Αναφορά 1ης Άσκησης
Ψηφιακές Τηλεπικοινωνίες

Ιάσων-Γεώργιος Παυλάκης 1059688

Ακαδημαϊκό έτος 2020-21
Χειμερινό Εξάμηνο

Ερώτημα 1ο - Κωδικοποίηση Huffman

Η κωδικοποίηση Huffman είναι μια αποτελεσματική μέθοδος συμπίεσης δεδομένων χωρίς απώλεια πληροφοριών. Παρέχει έναν αποτελεσματικό και ξεκάθαρο κώδικα αναλύοντας τις συχνότητες που εμφανίζονται ορισμένα σύμβολα σε ένα μήνυμα. Τα σύμβολα που εμφανίζονται πιο συχνά θα κωδικοποιούνται ως συμβολοσειρά λιγότερων bit, ενώ σύμβολα που δεν χρησιμοποιούνται τόσο πολύ θα κωδικοποιούνται με μεγαλύτερες συμβολοσειρές. Δεδομένου ότι οι συχνότητες των συμβόλων διαφέρουν μεταξύ των μηνυμάτων, δεν υπάρχει κανένας κωδικοποιητής Huffman που θα λειτουργεί για όλα τα μηνύματα. Αυτό σημαίνει ότι η κωδικοποίηση Huffman για την αποστολή ενός μηνύματος X μπορεί να διαφέρει από την κωδικοποίηση Huffman που χρησιμοποιείται για την αποστολή ενός άλλου μηνύματος Y. Η κωδικοποίηση Huffman λειτουργεί χρησιμοποιώντας δυαδικό δέντρο ταξινόμησης με βάση τη συχνότητα εμφάνισης για την κωδικοποίηση των συμβόλων.

1.

Τα functions βρίσκονται στο τέλος της αναφοράς με τα ονόματα huffman_dict, huffman_enco, huffman_deco αντίστοιχα.

2.

Το dictionary αυτού του ερωτήματος δημιουργήθηκε με βάση την εμφάνιση των γραμμάτων της αγγλικής αλφαβήτου στην καθημερινότητα. Με βάση τις ίδιες πιθανότητες παρήγαγε τυχαία σύμβολα η Πηγή Α. Αυτό όμως δεν ισχύει για την Πηγή Β, η οποία ειναι καθαρά προκατειλημμένη προς το γράμμα "k" το οποίο εμφανίζεται πολύ πιο συχνά απότι θα εμφανιζόταν στην καθημερινότητα (και κατά συνέπεια έχει αντιστοιχηθεί με περισσότερα bits στο dictionary). Για τον λόγο αυτό, μπορούμε να προβλέψουμε ότι η κωδικοποίηση των 10.000 συμβόλων της Πηγής Α θα έχει πολύ μικρότερο μέγεθος από αυτό της Πηγής Β. Όπως φαίνεται και από τα αποτελέσματα, η κωδικοποίηση της Πηγής Α έγινε με 42.124 bits, ενώ της Πηγής Β με 135.204 bits (σχεδόν 100.000 bits παραπάνω).

3.

Αυτή τη φορά το μήκος της κωδικοποίησης της Πηγής Β είναι 119.111 bits. Παρά το γεγονός ότι διπλασιάστηκαν τα σύμβολα του dictionary, τα bits κωδικοποίησης μειώθηκαν κατά περίπου 15.000. Αυτό εξηγείται επειδή το dictionary δημιουργήθηκε με βάση το περιεχόμενο του αρχείου, οπότε το γράμμα "k" που εμφανίζεται πολύ συχνά μέσα στο αρχείο, κωδικοποιείται με πολύ λιγότερα bits σε σχέση με το dictionary του προηγούμενου υποερωτήματος.

4.

Η εντροπία της Πηγής A στο ερώτημα 2 είναι $4,1815$, ενώ η δεύτερης τάξης επέκταση της Πηγής A είναι $8,3631$. Παρατηρούμε ότι είναι ακριβώς η διπλάσια, οπότε αντιλαμβανόμαστε ότι μεταφέρει πολλαπλάσια πληροφορία και ότι χρειάζεται περισσότερα bits για την κωδικοποίησή της. Συγκεκριμένα, η κωδικοποίηση 10.000 χαρακτήρων της Πηγής A στο ερώτημα 2 χρειάστηκε 42.124 bits, ενώ η δεύτερης τάξης επέκτασή της στο ερώτημα 4 χρειάστηκε $1.271.744$ bits για την κωδικοποίηση 5.000 ζευγών χαρακτήρων.

5.

Για την κωδικοποίηση της δεύτερης τάξης επέκτασης της Πηγής B με βάση το dictionary της δεύτερης τάξης επέκταση της Πηγής A χρειάστηκαν $7.773.573$ bits. Αντίστοιχα, για την ίδια κωδικοποίηση με βάση το dictionary που παράγεται από τη Πηγή B είναι $452.300.525$ bits. Αυτή η τεράστια διαφορά προκύπτει διότι το πρώτο dictionary περιέχει $676(= 26 \times 26)$ σύμβολα (δηλ. όλοι οι συνδιασμοί των μικρών ζεύγων γραμμάτων του αγγλικού αλφαβήτου) ενώ το δεύτερο περιέχει $2704(= 52 \times 52)$ σύμβολα (δηλ. όλοι οι συνδιασμοί μικρών και κεφαλαίων ζεύγων γραμμάτων του αγγλικού αλφαβήτου). Άρα, οι πιθανότητες κατανέμονται πιο αραιά στο δεύτερο, οπότε χρειάζονται πολλά bits ακόμα και για τα πιο συχνά εμφανιζόμενα ζεύγη.

Ερώτημα 2^o - Κωδικοποίηση PCM

Η Παλμοκωδική Διαμόρφωση (PCM) είναι μια μέθοδος που χρησιμοποιείται για τη μετατροπή ενός αναλογικού σήματος σε ψηφιακό σήμα έτσι ώστε ένα τροποποιημένο αναλογικό σήμα να μπορεί να μεταδοθεί μέσω του δικτύου ψηφιακής επικοινωνίας. Η διαδικασία ανάκτησης του αρχικού σήματος από το κβαντισμένο ονομάζεται αποδιαμόρφωση. Η διαδικασία διαμόρφωσης παλμού κώδικα γίνεται σε τρία βασικά στάδια: δειγματοληψία, κβαντισμός και κωδικοποίηση.

1.

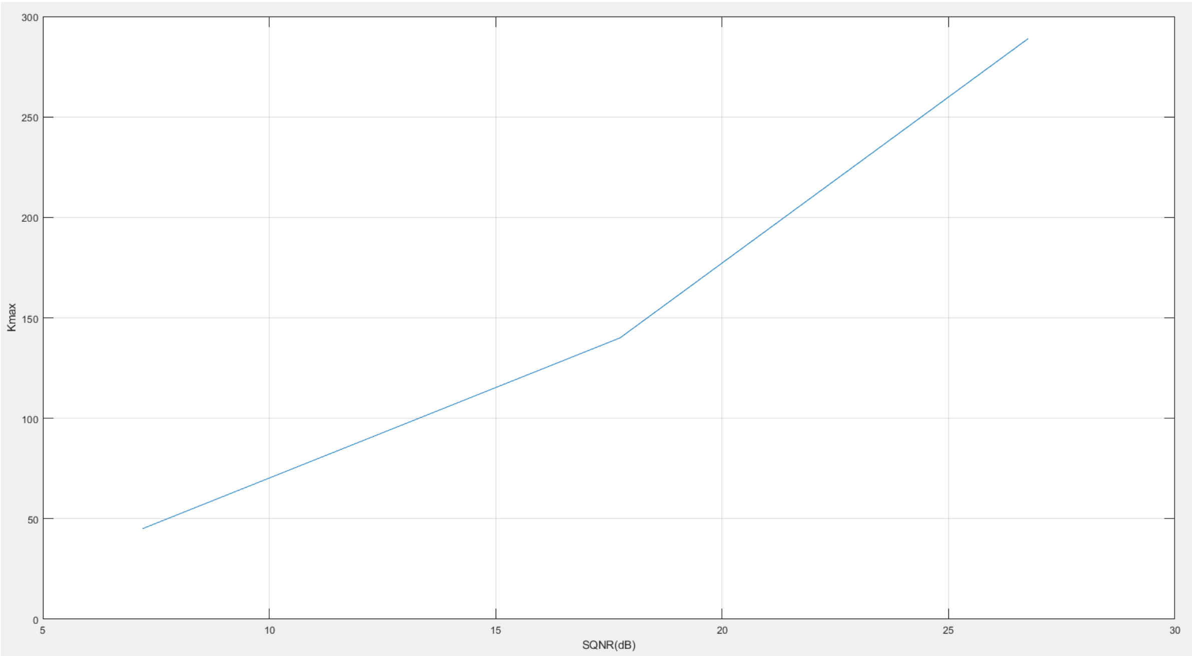
a . Τα αποτελέσματα για το θεωρητικό και το πρακτικό SQNR φαίνονται στον παρακάτω πίνακα:

	4-bit	6-bit
Θεωρητικό	$16,2893$ dB	$17,2125$ dB
Πρακτικό	$16,1258$ dB	$16,7111$ dB

b . Εκτελώντας τον κώδικα, η πιθανότητα η είσοδος του κβαντιστή να είναι εκτός της δυναμικής περιοχής είναι περίπου $1,8\%$

2.

a . Για $\epsilon = 2,2204 \times 10^{-16}$ η μεταβολή του SQNR σε σχέση με το K_{max} φαίνεται στο παρακάτω γράφημα:



b . Τα αποτελέσματα της σύγκρισης του SQNR από τον Lloyd-Max και τον ομοιόμορφο κβαντιστή φαίνονται στον παρακάτω πίνακα:

	Lloyd-Max	Uniform Quantizer
2-bit	$7,19457$ dB	$-2,90006$ dB
4-bit	$17,7441$ dB	$10,75553$ dB
6-bit	$26,7600$ dB	$23,70500$ dB

Όπως φαίνεται, το SQNR είναι πολύ καλύτερο χρησιμοποιώντας τον αλγόριθμο Lloyd-Max σε αντίθεση με αυτό του ομοιόμορφου κβαντιστή. Αυτό ισχύει διότι η πηγή του συγκεκριμένου υποερωτήματος είναι η ανθρώπινη φωνή, η οποία δεν ακολουθεί ομοιόμορφη κατανομή. Όμως, όσο αυξάνουμε τα κέντρα κβαντισμού στον ομοιόμορφο κβαντιστή, τόσο η απόδοσή του συγκλίνει με αυτή του Lloyd-Max.

c . Οι πιθανότητες εμφάνισης κάθε στάθμης και η εντροπία των επιπέδων κβάντισης ανάλογα το M και την μέθοδο κβαντισμού υπολογίζονται από τη συνάρτηση prob_2 (βρίσκεται στο τέλος της αναφοράς). Τα αποτελέσματα τυπώνονται με την εκτέλεση του κώδικα. Ενδεικτικά δίνεται ο πίνακας με την υπολογισμένη εντροπία.

Εντροπία επιπέδων

	Lloyd-Max	Uniform Quantizer
2-bit	$1,47709$	$1,00773$
4-bit	$3,04289$	$2,06386$
6-bit	$4,42351$	$3,77966$

d . Υπολογίζοντας το MSE για την μέθοδο Lloyd-Max παίρνω τα παρακάτω αποτελέσματα:

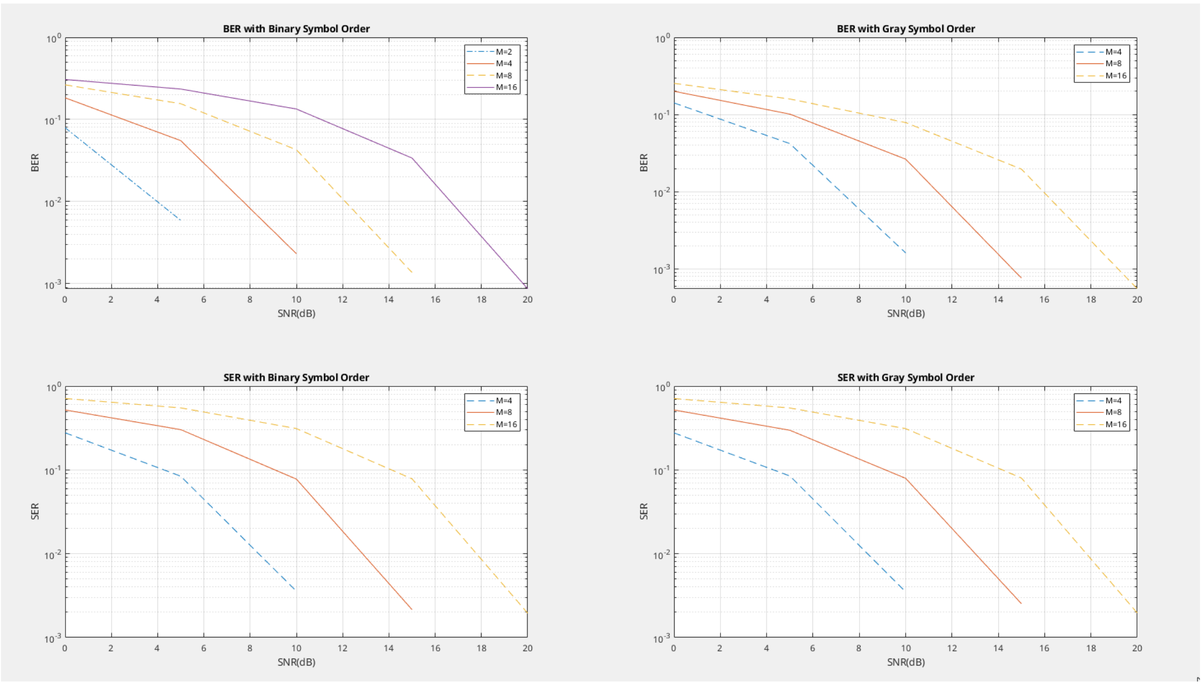
M	MSE
2-bit (45 iterations)	$3,5448 \times 10^{-3}$
4-bit (140 iterations)	$3,1234 \times 10^{-4}$
6-bit (289 iterations)	$3,9178 \times 10^{-5}$

Συνεπώς είναι κατανοητό ότι όσο αυξάνονται το M και οι επαναλήψεις, τόσο μειώνεται το MSE, δηλαδή αυξάνεται το SQNR και έτσι το σήμα εξόδου (δηλ. το κβαντισμένο σήμα) βελτιώνεται.

Ερώτημα 3ο - Μελέτη Απόδοσης Ομόδυνου Ζωνοπερατού Συστήματος M-PAM

Η Διαμόρφωση Πλάτους Παλμού (Pulse Amplitude Modulation) είναι μια τεχνική στην οποία το πλάτος κάθε παλμού ελέγχεται από το στιγμιαίο πλάτος του σήματος διαμόρφωσης. Πρόκειται για ένα σύστημα διαμόρφωσης στο οποίο το σήμα λαμβάνεται σε τακτά χρονικά διαστήματα και κάθε δείγμα γίνεται ανάλογο με το πλάτος του σήματος τη στιγμή της δειγματοληψίας (ανάλογα την τάξη του M-PAM υπάρχει και διαφορετικός αριθμός από πιθανά πλάτη για την δειγματοληπτημένη τιμή). Με αυτή τη τεχνική τα δεδομένα μεταδίδονται κωδικοποιώντας το πλάτος μιας σειράς παλμών σήματος.

Με βάση τα ζητούμενα, το γράφημα με τις απαραίτητες πληροφορίες φαίνεται παρακάτω:



Κώδικας MATLAB

Σημείωση: Ο χαρακτήρας ¨∼¨ δεν εμφανίζεται στο text του κώδικα λόγω της LaTeX.

Ερώτημα 1ο - Κωδικοποίηση Huffman

```
%================== STEPS FOR 1.2)
letters = double('etaoinsrhdlucmfywgpbvkxqjz');
probs = [12.02, 9.1, 8.12, 7.68, 7.31, 6.95, 6.28, 6.02, 5.92,
    4.32, 3.98, 2.88, 2.71, 2.61, 2.3, 2.11, 2.1, 2.03, 1.82,
    1.49, 1.11, 0.69, 0.17, 0.11, 0.1, 0.07];
probs = probs/100;
x = [letters ; probs];
source_A = randsrc(10000,1,x);
dict_1 = huffman_dict(x(1,1:end),x(2,1:end));

fid = fopen('kwords.txt');
source_B = textscan(fid,'%s');
source_B = source_B{1,1}(:,1);
fclose(fid);

enco1_A = huffman_enco(source_A, dict_1);
deco1_A = huffman_deco(enco1_A, dict_1);
enco1_B = huffman_enco(source_B, dict_1);
deco1_B = huffman_deco(enco1_B, dict_1);


%================== STEPS FOR 1.3)
fileId = fopen('kwords.txt', 'r');
file_content = fscanf(fileId,'%c');
fclose(fileId);
alphabet = 'A':'z';
d_alphabet = double(alphabet);
d_alphabet = d_alphabet';
d_alphabet(27:32) = [];
alphabet = char(d_alphabet);

cell_arr = cell(52,2);

for i = 1:length(alphabet)
    cell_arr{i,1} = alphabet(i);
    cell_arr{i,2} = 0;
end

total_characters = 0;

for i = 1:length(file_content)
```

```
    for j = 1:length(alphabet)
        if file_content(i) == cell_arr{j,1}
            cell_arr{j,2} = cell_arr{j,2} + 1;
            total_characters = total_characters + 1;
        end

    end
end

for i = 1:length(alphabet)
    cell_arr{i,2} = cell_arr{i,2} / total_characters;
end

d_alphabet = d_alphabet';
dict_2 = huffman_dict(cell_arr(:,1), cell2mat(cell_arr(:,2)));

enco2_B = huffman_enco(source_B, dict_2);
deco2_B = huffman_deco(enco2_B, dict_2);

%================= STEPS FOR 1.4)
letters = double('etaoinsrhdlucmfywgpbvkxqjz');
probs = [12.02, 9.1, 8.12, 7.68, 7.31, 6.95, 6.28, 6.02, 5.92,
   4.32, 3.98, 2.88, 2.71, 2.61, 2.3, 2.11, 2.1, 2.03, 1.82,
   1.49, 1.11, 0.69, 0.17, 0.11, 0.1, 0.07];
probs = probs/100;
x = [letters ; probs];
letters_2 = cell(2,676);
index = 1;
for i = 1:length(letters)
    char_1  = letters(i);
    prob_1 = probs(i);
    for j = 1:length(letters)
        char_2 = letters(j);
        prob_2 = probs(j);
        letters_2{1,index} = strcat(char_1, char_2);
        letters_2{2,index} = prob_1 * prob_2;
        index = index + 1;
    end
end

dict_3 = huffman_dict(letters_2(1,:), cell2mat(letters_2(2,:)));
source_A = randsrc(5000,2,x);
source_A = char(source_A);
enco3_A = huffman_enco(source_A, dict_3);

entropy = 0;
for i=1:length(dict_3)
    temp = letters_2{2,i};
    entropy = entropy + temp * log2(temp);
```

```
end
entropy = -entropy;

%================= STEPS FOR 1.5)
enco3_B = huffman_enco(source_B, dict_3);
idx = 1;
for i='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
  for j='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
    arr{idx,1} = append(i,j);
    arr{idx,2} = count(file,append(i,j)) ;
    idx = idx + 1;
  end
end
idx = 1;
sumOfLetters = sum(cell2mat(arr(:,2)));
for i=1:length(cell2mat(arr(:,1)))
  arr{i,2} = (cell2mat(arr(i,2)) / sumOfLetters);
end
dict_4 = huffman_dict(arr(:,1),cell2mat(arr(:,2)));
enco4_B = huffman_enco(source_B, dict_4);
```

```
function [dict] = huffman_dict(symbols,prob)

    if  iscell(symbols)
        symbols = num2cell(symbols);
    end

    huff_tree = struct('signal', [], 'probability', [], 'child',
     [], 'code', [], 'origOrder', -1);

    for i = 1:length( symbols )
        huff_tree(i).signal = symbols{i};
        huff_tree(i).probability = prob(i);
        huff_tree(i).origOrder = i;
    end

     [ , i] = sort(prob);
    huff_tree = huff_tree(i);

    huff_tree = create_huff_tree(huff_tree);

     [ ,dict] = create_huff_dict(huff_tree,{},0);

     [ , dictsortorder] = sort([dict{:,4}]);
    lenDict = length(dictsortorder);
    finaldict = cell(lenDict, 2);
    for i=1:length(dictsortorder)
        finaldict{i,1} = dict{dictsortorder(i), 1};
        finaldict{i,2} = dict{dictsortorder(i), 2};
```

```
    end

    for i=1:length(finaldict)
        finaldict{i,1} = char(finaldict{i,1});
    end
dict = finaldict;

% ==== CREATE_HUFF_TREE
function huff_tree = create_huff_tree(huff_tree)

    numRemNodes = length(huff_tree);
    if( numRemNodes <= 1)
        return;
    end

    numNodesToComb = rem(length(huff_tree)-1, 1) + 1;
    if numNodesToComb == 1
       numNodesToComb = 2;
    end

    temp = struct('signal', [], 'probability', 0, 'child', [], 'code', []);

    for i = 1:numNodesToComb
        if isempty(huff_tree), break; end
        temp.probability = temp.probability + huff_tree(1).probability; % for ascending order
        temp.child{i} = huff_tree(1);
        temp.origOrder = -1;
        huff_tree(1) = [];
    end

    huff_tree = insertNode(huff_tree, temp);

    huff_tree = create_huff_tree(huff_tree);


% ==== INSERT_NODE
function huff_tree = insertNode(huff_tree, newNode)
    i = 1;
    while i <= length(huff_tree) && newNode.probability > huff_tree(i).probability
        i = i+1;
    end

    huff_tree = [huff_tree(1:i-1) newNode huff_tree(i:end)];

% ===== CREATE_HUFF_DICT
function [huff_tree,dict,total_wted_len] = create_huff_dict(
```

```
    huff_tree,dict,total_wted_len)
    if isempty(huff_tree.child)
        dict{end+1,1} = huff_tree.signal;
        dict{end, 2} = huff_tree.code;
        dict{end, 3} = length(huff_tree.code);
        dict{end, 4} = huff_tree.origOrder;
        total_wted_len = total_wted_len + length(huff_tree.code)
    *huff_tree.probability;
        return;
    end
    num_childrens = length(huff_tree.child);
    for i = 1:num_childrens
        huff_tree.child{i}.code = [huff_tree(end).code, (
    num_childrens-i)];
        [huff_tree.child{i}, dict, total_wted_len] =
    create_huff_dict(huff_tree.child{i}, dict, total_wted_len);
    end
```

```
function [ vector ]  = huffman_enco(src, dictionary)
  if iscell(src)
    input = [src{:}];
  else input = src;
  end
  dict1 = cell2mat(dictionary(:,1));
  dict2 = dictionary(:,2);
  dict2 = cellfun(@num2str,dict2,'UniformOutput',false);
  input = double(input);
  idx = 1;
  for i = 1:length(input)
    for j = 1:length(dict1)
      if input(i) == double(dict1(j))
        encoding = cell2mat( dict2(j) );
        encoding = str2num(encoding);
        for k=1:length(encoding)
          vector(idx) = encoding(k);
          idx = idx+1;
        end
      end
    end
  end
end
```

```
function [ vector ] = huffman_deco(input,dictionary)
  dict1 = cell2mat(dictionary(:,1));
  dict2 = dictionary(:,2);
  dict2 = cellfun(@num2str,dict2,'UniformOutput',false);
  idx = 1;
  k = 1;

```

```
  for i=1:length(input)
    temp(k) = input(i);
    check = num2str(temp);
    for j=1:length(dict1)
      code = dict2{j}(:)';
      flag = strcmp(code,check);
      if flag == 1
       vector(idx) = dict1(j);
       idx = 1 + idx;
       temp = [];
       k=0;
      end
    end
    k = k + 1;
  end
end
```

Ερώτημα 2° - Κωδικοποίηση PCM

- **1.**

```
function data = source_A(M)

temp = (randn(M,1) + j*randn(M,1)) / sqrt(2);
data = abs(temp) .^ 2;
```

```
function [xq, centers] = my_quantizer(x, N, min_value,
    max_value)
d = max_value / 2^N;
level = min_value;

if min(x) < min_value
    for i=1:size(x)
        if x(i) < min_value
            x(i) = min_value;
        end
    end
end
if max(x) > max_value
    for i=1:size(x)
        if x(i) > max_value
            x(i) = max_value;
        end
    end
end

step = (max_value - min_value) / 2^N;
centers(1) = max_value - step/2;
for i=2:2^N
    level = level + d;
```

```
    centers(i) = centers(i-1) - step;
end


xq = [];
for i=1:size(x)
    [distance index] = min(abs(centers - x(i)));
    xq(i) = index;
end

xq = xq';
```

```
function SQNR_1()

x = source_A(10000);
[xq, centers] = my_quantizer(x, 4, 0, 4);

new = centers(xq);
new = new';

% calculation of SQNR
S = mean(x.^2);
N = mean((new-x).^2);

R = 10 * log10(S/N);

%===========================================

x = source_A(10000);
[xq, centers] = my_quantizer(x, 6, 0, 4);

new = centers(xq);
new = new';

% calculation of SQNR .
S = mean(x.^2);
N = mean((new-x).^2);

R = 10 * log10(S/N);
```

```
function prob_1()

x = source_A(10000);

counter = 0;
for i=1:length(x)
    if x(i) < 0 || x(i) > 4
        counter = counter + 1;
    end
```

```
end

prob = (counter / length(x)) * 100
```

- **2.**

```
function data = source_B(min_value, max_value)

[y,  ] = audioread('speech.wav');

if min(y) < min_value || max(y) > max_value
    y = normc(y); % or normr
    % scale to [x, y]
    range = max_value - min_value;
    new = (new * range) + min_value;
    y = new
end

data = y;
```

```
function [xq, centers, D, iterations] = lloyd_max(x, N,
    min_value, max_value)

s = size(x);
s = s(1);

[ , centers] = my_quantizer(x, N, min_value, max_value);

if min(x) < min_value
    for i=1:s
        if x(i) < min_value
            x(i) = min_value;
        end
    end
end
if max(x) > max_value
    for i=1:s
        if x(i) > max_value
            x(i) = max_value;
        end
    end
end

centers = flip(centers);
centers = [min_value centers max_value];

% D is the Distortion log.
D = [0 1];

```

```
% Main algorithm loop.
k = 2;
while abs(D(k) - D(k-1)) >= eps
    xq = [];
    total = 0;

    counted   = zeros(length(centers));
    cond_mean = zeros(length(centers));

    % calculating the quantization zones.
    T = [];
    T(1) = min_value;
    for i=2:(length(centers)-2)
        T(i) = (centers(i) + centers(i+1))/2;
    end
    T(i+1) = max_value;


    for i=1:s
        % iterating over the zones to find the correct one.
        for j=1:(length(T)-1)
            if T(j) < x(i) && x(i) <= T(j+1)

                xq(i) = j;

                total = total + abs(centers(j+1) - x(i));

                cond_mean(j) = cond_mean(j) + x(i);
                counted(j)   = counted(j) + 1;
            end
        end

        if x(i) == T(1)
            xq(i) = 1;
            total = total + abs(centers(2) - x(i));
            cond_mean(1) = cond_mean(1) + x(i);
            counted(1)   = counted(1) + 1;
        end
    end
    avg_distortion = total/s;

    D = [D avg_distortion];
    k = k + 1;

    % finding the new centers, since that's what we need to get the zones.
    for j=2:(length(centers)-1)
        if counted(j-1)  = 0
            centers(j) = cond_mean(j-1)/counted(j-1);
```

```
        end
    end
end

% erasing the required-until-now extra values in the Distortion log vector.
% now D(i) is the mean distortion of the i-th iteration.
D(1) = [];
D(2) = [];


centers(1) = [];
centers(length(centers)) = [];

xq = xq';

iterations = k-2;
```

```
function SQNR_2()

for n=2:2:6
    x = source_B(-1, 1);
    [xq, centers, , iter] = lloyd_max(x, n, -1, 1);
    new = centers(xq);
    new = new';
    S = mean(x.^2);
    N = mean((new-x).^2);

    R_lloyd(n/2) = 10 * log10(S/N);

    squaredError = (double(x)- double(new)).^2;
    MSE(n/2) = sum(squaredError) / numel(squaredError);

    iterations(n/2) = iter;
end


%====================================================

for n=2:2:6
    x = source_B(-1, 1);
    [xq, centers] = my_quantizer(x, n, -1, 1);
    new = centers(xq);
    new = new';
    S = mean(x.^2);
    N = mean((new-x).^2);

    R_uni(n/2) = 10 * log10(S/N);

```

```
end
```

```
function prob_2()

x = source_B(-1, 1);

% Uniform
for n=2:2:6
    [xq,  ] = my_quantizer(x, n, -1, 1);
    probs = tabulate(xq);
    fprintf('Probabilities (Uniform Quantizer) with %d-bit are: [second column is percentage]\n', n);
    prob = probs(:,3);
    [probs(:,1) prob]
    prob = prob / 100;

    temp = [];
    for i=1:length(prob)
        if prob(i)>0
            temp(i) = prob(i) * log2(prob(i));
        end
    end
    entropy_uni(n/2) = -sum(temp);
end

% Lloyd-Max
for n=2:2:6
    [xq,  ,  ,  ] = lloyd_max(x, n, -1, 1);
    probs = tabulate(xq);
    fprintf('Probabilities (Lloyd-max) with %d-bit are: [second column is percentage]\n', n);
    prob = probs(:,3);
    [probs(:,1) prob]
    prob = prob / 100;

    temp = [];
    for i=1:length(prob)
        if prob(i)>0
            temp(i) = prob(i) * log2(prob(i));
        end
    end
    entropy_lloyd(n/2) = -sum(temp);
end
```

Ερώτημα 3° - Μελέτη Απόδοσης Ομόδυνου Ζωνοπερατού Συστήματος M-PAM

```
% 1. BER for 'bin'
M = [2 4 8 16];
SNR = 0:5:40;
symbol_enc = 'bin';
BER = [];

for i=1:length(M)
    input = 0:M(i)-1;
    X = randi([0 M(i)-1], 1, 100000);
    mod = pam_mod(input, M(i), symbol_enc);
    scale = modnorm(mod, 'avpow', 1);
    Eb = 1/log2(M(i));

    for j=1:length(SNR)
        div = Eb/2 * 10^(-SNR(j)/10);
        noise = sqrt(div) * randn(1, length(X));
        normalized_signal = scale * pam_mod(X, M(i), symbol_enc)
    ;
        noise_signal = normalized_signal + noise;

        noise_signal = noise_signal/scale;
        normalized_signal = normalized_signal/scale;
        noise_demod = pam_demod(noise_signal, M(i), symbol_enc);
        perfect_demod = pam_demod(normalized_signal, M(i),
    symbol_enc);

         [ ,BER(i,j) , ] =  biterr(noise_demod, perfect_demod);
    end
end

%==================================================

% 2. BER for 'gray'
M = [4 8 16];
SNR = 0:5:40;
symbol_enc = 'gray';
BER = [];

for i=1:length(M)
    input = 0:M(i)-1;
    X = randi([0 M(i)-1], 1, 100000);
    mod = pam_mod(input, M(i), symbol_enc);
    scale = modnorm(mod, 'avpow', 1);
    Eb = 1/log2(M(i));

```

```
    for j=1:length(SNR)
        div = Eb/2 * 10^(-SNR(j)/10);
        noise = sqrt(div) * randn(1, length(X));
        normalized_signal = scale * pam_mod(X, M(i), symbol_enc);
        noise_signal = normalized_signal + noise;

        noise_signal = noise_signal/scale;
        normalized_signal = normalized_signal/scale;

        noise_demod = pam_demod(noise_signal, M(i), symbol_enc);
        perfect_demod = pam_demod(normalized_signal, M(i), symbol_enc);

        [ ,BER(i,j) , ] = biterr(noise_demod, perfect_demod);
    end
end

%==================================================

% SER

%==================================================

% 3. SER for 'bin'
M = [4 8 16];
SNR = 0:5:40;
symbol_enc = 'bin';
BER = [];

for i=1:length(M)
    input = 0:M(i)-1;
    X = randi([0 M(i)-1], 1, 100000);
    mod = pam_mod(input, M(i), symbol_enc);
    scale = modnorm(mod, 'avpow', 1);
    Eb = 1/log2(M(i));

    for j=1:length(SNR)
        div = Eb/2 * 10^(-SNR(j)/10);
        noise = sqrt(div) * randn(1, length(X));
        normalized_signal = scale * pam_mod(X, M(i), symbol_enc);
        noise_signal = normalized_signal + noise;

        noise_signal = noise_signal/scale;
        normalized_signal = normalized_signal/scale;

        noise_demod = pam_demod(noise_signal, M(i), symbol_enc);
        perfect_demod = pam_demod(normalized_signal, M(i),
```

```
    symbol_enc);

          [ ,SER(i,j)] = symerr(noise_demod, perfect_demod);
    end
end

%==================================================

% 4. SER for 'gray'
M = [4 8 16];
SNR = 0:5:40;
symbol_enc = 'gray';
BER = [];

for i=1:length(M)
    input = 0:M(i)-1;
    X = randi([0 M(i)-1], 1, 100000);
    mod = pam_mod(input, M(i), symbol_enc);
    scale = modnorm(mod, 'avpow', 1);
    Eb = 1/log2(M(i));

    for j=1:length(SNR)
        div = Eb/2 * 10^(-SNR(j)/10);
        noise = sqrt(div) * randn(1, length(X));
        normalized_signal = scale * pam_mod(X, M(i), symbol_enc);
        noise_signal = normalized_signal + noise;

        noise_signal = noise_signal/scale;
        normalized_signal = normalized_signal/scale;

        noise_demod = pam_demod(noise_signal, M(i), symbol_enc);
        perfect_demod = pam_demod(normalized_signal, M(i), symbol_enc);

          [ ,SER(i,j)] = symerr(noise_demod, perfect_demod);
    end
end
```

```
function y = pam_mod(x,M,Symbol_Ordering)

% create constellation
const = (-(M-1):2:(M-1));

% Cast x into double
x = double(x);

% gray encode if necessary
if (strncmpi(Symbol_Ordering,'GRAY',size(Symbol_Ordering,2)))
```

```
    [ ,gray_map] = bin2gray(x,'pam',M);   % Gray encode
    [ ,index]=ismember(x,gray_map);
    x = index-1;
end

% modulate
C = const(:);
ynew = complex(C(x(:)+1));

% ensure y has same orientation as x
if isvector(x) && size(x,1) = size(ynew,1)
    y = ynew.';
else
    y = ynew;
end
```

```
function z = pam_demod(y,M,Symbol_Ordering)

y = y';

% move the real part of input signal; scale appropriately and round the
% values to get ideal constellation points
z = round( ((real(y) + (M-1)) ./ 2) );
% clip the values that are outside the valid range
z(z <= -1) = 0;
z(z > (M-1)) = M-1;

% gray decode if necessary
if (strncmpi(Symbol_Ordering,'GRAY',size(Symbol_Ordering,2)))
    [ ,gray_map] = gray2bin(z,'pam',M);   % Gray decode
    z = gray_map(z+1);
end

z = z';
```