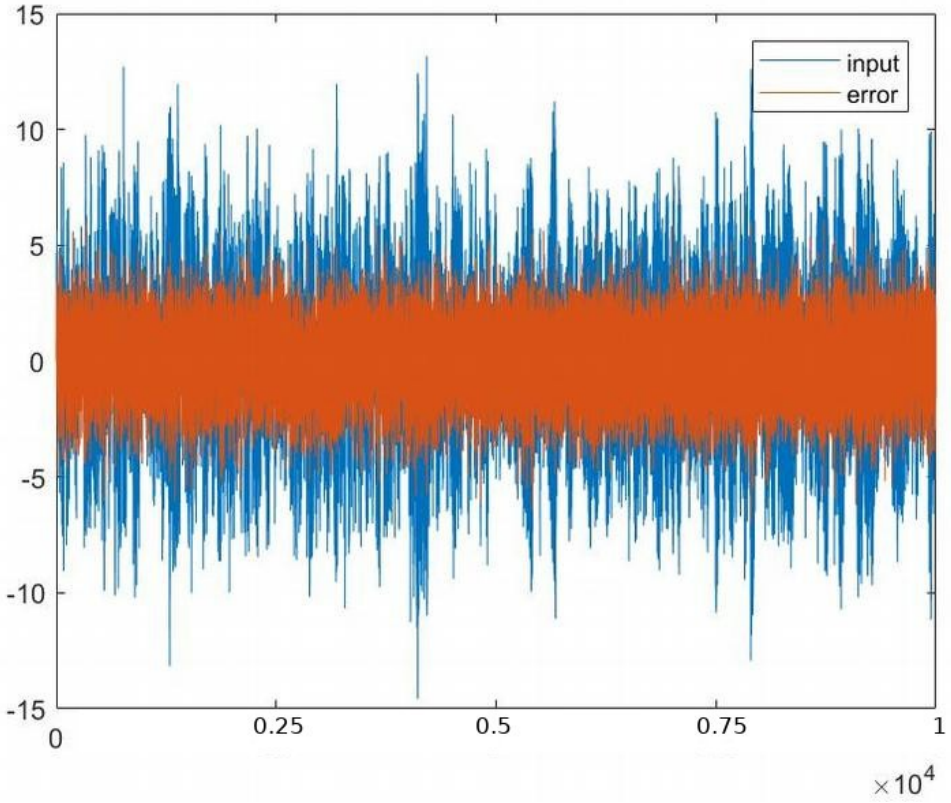
Δείγμα: 10.000 - p= 5 - N= 3



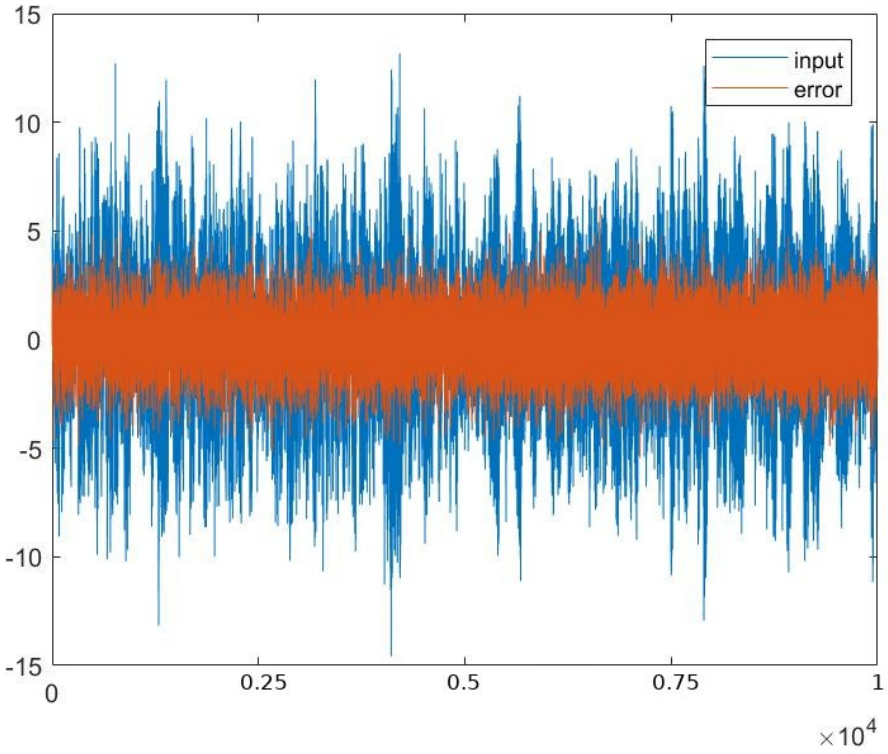
Δείγμα: 10.000 - p= 8 - N= 1



Δείγμα: 10.000 - p= 8 - N= 2



Δείγμα: 10.000 - p= 8 - N= 3

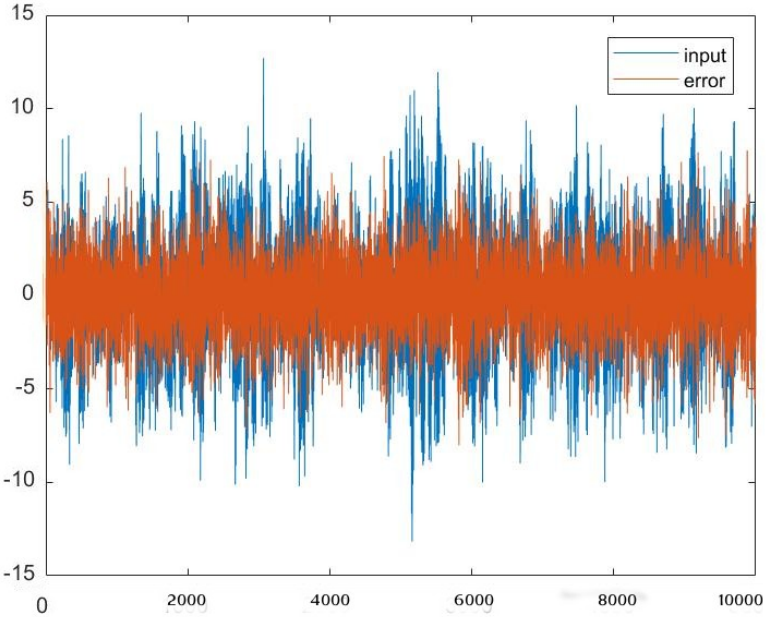


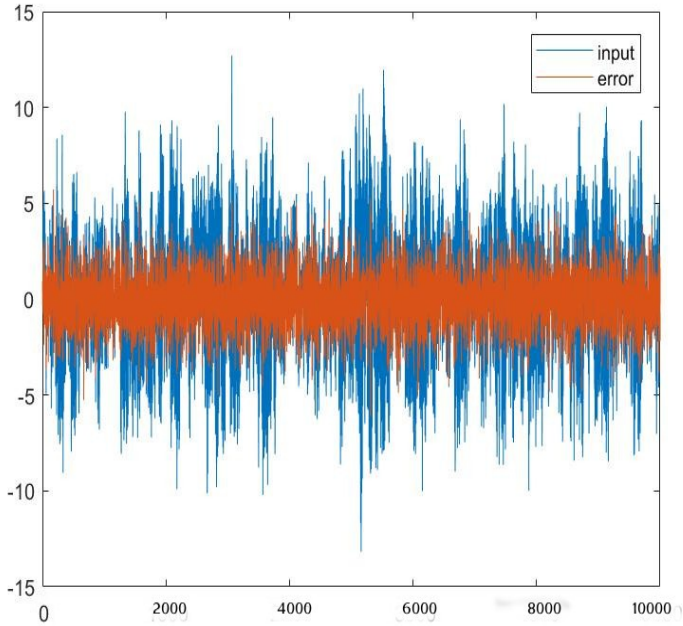
Από τα παραπάνω αποτελέσματα, κατά την αύξηση του N και του p, προκύπτει μείωση της περιοχής του σφάλματος πρόβλεψης.Όσο μεγαλύτερο είναι το N τόσο περισσότερες στάθμες - αντιρόσωποι της κβαντισμένης τιμής υπάρχουν. Έτσι, το κβαντισμένο σφάλμα να βρίσκεται πιο κοντά στην πραγματική του τιμή.Η τιμή του p καθορίζει τη μνήμη αποθήκευσης. Προφανώς όσο αυξάνεται αυτή, τόσο αυξάνεται η αξιοπιστεία της πρόβλεψης. Παρ ‘όλα αυτά, η αύξηση του p δεν βελτιώνει τόσο στην μείωση του σφάλματος όσο η αύξηση του N.

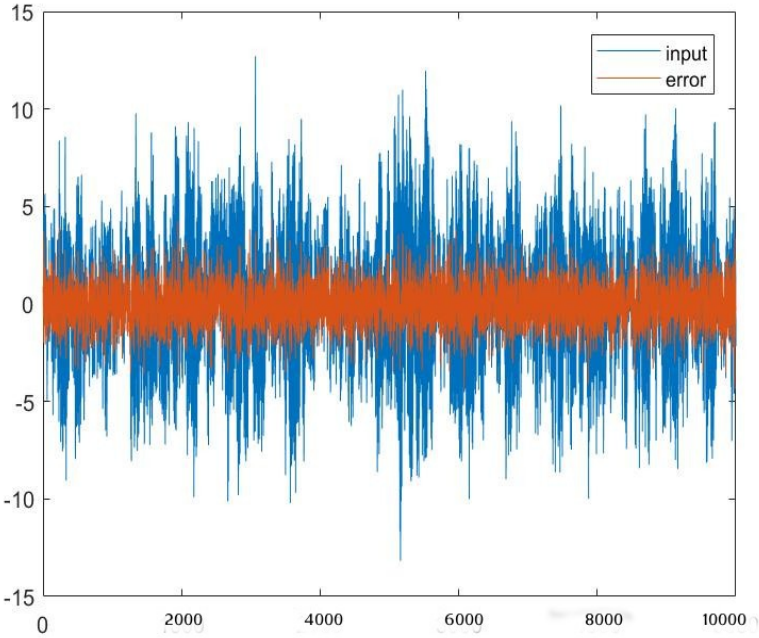
Το πλάτος του σφάλματος είναι αρκετά μικρότερο (υποδιπλάσιο περίπου) του αρχικού σήματος.

Ερώτημα 3)

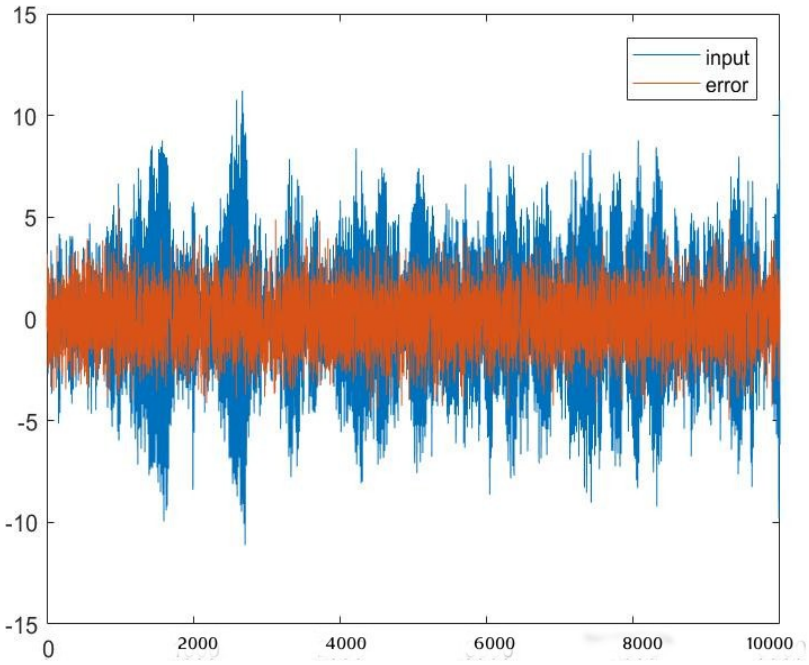
Δείγμα: p= 5 - N= 1 - N= 2 - N= 3

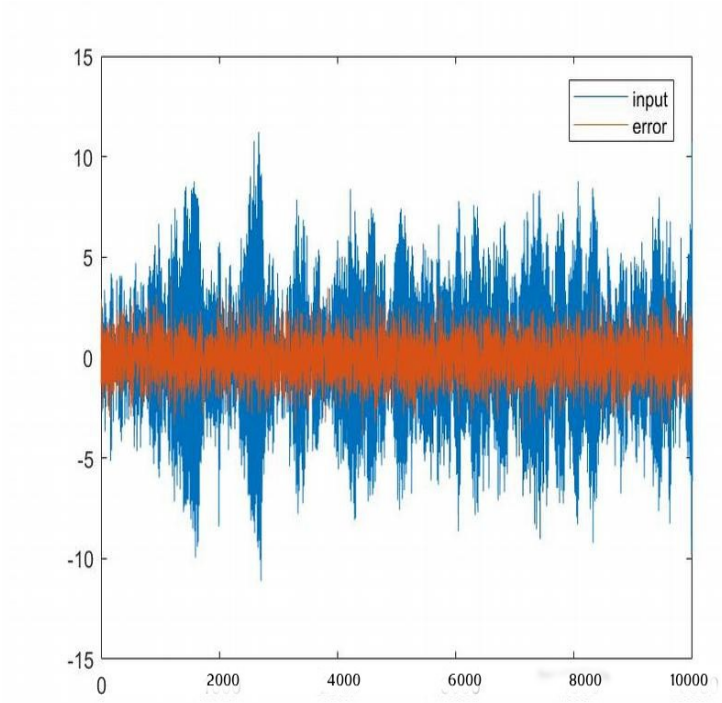

input
error
15
10
5
0
-5
-10
-15
0
2000
4000
6000
8000
10000

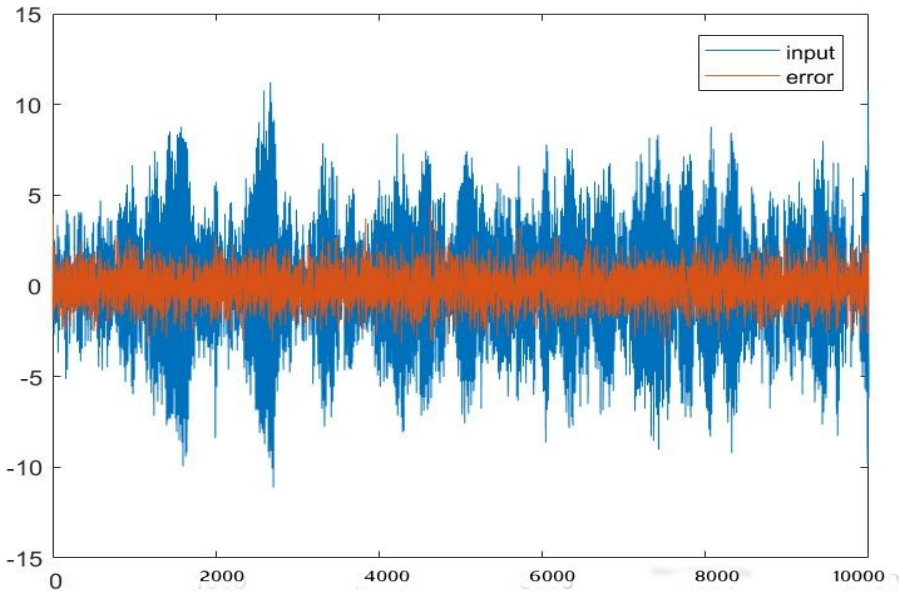

input
error
15
10
5
0
-5
-10
-15
0
2000
4000
6000
8000
10000


input
error
15
10
5
0
-5
-10
-15
0
2000
4000
6000
8000
10000

Δείγμα: p= 8 - N= 1 - N= 2 - N= 3



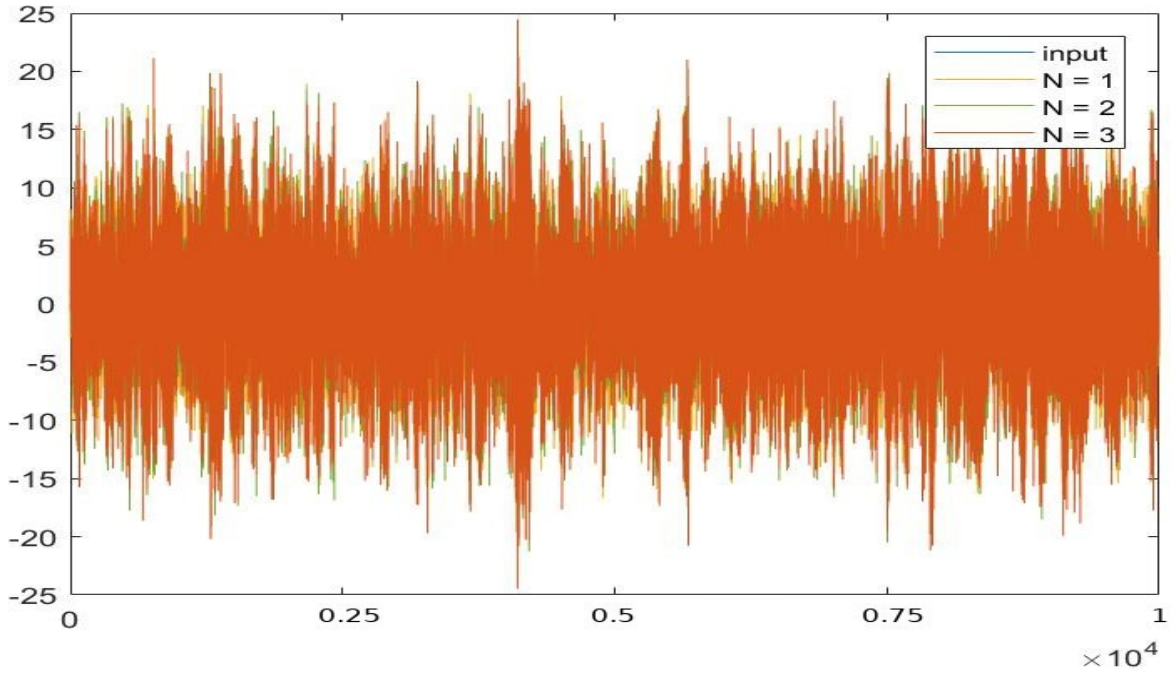

input
error
15
10
5
0
-5
-10
-15
0
2000
4000
6000
8000
10000

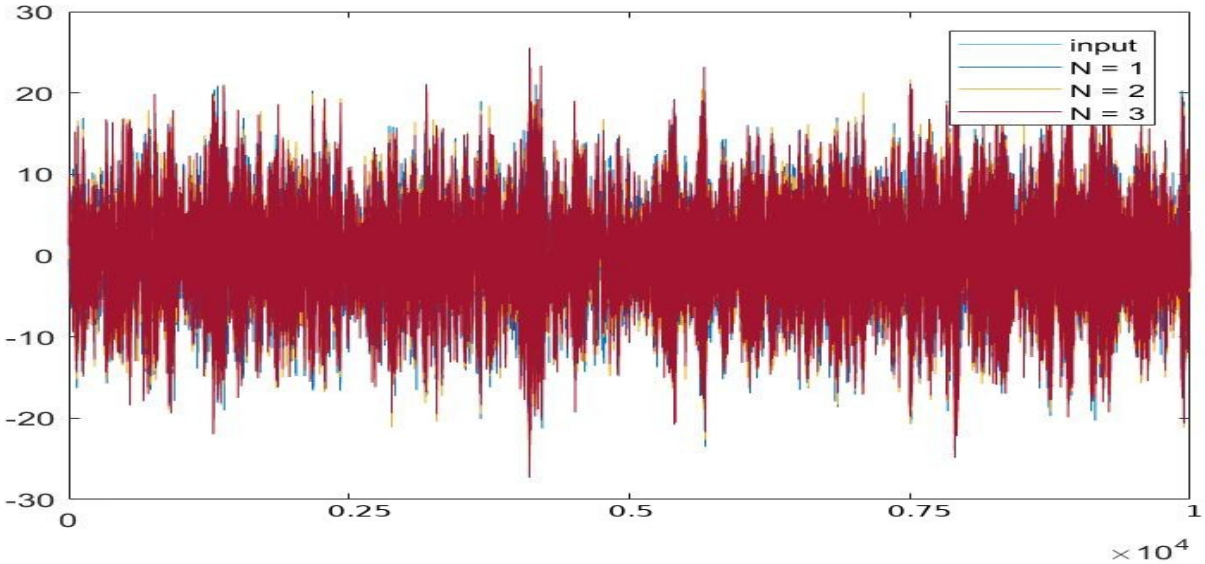

input
error
15
10
5
0
-5
-10
-15
0
2000
4000
6000
8000
10000

Στις μετρήσεις που πήραμε παρατηρούμε πως η σταδιακή αύξηση του N δίνει μικρότερο σφάλμα. Το ίδιο συμβαίνει με και με την αύξηση του p όμως σε αισθητά μικρότερη κλίμακα.

Ερώτημα 4)

Δείγμα: 10.000 - p= 4 - p= 8 - N= 1 - N= 2 - N= 3





Σύμφωνα με τα παραπάνω αποτελέσματα όσο μεγαλύτερ τιμές έχουν το N και το p, τόσο καλύτερα αναπαρίσταται το σήμα.

Συμπεραίνουμε πως το N παρουσιάζει καθοριστικό ρόλο για την ανακατασκευή του σήματος.

Στην συνέχεια παρουσιάζεται ο κώδικας του μέρους Β

main.m

```
function [y_kapelo_tonos_next, y_kapelo, y_kapelo_tonos, y, y_tonos, a] =
main(p, N)

load source.mat

for p = 4:1:8
        for N=1:3
                a = Calculation_of_a(x(0:10000),p, N);
        end
end

[y_kapelo,y_kapelo_tonos,y,y_predicted] = encoder(x,a,p,N);
 y_kapelo_tonos_next = decoder(y_kapelo,a,x,p);


end
```

predictor.m

```
function [ predicted_value ] = predictor(buffer,  a)


        predicted_value =  sum(a*.buffer);
end
```

Calculation_of_a.m

```
function [ vector_a ] = Calculation_of_a(x, p, N)
        for i = 1:p   %διατρεχουμε το πινακα αυτοσυσχετισης
           r(i) = 0;
           for n = (p + 1):length(x)
```

```
            r(i) = r(i) + x(n)*x(n-i);
        end
        r(i) = r(i) /(length(x) - p);
            for j=1:p
                R(i,j)=0;
                for n = (p+1):N
                    R(i,j) = R(i,j) + x(n-j) * x(n-i);
                end
            R(i,j) = R(i,j) / (N-p);
            end
    end
    a = inv(R)*r;
    for i=1:p
        vector_a(i) = my_quantizer(a(i),N,3.5,-3.5);
    end
end
```

decoder.m

```
function [ y_kapelo_tonos ] = decoder( y_kapelo, vector_a, x, p )

    y_predicted = zeros(length(x),1);
    y_kapelo_tonos = zeros(length(x),1);

        for i=1:p
            buffer(i)=x(i);
        end
```

```
        for i=1:10000
            y_predicted(i) = predictor(buffer,vector_a);
            y_kapelo_tonos(i) = y_kapelo(i) + y_predicted(i);

            buffer(2:end) = buffer(1:end-1);
            buffer(1) = y_kapelo_tonos(i);
        end
end
```

encoder.m

```
function [ y_kapelo, y_kapelo_tonos, y, y_predicted ] = encoder(x, a, p, N)

        y_kapelo = zeros(length(x),1);
        y_predicted = zeros(length(x),1);
        y_kapelo_tonos = zeros(length(x),1);
        y = zeros(length(x),1);
        buffer = zeros(p,1) ;

        %{
        αρχικοποιηση του buffer στις πρωτες p τιμες ωστε να ειναι
        γεματος οταν τον χρησιμοποιησουμε για πρωτη φορα
        %}

        for i=1:p
                buffer(i)=x(i);
        end

        for i=p+1:10000
```

```
            y_predicted(i) = predictor(buffer, a);     %υπολογισμος προβλεψης
            y(i) = x(i) - y_predicted(i);  %υπολογισμος του σηματος που θα εισελθει στον κβαντιστη
            y_kapelo(i) = my_quantizer(y(i),N,3.5,-3.5);   %υπολογισμος εξοδου κβαντιστη
            y_kapelo_tonos(i) = y_kapelo(i) + y_predicted(i);    %εισοδος στο buffer
            buffer(2:end) = buffer(1:end-1);
            buffer(1) = y_kapelo_tonos(i);
        end
end
```

my_quantizer.m

```
function [ output ] = my_quantizer( quant_input, N, min_value, max_value )

        S = max_value - min_value;
        levels = 2 ^ N;  %πληθος των επιπεδων
        eyros_perioxhs = S/levels;      %περιοχη κβαντισης
        centers = zeros(1,levels);  %αρχικοποιηση κεντρων περιοχων
        for i = 1:levels / 2
           centers(1,i) = (i - 1) * eyros_perioxhs + eyros_perioxhs/2 ;
        %υπολογισμος κεντρων
        end
        for j = levels/2+1:levels
           centers(1,j) = - ((j - 1) * eyros_perioxhs + eyros_perioxhs/2);
        end
        [~,I] = min(abs(centers - quant_input));
        output = centers(1,I);  %δεικτης στο διανυσμα centers
end
```

Στην συνέχεια παρουσιάζεται ο κώδικας του μέρους Α

huffman_dict.m

```
function dict = huffmandict_func( alphabet, pith, debug )
        alphabet = {'a' 'b' 'c' 'd' 'e'} % Διάνυσμα αλφαβήτου
    pith     = [0.4 0.3 0.05 0.05 0.25]  % Διάνυσμα Πιθανοτήτων
    dict     = huffmandict(alphabet,pith)


    symbol: {'a'  'b'  'c'  'd'  'e'}
    code: {'110'  '0'  '111'  '100'  '101'}
    if ( (min(pith) < 0) or (max(pith) >1) )
        error('πιθανότητες λάθος!')
    end
    if( length(alphabet) ~= length(pith) )
        error('Το αλφάβητο και οι πιθανότητες κάθε συμβόλου πρέπει να
είναι ίδια σε μέγεθος')
    end
    debug_ = 0; % global
    if ( nargin > 2 && debug )
        debug_ = 1;
        loop_ = 1;
        fileID = fopen(strcat(get_timestamp, '_huffmandict_func.txt'),'w');
% Create the log file.
        fprintf(fileID,'Debug Log - huffmandict \n');
    end
```

```
    for i = 1:length( pith )    % For each probability.
        codewords{i} = '';      % Create an empty codeword.
        symbol{i} = i;          % Index the codeword.
        if debug_
            word(i) = alphabet(i);  % Append it is a symbol.
        end
    end
%Main function
    while ( pith ~= 1 ) % Loop, until we reach the root.
        [~, arr] = sort(pith);  % Ταξινόμηση σε κάθε βήμα και παίρνω τη
διάταξη
        last = arr(1);
        next = arr(2);
        right_set = symbol{last};
        left_set  = symbol{next};
        right_probability = pith(last); %παίρνω τις πιθανότητες
        left_probability  = pith(next);
        merged_set = [right_set, left_set]; %τα βάζω σε μια καινούργια
δομή
        new_prob   = right_probability + left_probability;
        if debug_
            merged_word = strcat(word{last},word{next});
            fprintf(fileID,'Επανάληψη : %d\n',loop_);
            fprintf(fileID,'\tΣυγχονευμένα Σύμβολα {"%s","%s"}-->{"%s"}\
n',word{last},word{next},merged_word);
            fprintf(fileID,'\tΠιθανότητες Συμβόλων {%.4f,%.4f}-->{%.4f}\
n',right_probability,left_probability,new_pith);
            word(arr(1:2)) = '';
```

```
            word = [word merged_word];
            loop_ = loop_ + 1;
        end

        symbol(arr(1:2)) = '';
        pith(arr(1:2))   = '';
        symbol = [symbol merged_set];
        pith   = [pith new_pith];
        % Get the updated codeword.
        codewords = append_(codewords,right_set,'1');
        codewords = append_(codewords,left_set,'0');
    end

    dict.symbol = alphabet;
    dict.code = codewords;
    % Debug logging.
    if debug_
        fprintf(fileID,'Κώδικας Συμπίεσης: \n');
        for i = 1:length(dict.symbol)
            fprintf(fileID,'\t{"%s"}-->{"%s"}\n',dic
symbol{i},dict.code{i});
        end
        fclose(fileID);
    end
end
```

huffman_enco.m

```
function enco = huffmanenco_func( sig, dict, debug )
        alphabet = {'a' 'b' 'c' 'd' 'e'} % Διάνυσμα αλφαβήτου
    pith     = [0.4 0.3 0.05 0.05 0.25]  % Διάνυσμα Πιθανοτήτων
    dict     = huffmandict(alphabet,pith)

    symbol: {'a'  'b'  'c'  'd'  'e'}
    code: {'110'  '0'  '111'  '100'  '101'}

    sig = [ 'c'    'd'    'e'    'a'    'a'    'b'    'b' ]
    sig_encoded = huffmanenco_func(sig,dict)

    [m,n] = size(sig);
        %Main Function

    debug_ = 0; % global
    if ( nargin > 2 && debug == 1 )
        debug_ = 1;
        fileID = fopen(strcat(get_timestamp,
'_huffmanenco_func.txt'),'w'); % Ανοίγω το bebug file.
        fprintf(fileID,'Debug Log - huffmanenco_func \n');
        fprintf(fileID,'Input Signal: \n');
        for i = 1:length(sig)
            fprintf(fileID,'%s',sig(i));
        end
    end
```

```
    dictLength = length(dict.code);
    if debug_
        fprintf(fileID,'Κωδικοποίηση συμβόλων: \n');
    end

    enco = '';
    while( ~isempty(sig) ) % για καθε τιμη του signal
        tempcode = '';

        for j = 1 : dictLength
            if( strcmp(sig(1),dict.symbol{j}) ) % If there is a match.
                tempcode = dict.code{j};
                fprintf(fileID,'\tSymbol %s : %s\n',sig(1),dict.code{j}); %
Write encryption to the bebug file.
                break;
            end
        end

        if isempty(tempcode)
            error('Δεν μπορει να γίνει Κωδικοποίηση σε όλα τα σύμβολα. \
n');
        end

        enco = strcat(enco, tempcode); % Κρυπτογραφημενο μηνυμα
        sig = sig(2:end);
    end
```

```
    if debug_
        fprintf(fileID,'Compression Code :\n\t'); % γραφω τη Κωδικοποίηση στο αρχειο
        for i = 1:length(enco)
            fprintf(fileID,'%s',enco(i));
        end
        fclose(fileID); % κλεινω το φακελο
    end
end
```

huffman_deco.m

```
function deco = huffmandeco_func( sig, dict , debug )
    alphabet = {'a' 'b' 'c' 'd' 'e'} % Διάνυσμα αλφαβήτου
    pith     = [0.4 0.3 0.05 0.05 0.25]  % Διάνυσμα Πιθανοτήτων
    % Random text
    dict = huffmandict_func( alphabet, prob, 0 )
    x = alphabet(randsrc(1,7,[1:length(prob); prob]))
    encoded = huffmanenco_func(x,dict)
    decoded = huffmandeco_func(encoded,dict)

    dict = symbol: {'a'  'b'  'c'  'd'  'e'}
    code: {'110'  '0'  '111'  '100'  '101'}
    x = ['b'    'b'    'b'    'b'    'b'    'c'    'b']
    encoded = 000001110
    decoded =  ['b'    'b'    'b'    'b'    'b'    'c'    'b']
    % Main
```

```
    debug_ = 0; % global
    if ( nargin > 2 && debug == 1 )
        debug_ = 1;
        fileID = fopen(strcat(get_timestamp,
'_huffmandeco_func.txt'),'w'); % Open the bebug file.
        fprintf(fileID,'Debug Log - huffmanenco_func \n');
        fprintf(fileID,'Input Signal: \n');
        for i = 1:length(sig)
            fprintf(fileID,'%s',sig(i));
        end
    end


    if debug_
        fprintf(fileID,'Αποκωδικοπίηση συμβόλων\n');
    end


    deco = []; % Output signal vector initialize.
    sig_ = sig;
    codepos_ = 1;


    while( ~isempty(sig_) ) %Για καθε κομματι συμβολων
        temp_ = sig_(codepos_);
        dictb = dict; %ενα δευτερο λεξικο
        while (1) % Loop
            [flag,dict_] = found_match( temp_, codepos_, dictb); %Παιρνω
ενα μικροτερο λεξικο
```

```
        if ( flag == 0 ) % If there is an error at the encoded word.
            error('Προβλημα κατα την Αποκωδικοπίηση.\n');
        end

        dictb = dict_; % Update στο λεξικο
        if ( length(dictb.code) ~= 1 )
            codepos_ = codepos_ + 1;
            temp_ = sig_(codepos_);
        else    % Αν βρεθηκε το συμβολο
            if ( debug_ == 1 )
                fprintf(fileID,'Κωδικη λεξη {"%s"}. \n Βρεθηκε το συμβολο
"%s" μετα απο %d επαναληψεις.\n',
dictb.code{1},dictb.symbol{1},codepos_);
                %το γραφω στο αρχειο
            end

            codepos_ = 1;
            sig_ = sig_(length(dictb.code{1})+1:end);
            break;
        end
    end
    deco = [deco dictb.symbol]; % Append char to decoded signal.
end

if ( debug_ == 1 )
    fprintf(fileID,'Αποσυμπιεσμενο Σημα: \n');
    for i = 1:length(deco)
```

```
            fprintf(fileID,'%s',deco{i});
        end
        fclose(fileID);
    end
end

function [flag,dict_] = found_match( code, pos, dict )

    dict_.symbol={}; dict_.code={}; %δημιουργω λεξικα
    j = 1;
    flag = 0;
    for i = 1:length(dict.code)
        if ( strcmp(dict.code{i}(pos), code) ) % Αν ταιριαζε η εισοδος
            flag = 1; % No error match.
            dict_.symbol(j) = dict.symbol(i);
            dict_.code(j) = dict.code(i);
            j = j + 1;
        end
    end
end
```