Πανεπιστημιο Πατρων

Μηχανικων Ηλεκτρονικων Υπολογιστων & Πληροφορικης

Ψηφιακές Τηλεπικοινωνίες: 1η Εργαστηριακή Άσκηση

Ομοιόμορφη και μη ομοιόμορφη Παλμοκωδική Διαμόρφωση PCM

Συγγραφείς:
Κων/νος Αραβανής ΑΜ: 3628
Δημήτρης Λεβεντέας ΑΜ: 3688

Επιβλέπων:
Κων/νος Μπερμπερίδης

19 Ιανουαρίου 2009

Copyright ©2009 Kostantinos Aravanis, Dimitris Levendeas.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Περιεχόμενα

1 Εισαγωγή

Για να εκμεταλευτούμε τα πλεονεκτήματα της ψηφιακής μετάδοσης σημάτων (antijamming, ευκολία υλοποίησης, μεγαλύτερη ανοσία στον θόρυβο κ.α.) πρέπει να μετατρέψουμε το αναλογικό σήμα σε ψηφιακό. Για να επιτευχθεί αυτός ο σκοπός χρειαζόμαστε έναν δειγματολήπτη, που θα παίρνει τιμές (δείγματα) από το αναλογικό σήμα με τέτοιο τρόπο ώστε να μπορούμε αργότερα να το επεξεργαστούμε χωρίς μεγάλη απώλεια πληροφορίας σε σχέση με την αρχική. Στην συνέχεια περνάμε το σήμα μέσα από έναν κβαντιστή όπου αντιστοιχούμε τιμές που βρίσκονται μέσα σε ορισμένα διαστήματα σε μια συγκεκριμένη τιμή ώστε να συμπιέσουμε το σήμα μας (με απώλεια πληροφορίας). Τέλος κωδικοποιούμε το σήμα μας μέσω ενός κωδικοποιητή ώστε στην συνέχεια να μπορέσουμε να το μεταδώσουμε αποδοτικά. Στην πηγή αποκωδικοποιείται το σήμα ώστε να μπορέσουμε να διαβάσουμε την μεταδιδόμενη πληροφορία. Ένα επιπρόσθετο βήμα που υπάρχει σε αρκετές περιπτώσεις είναι η χρήση συμπιεστή πριν την κβάντιση ώστε να έχουμε καλύτερα αποτελέσματα ως προς την ποιότητα του σήματος.

Στην παρούσα εργασία ασχοληθήκαμε με όλους τους παραπάνω τομείς εκτός από την διαδικασία δειγματολήπτησης από ένα αναλογικό σήμα για πρακτικούς λόγους.

2 PCM κωδικοποίηση - Παλμοκωδική Διαμόρφωση

Η παλμοκωδική διαμόρφωση (PCM) είναι το παλιότερο και απλούστερο σχήμα ψηφιακής διαμόρφωσης αναλογικών δεδομένων. Αποτελείται από τρία βασικά μέρη:

- δειγματολήπτη
- κβαντιστή
- κωδικοποιητή

όπως φαίνονται και στο σχήμα 1.

Σχήμα 1: Διάγραμμα βαθμίδων ενός συστήματος PCM

Πριν την εισαγωγή στο δειγματολήπτη ενός σήματος, αυτό έχει περάσει (συνήθως) από ένα προδειγματοληπτικό φίλτρο που εμποδίζει την είσοδο συνιστωσών του σήματος πέρα από το εύρος ζώνης W που μας ενδιαφέρει.

Στην περίπτωση μας, επειδή επεξεργαζόμαστε σήμα φωνής μέσω της συνάρτησης *wavread()* της Matlab το σήμα μας περιορίζεται σε τιμές στο διάστημα από -1 έως 1.

Αφού το σήμα περάσει από τον δειγματολήπτη (όπου συνήθως δειγματοληπτείται με συχνότητα μεγαλύτερη από αυτή του Nyquist), εισέρχεται σε έναν βαθμωτό κβαντιστή. Ο κβαντιστής αυτός μπορεί να είναι είτε ομοιόμορφος είτε μη ομοιόμορφος ανάλογα με τα στατιστικά χαρακτηριστικά της εξόδου της πηγής.

Μετά την κβάντιση του το σήμα κωδικοποιείται από τον κωδικοποιητή με μια δυαδική ακολουθία μήκους ν όπου $N = 2^{\nu}$ είναι ο αριθμός των σταθμών κβάντισης.

2.1 Ομοιόμορφος κβαντιστής

Ο ομοιόμορφος κβαντιστής περιορίζει την δυναμική περιοχή του σήματος εισόδου στις τιμές $[-max\_value, max\_value]$ θέτοντας όποιες τιμές είναι εκτός αυτού του εύρους στις αντίστοιχες ακραίες τιμές.

Η διαδικασία η οποία ακολουθείται είναι αφού ο κβαντιστής χωρίσει την περιοχή σε N τμήματα, αντιστοιχεί τις τιμές που πέφτουν σε κάθε ένα από αυτά τα τμήματα στην τιμή του μέσου της αντίστοιχης περιοχής κβάντισης. Οι περιοχές στις οποίες χωρίζεται το αρχικό εύρος τιμών του σήματος είναι ισομήκης Σύμφωνα με τα παραπάνω, κάθε περιοχή έχει εύρος $\Delta = \frac{2x_{max}}{N} = \frac{x_{max}}{2^{\nu-1}}$. Συνεπώς και τα κέντρα κβάντισης απέχουν μεταξύ τους την ίδια απόσταση Δ. Στο σχήμα 2 βλέπουμε ένα ομοιόμορφα κβαντισμένο ημίτονο.

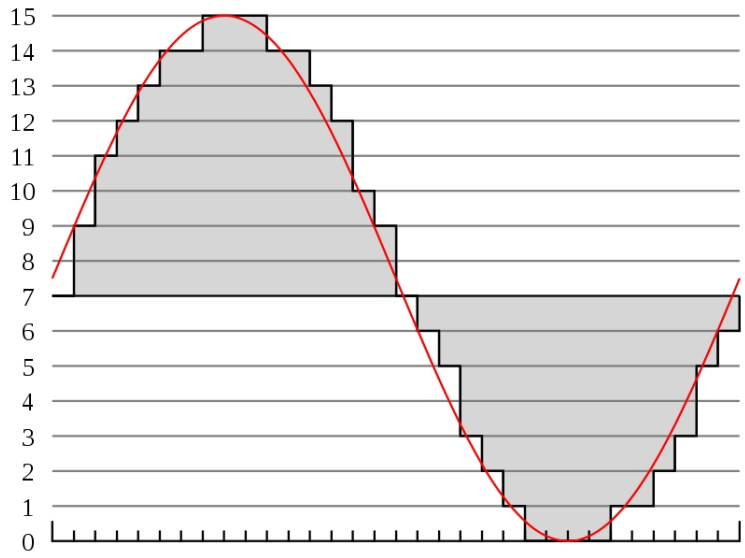
Ως μέση τετραγωνική παραμόρφωση (distortion) καλούμε το μέτρο $D = E[(X - Q(X))^2]$ με X τυχαία μεταβλητή που αντιστοιχεί στο σήμα εισόδου και $Q(X)$ το κβαντισμένο σήμα. Ένα από τα πιο σημαντικά μέτρα της επίδοσης της κβάντισης είναι μια κανονικοποιημένη έκδοση του θορύβου κβάντισης ως προς την ισχύ του αρχικού σήματος:

$$SQNR = \frac{E[X^2]}{E[(X - Q(X))^2]} \tag{1}$$

$$= \frac{\sigma^2}{D} \tag{2}$$

Παρακάτω παραθέτουμε τον κώδικα στην matlab που προσομοιώνει την λειτουργία του ομοιόμορφου κβαντιστή.

```
function [xq, centers, p, D] = my_quantizer(x, N, max_value)
% Input arguments
% x: input signal, already in discrete form
% N: number of bits that will be used
% max_value: maximum acceptable value of the signal
```



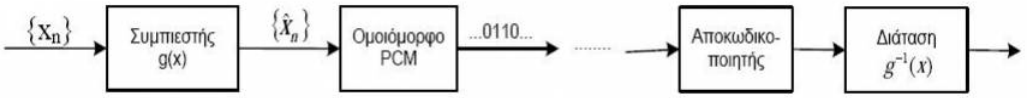
Σχήμα 2: Δειγματοληπτημένο και ομοιόμορφα κβαντισμένο σήμα ημιτόνου για 4-bit PCM

```
% Output arguments
% xq: the vector of output signal
% centers: centers of quantized region
% p: vector that includes the appearance probability of each center of the
% quantizer
% D: distortion of the quantized signal

% the number of quantized regions
num_of_regions = 2^N;

% how many times we meet a value in a specific interval
occurrences = zeros(num_of_regions, 1);

% length of each region
% each region has the same length
length_of_region = 2 * max_value / num_of_regions;

% the limits of each region without the "infinities"
vector_of_intervals = zeros(num_of_regions - 1, 1);

num_of_intervals = length(vector_of_intervals);

for j=1:1:num_of_intervals
    vector_of_intervals(j, 1) = - max_value + j * length_of_region;
end

% the number of input samples
num_of_samples = length(x);

centers = zeros(num_of_regions, 1);

% calculate the center of each region
for j = 1: 1: num_of_intervals
    centers(j, 1) = vector_of_intervals(j, 1) - length_of_region / 2;
end
centers(num_of_regions, 1) = vector_of_intervals(num_of_intervals, 1) + length_of_region / 2;
```

```
for j = 1: 1: num_of_samples
    pos = binarysearch(x(j, 1), vector_of_intervals);
    xq(j, 1) = centers(pos, 1);
    occurrences(pos, 1) = occurrences(pos, 1) + 1;
end

for j = 1: 1: num_of_regions
    p(j, 1) = occurrences(j, 1) / num_of_samples;
end

D = distortion(x, xq);

return;
```

2.2 Μη ομοιόμορφο PCM

Κατά την κωδικοποίηση ορισμένων σημάτων όπως η ομιλία, η κατανομή της εισόδου απέχει πολύ από την ομοιόμορφη, σε αυτές τις περιπτώσεις προτιμούμε το μη ομοιόμορφο PCM. Η συνηθέστερη μέθοδος για την υλοποίηση της μη ομοιόμορφης κβάντισης είναι τα δείγματα να διέλθουν πρώτα από ένα μη γραμμικό στοιχείο προκειμένου να συμπιεστούν τα μεγάλα πλάτη (μείωση δυναμική περιοχής του σήματος) και στην συνέχεια η έξοδος του μη γραμμικού σημείου να κβαντιστεί ομοιόμορφα. Στη λήψη εφαρμόζεται η αντίστροφη λειτουργία της συμπίεσης (διάταση) για να ανακτήσουμε τις τιμές των δειγμάτων. Η τεχνική αυτή ονομάζεται companding και περιγράφεται από το σχήμα 3.



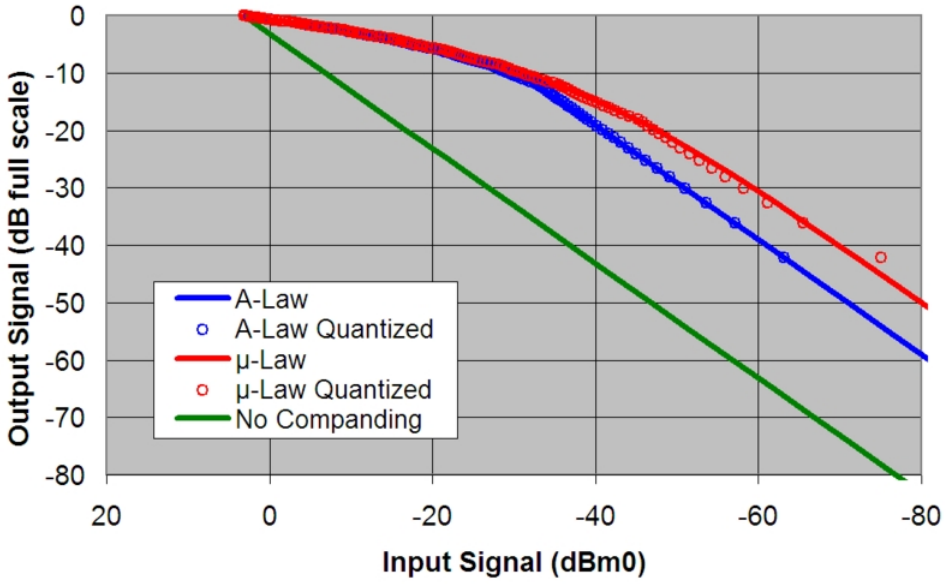
Σχήμα 3: Διάγραμμα βαθμίδων ενός συστήματος μη ομοιόμορφου PCM

Οι αλγόριθμοι συμπίεσης έχουν την ιδιότητα να μειώνουν δυναμικό εύρος ενός σήματος ήχου. Στα ψηφιακά συστήματα μπορούν να αυξήσουν το SQNR μειώνοντας το σφάλμα κβάντισης.

Για την κωδικοποίηση ομιλίας δύο τύποι συμπιεστών χρησιμοποιούνται ευρέως. Ο συμπιεστής τύπου-μ, ο οποίος χρησιμοποιείται στην Βόρεια Αμερική και Ιαπωνία, και ο συμπιεστής τύπου-Α που χρησιμοποιείται κυρίως στην Ευρώπη. Ο συμπιεστής τύπου-μ παρέχει ένα ελαφρώς δυναμικό εύρος σε σχέση με τον συμπιεστή τύπου-Α με το κόστος χειρότερης αναλογικής παραμόρφωσης για μικρά σήματα. Από σύμβαση, ο συμπιεστής τύπου-Α χρησιμοποιείται για διεθνής επικοινωνίες όταν τουλάχιστον μια χώρα τον χρησιμοποιεί.

Στο γράφημα 4 συγκρίνονται ο μ-συμπιεστής με τον Α-συμπιεστή.

Οι τύποι που περιγράφουν την διαδικασία συμπίεσης και διάτασης των αντίστοιχων αλγορίθμων παρουσιάζονται παρακάτω:



Σχήμα 4: Γράφημα του μ-συμπιεστή και Α-συμπιεστή

2.2.1 Συμπιεστής Τύπου-μ

Συμπίεση:

$$f(x) = sgn(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)}, -1 \leq x \leq 1 \tag{3}$$

Αποσυμπίεση:

$$f^{-1}(x) = sgn(x)\frac{1}{\mu}((1+\mu)^{|x|}-1), -1 \leq x \leq 1 \tag{4}$$

2.2.2 Συμπιεστής Τύπου-Α

Συμπίεση:

$$g(x) = \begin{cases} sign(x)\dfrac{A|x|}{1+\ln A}, & |x| \leq \dfrac{1}{A} \\ sign(x)\dfrac{1+\ln(A|x|)}{1+\ln A}, & \dfrac{1}{A} \leq |x| \leq 1 \end{cases} \tag{5}$$

Αποσυμπίεση:

$$g^{-1}(x) = \begin{cases} sign(x)\dfrac{(1+\ln A)|x|}{A}, & |x| \leq \dfrac{1}{1+\ln A} \\ sign(x)\dfrac{e^{|x|(1+\ln A)-1}}{A}, & \dfrac{1}{1+\ln A} < |x| \leq 1 \end{cases} \quad (6)$$

2.3 Υλοποίηση συμπιεστών - αποσυμπιεστών

2.3.1 Συμπιεστής Τύπου-μ

```
function [xc] = compressor_m(x, m)
% Input arguments
% x: input signal, already in discrete form
% m: coeffivient of compression
% Output arguments
% xc: the x signal in compressed form

% the number of input samples
num_of_samples = length(x);

% error message
Error = 'We could not compress this signal because it is expanded over a large area';

for i = 1: 1: num_of_samples
    % if the signal x is expanded over the limits -1<=x<=1, can not be compressed
    if (x(i,1) < -1) || (x(i,1) > 1)
        Error
        return;
    end

    % compress each value of the discrete signal
    xc(i,1) = sign( x(i,1) ) * reallog(1 + m * abs( x(i,1) ) ) / reallog( 1 + m ) ;
end

return;
```

2.3.2 Αποσυμπιεστής Τύπου-μ

```
function [xd] = decompressor_m(x, m)
% Input arguments
% x: input signal, already in discrete form
% m: coeffivient of decompression
% Output arguments
% xd: the x signal in uncompressed form

% the number of input samples
num_of_samples = length(x);

% error message
Error = 'We could not decompress this signal because it is expanded over a large area';

for i = 1: 1: num_of_samples
    % if the signal x is expanded over the limits -1<=x<=1, can not be decompressed
    if (x(i,1) < -1) || (x(i,1) > 1)
        Error
        return;
    end
```

```
    % decompress each value of the discrete signal
    xd(i,1) = sign( x(i,1) ) * ( (1 + m) ^ abs( x(i,1) ) -1 ) / m ;
end

return;
```

2.3.3 Συμπιεστής Τύπου-Α

```
function [xc] = compressor_a(x, a)
% Input arguments
% x: input signal, already in discrete form
% a: coeffivient of compression
% Output arguments
% xc: the x signal in compressed form

% the number of input samples
num_of_samples = length(x);

% error message
Error = 'We could not compress this signal because it is expanded over a large area';

for i = 1: 1: num_of_samples
    % if the signal x is expanded over the limits 0<=|x|<=1, can not be compressed
    if (x(i,1) < -1) || (x(i,1) > 1)
        Error
        return;
    % compress each value of the discrete signal
    elseif ( abs( x(i,1) ) <= 1/a )
        xc(i,1) = sign( x(i,1) ) * a * abs( x(i,1) ) / ( 1 + reallog(a) );
    else
        xc(i,1) = sign( x(i,1) ) * ( 1 + reallog( a * abs( x(i,1) ) ) ) / (1 + reallog(a));
    end
end
return;
```

2.3.4 Αποσυμπιεστής Τύπου-Α

```
function [xd] = decompressor_a(x, a)
% Input arguments
% x: input signal, already in discrete form
% a: coeffivient of decompression
% Output arguments
% xd: the x signal in decompressed form

% the number of input samples
num_of_samples = length(x);

% error message
Error = 'We could not decompress this signal because it is expanded over a large area';

for i = 1: 1: num_of_samples
    % if the signal x is expanded over the limits 0<=|x|<=1, can not be decompressed
    if (x(i,1) < -1) || (x(i,1) > 1)
        Error
        return;
    % decompress each value of the discrete signal
    elseif ( abs( x(i,1) ) <= 1/( 1 + reallog(a) ) )
        xd(i,1) = sign( x(i,1) ) * ( 1 + reallog( a)) * abs( x(i,1) ) / a;
    else
        xd(i,1) = sign( x(i,1) ) * exp( abs( x(i,1) ) * ( 1 + reallog(a) )  - 1 ) / a;
```

```
    end
end
return;
```

2.4 Μη ομοιόμορφος κβαντιστής

Χαλαρώνοντας την συνθήκη από τον ομοιόμορφο κβαντιστή όπου όλες οι περιοχές κβάντισης (εκτός πρώτης και τελευταίας) πρέπει να έχουν το ίδιο εύρος, μπορούμε να δημιουργήσουμε έναν κβαντιστή που θα λειτουργεί με καλύτερες επιδόσεις σε σύγκριση με έναν ομοιόμορφο ίδιων σταθμών. Αυτό συμβαίνει γιατί μπορούμε να κάνουμε την ελαχιστοποίηση της παραμόρφωσης με λιγότερους περιορισμούς, αν και ο κβαντιστής που θα προκύψει θα είναι πιο πολύπλοκος σε σχέση με έναν ομοιόμορφο.

Για να έχουμε έναν βέλτιστο κβαντιστή, τα άκρα των περιοχών κβάντισης θα πρέπει να δίνονται από τον αριθμητικό μέσο των γειτονικών τιμών κβάντισης. Έτσι η κβάντιση γίνεται με βάση την ελάχιστη απόσταση, δηλαδή κάθε τιμή x κβαντίζεται στο πλησιέστερο $\{\hat{x}_i\}_{i=1}^{N}$.

Σύμφωνα με την παραπάνω παρατήρηση, για να έχουμε έναν βέλτιστο βαθμωτό κβαντιστή, πρέπει να πληρούνται οι συνθήκες Lloyd-Max. Τα κριτήρια για την κβάντιση συνοψίζονται:

1. Τα άκρα των περιοχών κβάντισης δίνονται από τον αριθμητικό μέσο των γειτονικών τιμών κβάντισης (νόμος πλησιέστερου γείτονα)

2. Οι τιμές κβάντισης είναι τα κέντρα μάζας των περιοχών κβάντισης

Δυστυχώς, παρόλου που αυτοί οι κανόνες είναι πολύ απλοί, δεν δίνουν αναλυτικές λύσεις για την σχεδίαση του βέλτιστου κβαντιστή. Για αυτό τον λόγο η πιο συνηθισμένη μέθοδος σχεδιασμού είναι να ξεκινήσουμε με ένα σύνολο περιοχών κβάντισης και συνεχίζουμε χρησιμοποιώντας το δεύτερο κριτήριο. Ακολούθως επανασχεδιάζουμε τις περιοχές κβάντισης και επαναλαμβάνουμε τα δυο παραπάνω βήματα μέχρι η παραμόρφωση από βήμα σε βήμα να έχει μειωθεί κάτω από μια επιθυμητή τιμή.

Ο αλγόριθμος συνοψίζεται στα εξής βήματα. Σε κάθε επανάληψη i του Lloyd-Max:

1. Υπολογίζουμε τα όρια των ζωνών κβαντισμού, που πρέπει να είναι στο μέσο των επιπέδων κβαντισμού, δηλαδή:

$$T_k = \frac{\tilde{x}_k^{(i)} + \tilde{x}_{k+1}^{(i)}}{2},\ 1 \leq k \leq M-1$$

2. Υπολογίζουμε το κβαντισμένο σήμα με βάση τις περιοχές αυτές και μετράμε την μέση παραμόρφωση D_i με βάση το δοθέν σήμα.

3. Τα νέα επίπεδα κβαντισμού είναι τα κεντροειδή των ζωνών:

$$\tilde{x}_k^{(i+1)} = E[x|T_{k-1} < x < T_k]$$

4. Επαναλαμβάνουμε τα τρία πρώτα βήματα μέχρι ότου:

$$|D_i - D_{i-1}| < \varepsilon$$

όπου ε καθορίζει και τον αριθμό των K_{max} επαναλήψεων

Παρακάτω παραθέτουμε τον κώδικα στην matlab που προσομοιώνει την λειτουργία του μη ομοιόμορφου κβαντιστή χρησιμοποιώντας τα κριτήρια Lloyd-Max και σύμφωνα με τον παραπάνω αλγόριθμο.

```
function [xq, centers, D] = Lloyd_Max(x, N, max_value)
% Input arguments
% x: input signal, already in discrete form
% N: number of bits that will be used
% max_value: maximum acceptable value of the signal
% Output arguments
% xq: the vector of output signal after K_{max} loops
% centers: centers of quantized region
% D: vector of distortions of the quantized signal of each iteration

% fault tolerrance
epsilon = eps;

% initial value of distortion
D(1,1) = 1;

% the number of quantized regions
num_of_regions = 2^N;

% length of each region
% each region has the same length
length_of_region = 2 * max_value / num_of_regions;

% the limits of each region without the "infinities"
vector_of_intervals = zeros(num_of_regions - 1, 1);

num_of_intervals = length(vector_of_intervals);

% initialization of the intervals according to a uniform distribution
for j=1:1:num_of_intervals
    vector_of_intervals(j, 1) = - max_value + j * length_of_region;
end

% the vector thet has the expected values of each region
expected_value = zeros(num_of_regions,1);

% Initialization of the expected values
for j=1:1:num_of_intervals
    expected_value(j, 1) = vector_of_intervals(j, 1) - length_of_region / 2;
end
expected_value(num_of_regions, 1) = vector_of_intervals(num_of_intervals, 1) + length_of_region / 2;

% the number of input samples
num_of_samples = length(x);
```

```
for i = 2:1:10^5

    % how many times we meet a value in a specific interval
    occurrences = zeros(num_of_regions, 1);

    % the vector that has the mass value the interval
    mass_value = zeros(num_of_regions,1);

    % Find in which interval each sample belongs
    % Count how many occurrences there are in a specific interval and calculate
    % their sum in order to find the expected value of this interval
    for j = 1: 1: num_of_samples
        pos = binarysearch(x(j, 1), vector_of_intervals);
        occurrences(pos, 1) = occurrences(pos, 1) + 1;
        mass_value(pos,1) = mass_value(pos,1) + x(j,1);
        xq(j, 1) = expected_value(pos, 1);
    end

    % Calculate the expected value for the next iteration
    for j = 1: 1: num_of_regions
        if mass_value(j,1) == 0
            if j ~= 1
               expected_value(j,1) = vector_of_intervals(j-1,1);
            else
               expected_value(j,1) = vector_of_intervals(j,1);
            end
        else
            expected_value(j,1) = mass_value(j,1) / occurrences(j,1);
        end
    end

    % Redifine the limits of each region for the next iteration
    for j = 1: 1: num_of_intervals
        vector_of_intervals(j,1) = ( expected_value(j,1) + expected_value(j + 1, 1) ) / 2;
    end

    % calculate the distortion
    D(i,1) = distortion(x, xq);

    if abs(D(i , 1) - D(i - 1, 1)) < epsilon
        centers = expected_value;
        return;
    end

end
return;
```

3 Πειραματική Αξιολόγηση

Κωδικοποιήσαμε τα δείγματα της πηγής για $N = 2, 4, 6$ bits σύμφωνα με τις παραπάνω μεθόδους και αξιολογήσαμε τα αποτελέσματα βασιζόμενοι:

1. Στις τιμές του *SQNR*.[1]

[1] Στην περίπτωση του μη ομοιόμορφου κβαντιστή που κατασκευάζεται με τον αλγόριθμο Lloyd-Max παραθέτουμε και διαγράμματα που αναπαριστούν την μεταβολή του SQNR σε σχέση με τον αριθμό των επαναλήψεων του αλγορίθμου Lloyd-Max

2. Στο *ακουστικό αποτέλεσμα* κάθε μεθόδου χρησιμοποιώντας την συνάρτηση *sound()* της Matlab.[2]

3. Στις κυματομορφές εξόδου.

3.1 Με βάση το SQNR

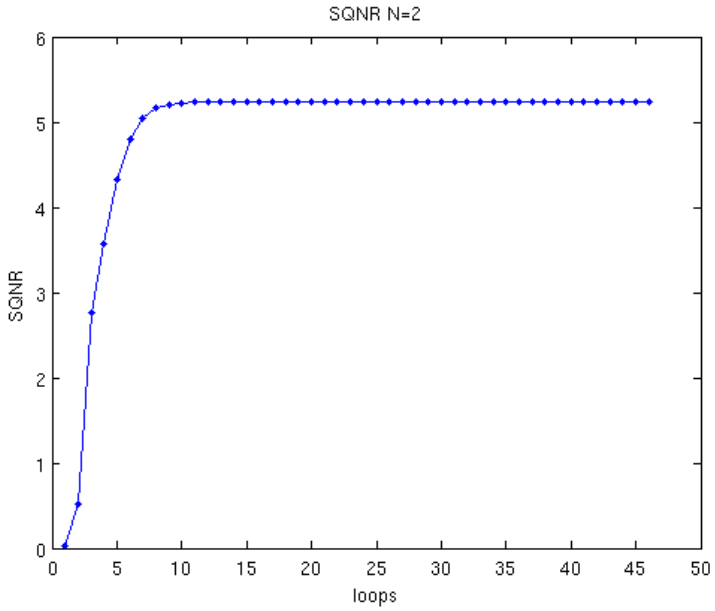
Με βάση τον τύπο υπολογισμού του $SQNR$ όπως αναφέρθηκε στην εξίσωση 2, υπολογίσαμε τις παρακάτω τιμές που δείχνουν μια μετρική της ποιότητας του σήματος μετά την κβάντιση του.

Τα αποτελέσματα για τις τρεις πρώτες μεθόδους φαίνονται στον Πίνακα 1.

	$N=2$	$N=4$	$N=6$
Ομοιόμορφο PCM	0.513	11.90	234.69
Συμπιεστής τύπου-μ	2.332	22.175	389.84
Συμπιεστής τύπου-A	2.414	23.593	414.94

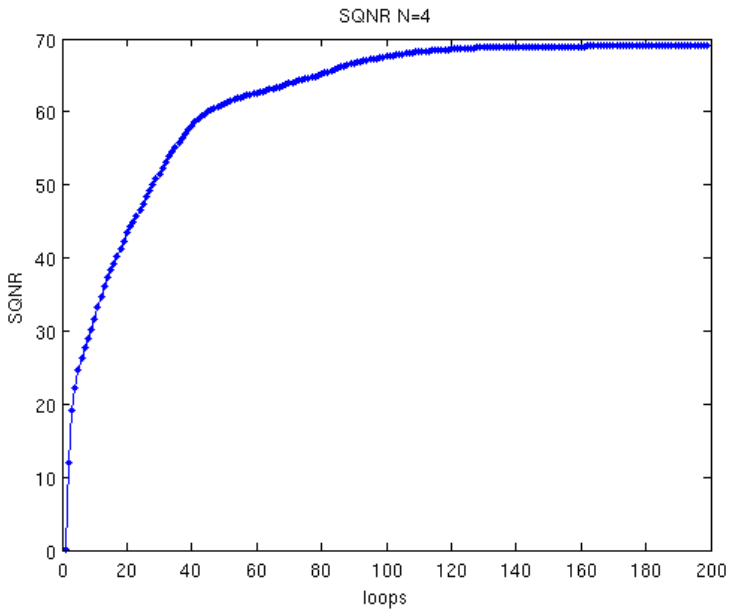
Πίνακας 1: Τιμές SQNR

Ενώ για την τέταρτη μέθοδο τα αποτελέσματα απεικονίζονται στα διαγράμματα 5, 6, 7 και συγκεκριμένα σε αυτά παρουσιάζεται η μεταβολή του *SQNR* σε σχέση με τον αριθμό των επαναλήψεων του αλγορίθμου Lloyd-Max.

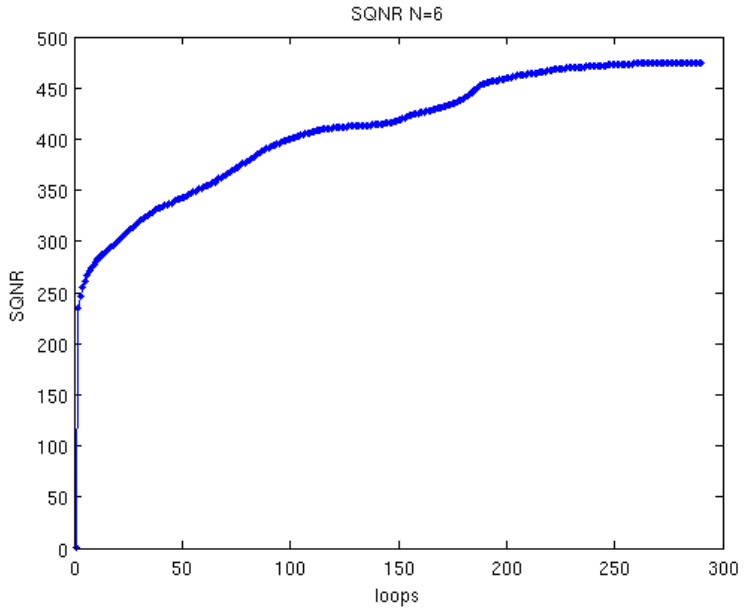


Σχήμα 5: Μεταβολή SQNR για κβάντιση 2-bit

[2]Η wavplay() απαιτεί λειτουργικό Microsoft Windows® ενώ η πραγματοποίηση των πειραμάτων έγινε σε GNU/Linux, Debian Lenny



Σχήμα 6: Μεταβολή SQNR για κβάντιση 4-bit



Σχήμα 7: Μεταβολή SQNR για κβάντιση 6-bit

Παρατηρούμε πως όσο αυξάνονται τα bit που χρησιμοποιούμε για την κβάντιση, τόσο καλύτερα αποτελέσματα παίρνουμε. Αναμενόμενο καθώς όπως γνωρίζουμε ισχύει ότι:

$$H(X) \leq \bar{R} \leq H(X) + \frac{1}{n} \tag{7}$$

Από τα παραπάνω αποτελέσματα φαίνεται πως ο ομοιόμορφος κβαντιστής εμφανίζει τη χαμηλότερη επίδοση όσον αφορά το εισαγόμενο σφάλμα μετά τη κβάντιση Αυτό συμβαίνει καθώς κβαντίζει το σήμα μας σε προκαθορισμένες στάθμες ίδιου εύρους, ανεξάρτητα από το σήμα. Αντίθετα, η μη-ομοιόμορφη κβάντιση χρησιμοποιώντας τα κριτήρια Lloyd-Max δίνει τα καλύτερα αποτελέσματα μιας και σκοπό της έχει να κβαντίσει το σήμα με την όσο το δυνατών λιγότερη παραμόρφωση, υπό το κόστος όμως περισσότερων υπολογισμών και αυξημένης πολυπλοκότητας υλοποίησης.

Όσον αφορά τους συμπιεστές, παρατηρούμε πως είναι μια πολύ καλή πρακτική λύση καθώς αν και δεν εμφανίζουν την πολύ καλή απόδοση του μη ομοιόμορφου κβαντιστή, αλλά χρησιμοποιώντας μόνο ένα μη γραμμικό στοιχείο μπορούμε να επιτύχουμε αρκετά ικανοποιητικά αποτελέσματα στην πράξη.

Τέλος, παρατηρούμε πως για την μέθοδο Lloyd-Max όσο αυξάνονται τα bits που χρησιμοποιούμε, τόσα πιο πολλά iterations χρειάζονται για τον τερματισμό του αλγορίθμου φθάνοντας στην τιμή στόχο, που καθορίζεται από την μικρή μεταβολή της παραμόρφωσης (distortion). Αυτό είναι και αναμενόμενο αν λάβουμε υπόψη μας πως για λίγα bits αναμένουμε μεγάλη παραμόρφωση και επομένως θα την φθάσουμε σχετικά γρήγορα και από εκεί και πέρα θα αλλάζει ελάχιστα.

3.2 Με βάση το ακουστικό αποτέλεσμα

Για να κάνουμε αυτές τις παρατηρήσεις ακούγαμε το αρχικό σήμα και στην συνέχεια το κβαντισμένο μέσω της συνάρτησης *sound()*.

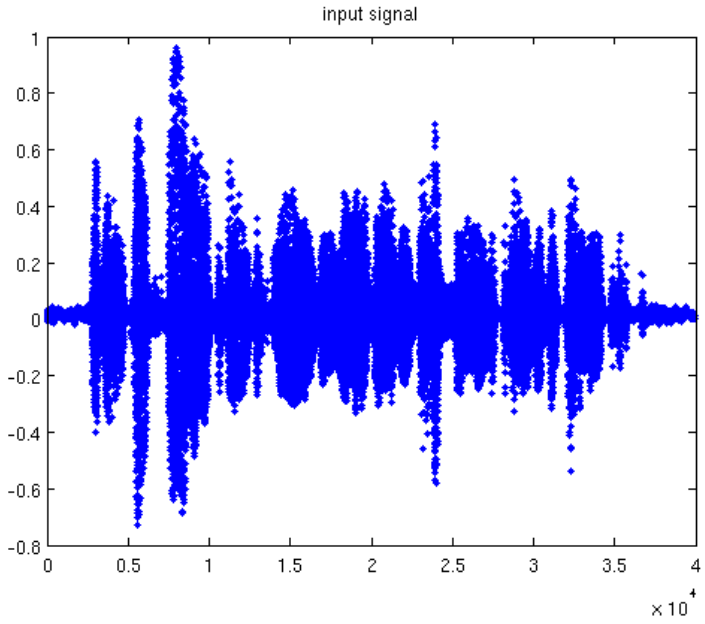
Αυξάνοντας τον αριθμό των bits κβάντισης παρατηρούσαμε μεγάλη διαφορά στην ποιότητα, και ειδικότερα από τα 2-bit στα 4-bit όπου ο περισσότερος θόρυβος έφευγε. Αντιληπτή ήταν επίσης και η διαφορά στην ποιότητα ανάμεσα σε ομοιόμορφο κβαντιστή και μη ομοιόμορφο σύμφωνα με Lloyd-Max, αλλά όχι στον ίδιο βαθμό. Για $N = 6$ το σήμα που ακούγαμε στην έξοδο ακουγόταν σχεδόν το ίδιο με αυτό της εισόδου.

Αυτό ήταν και το πιο εντυπωσιακό μέρος της εργασίας μαζί με τις κυματομορφές εξόδου καθώς μπορούσαμε να ακούσουμε το σήμα μέσα από το διάνυσμα με το οποίο περιγραφόταν.

3.3 Με βάση τις κυματομορφές εξόδου

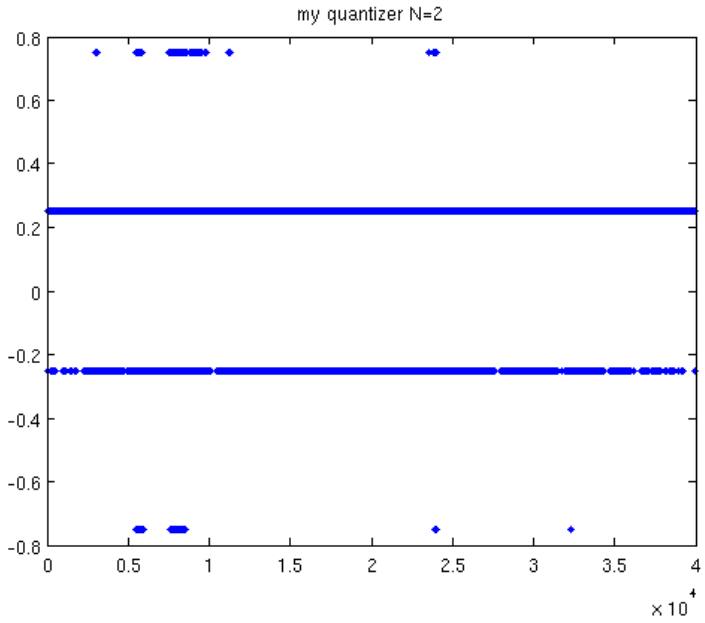
Για να συγκρίνουμε το αποτέλεσμα μετά τη κβάντιση τού σήματος με το αρχικό πήραμε την κυματομορφή εισόδου του αρχικού σήματος (Σχήμα 8 και την συγκρίναμε με τη κυματομορφή του κβαντισμένου σήματος. Όπως μπορεί να διαπιστώσει κάποιος από τα παρακάτω σχήματα μπορούν να εξαχθούν παρόμοια

συμπεράσματα όπως και παραπάνω, συγκρίνοντας το κατά πόσο το κάθε κβαντισμένο σήμα προσεγγίζει το αρχικό μας.

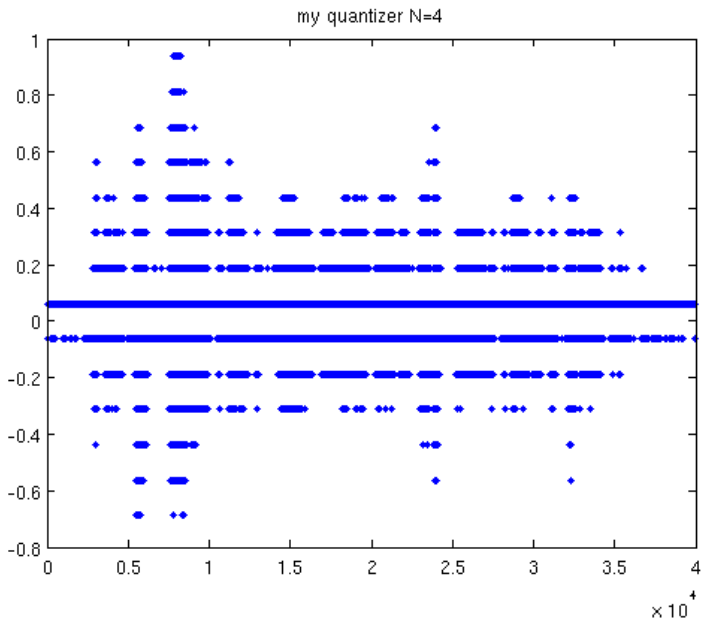


Σχήμα 8: Κυματομορφή αρχικού σήματος

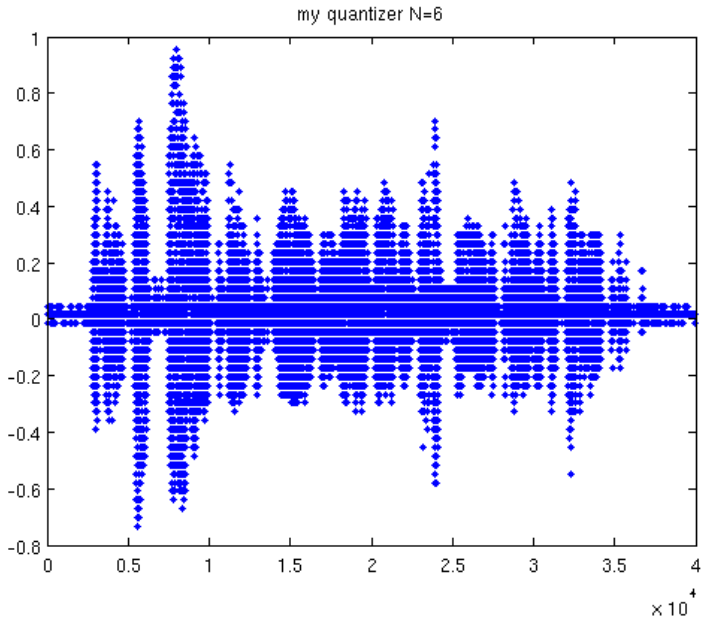
3.3.1 Ομοιόμορφη Κβάντιση



Σχήμα 9: Κυματομορφή εξόδου για ομοιόμορφη κβάντιση 2-bit

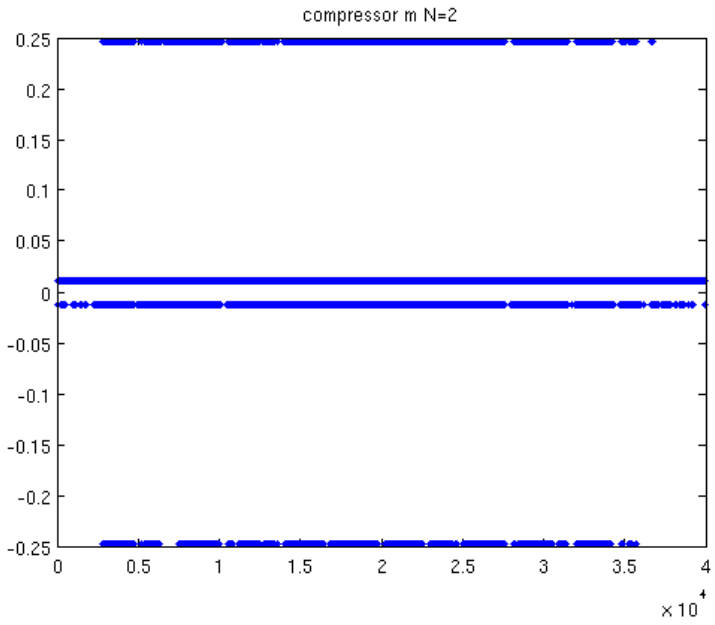


Σχήμα 10: Κυματομορφή εξόδου για ομοιόμορφη κβάντιση 4-bit

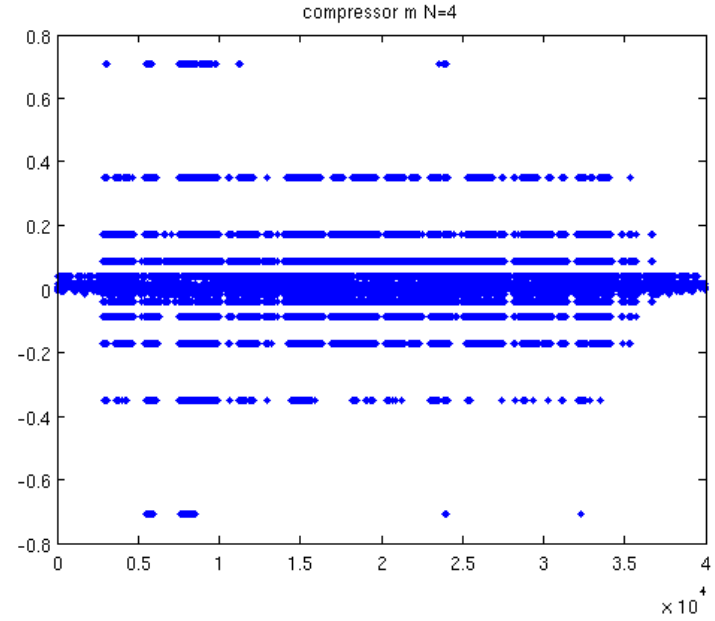


Σχήμα 11: Κυματομορφή εξόδου για ομοιόμορφη κβάντιση 6-bit

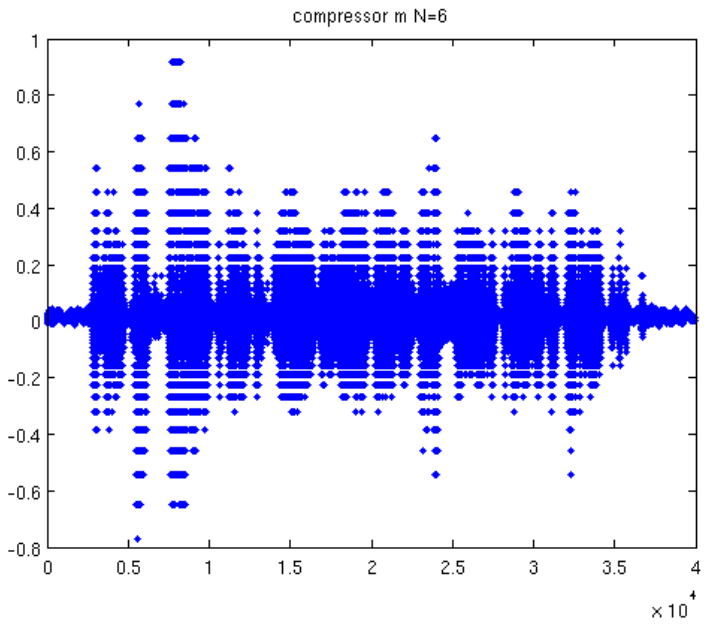
3.3.2 Μη ομοιόμορφο PCM με συμπιεστή τύπου-μ



Σχήμα 12: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 2-bit τύπου-μ

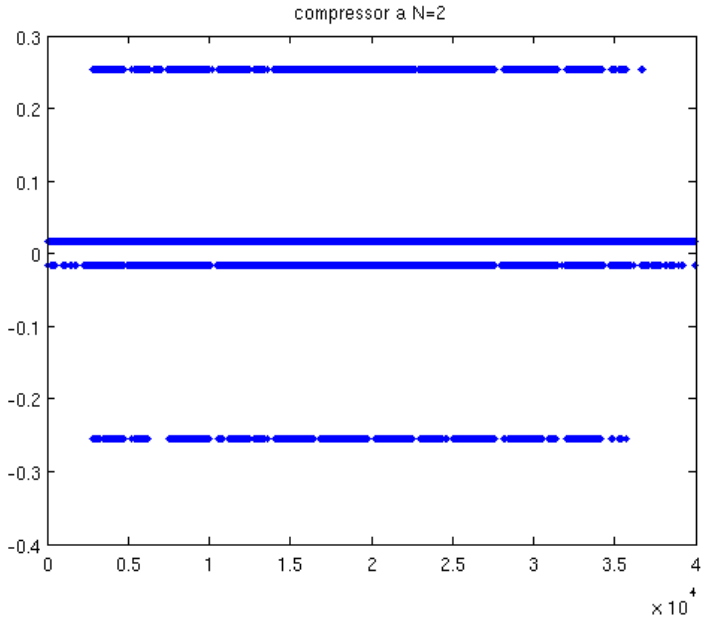


Σχήμα 13: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 4-bit τύπου-μ

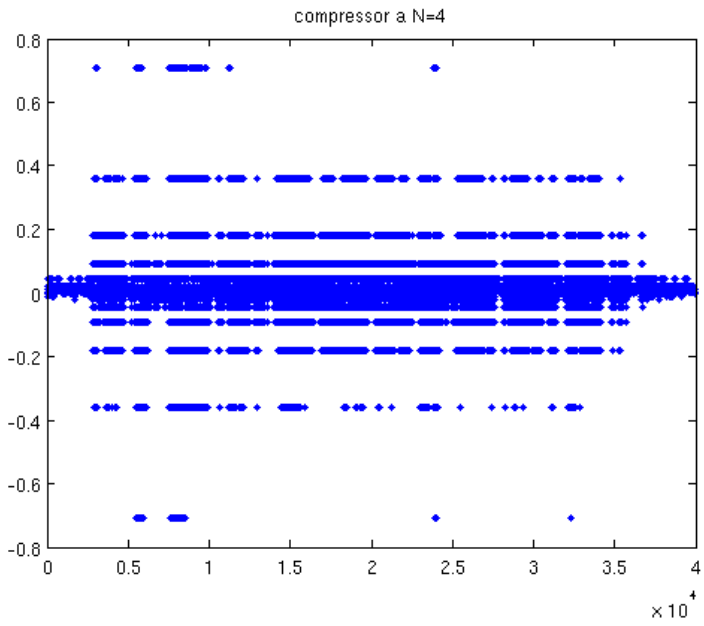


Σχήμα 14: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 6-bit τύπου-μ

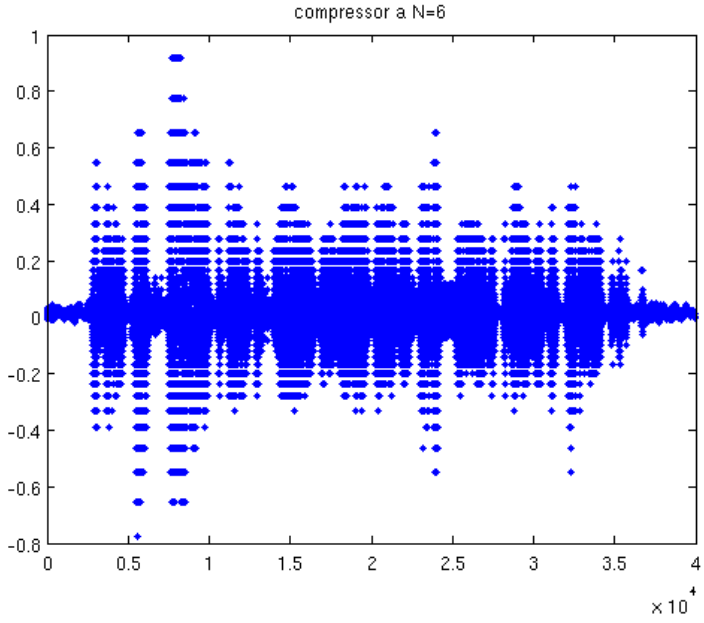
3.3.3 Μη ομοιόμορφο PCM με συμπιεστή τύπου-A



Σχήμα 15: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 2-bit τύπου-A

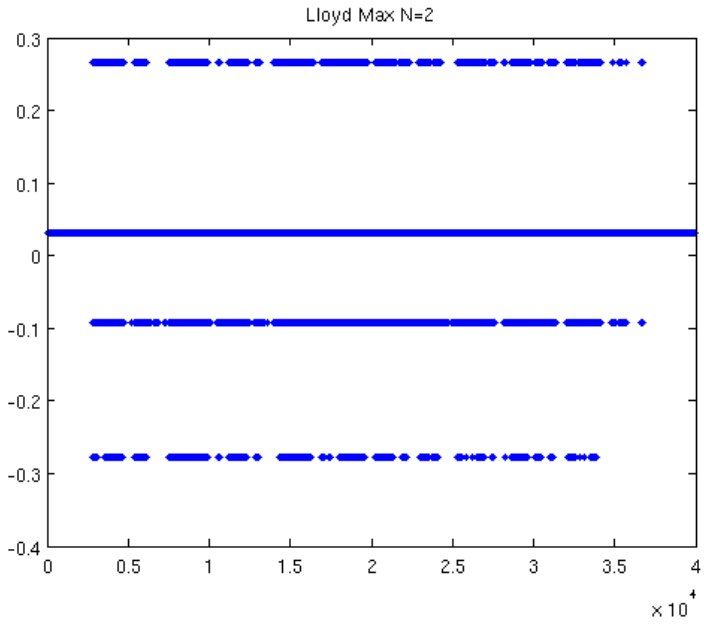


Σχήμα 16: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 4-bit τύπου-Α

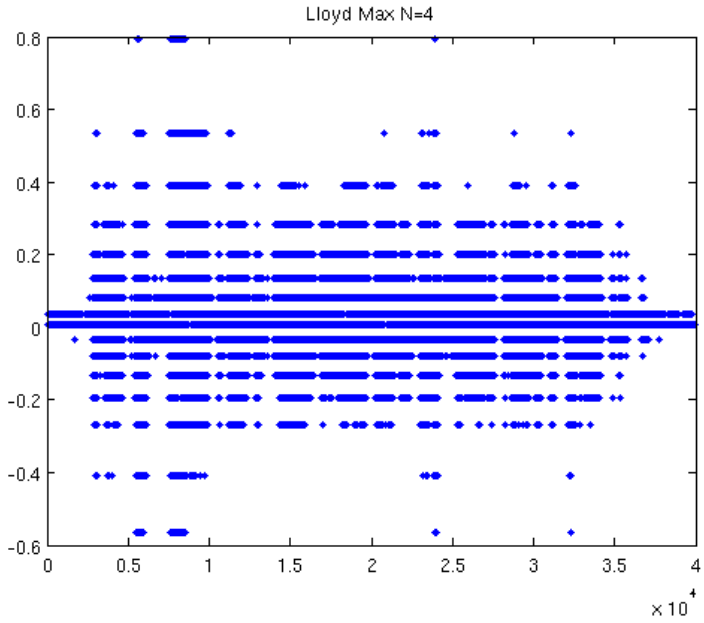


Σχήμα 17: Κυματομορφή εξόδου για μη ομοιόμορφο PCM 6-bit τύπου-Α

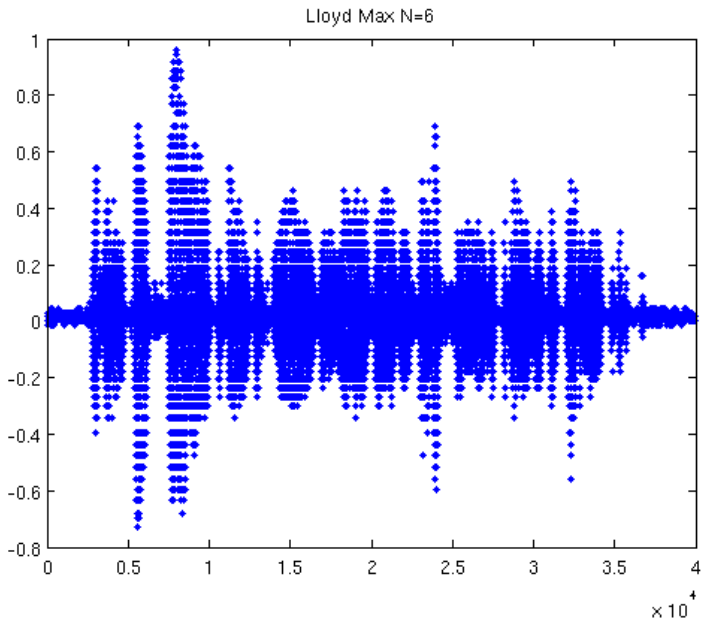
3.3.4 Μη ομοιόμορφη κβάντιση



Σχήμα 18: Κυματομορφή εξόδου για μη ομοιόμορφο κβαντιστή 2-bit



Σχήμα 19: Κυματομορφή εξόδου για μη ομοιόμορφο κβαντιστή 4-bit



Σχήμα 20: Κυματομορφή εξόδου για μη ομοιόμορφο κβαντιστή 6-bit

3.4 Υλοποίηση σε Matlab

Παρακάτω παραθέτουμε τον κώδικα που χρησιμοποιήσαμε σε Matlab για την εξαγωγή των παραπάνω συμπερασμάτων. Πρόκειται για ένα αυτοματοποιημένο script όπου μετά την εκτέλεση του παράγονται όλα τα παραπάνω αποτελέσματα. Οι συναρτήσεις που η υλοποίηση τους δεν έχει παρουσιαστεί παραπάνω, υπάρχει στο παράρτημα.

```
% SCRIPT FOR THE EXERCISE 2

%we print the results with long format
format long;

% we read the input signal
A=wavread('speech');

% the numper of the samples of the input signal
length_x = length(A);

% the data that we print on the axis x when we design the
% desired signal
axis_x=[1:1:length_x];

% a plot of input signal
plot( axis_x, A, '.');
title('input signal');

%-------------------------------------------------------------------------
% MY_QUANTIZER

% initialization of the vector that keeps the SQNRs after the
% quantization with my_quantizer
```

```
SQNR_my_quantizer = zeros(3,1);

% titles that we give to our plots
title_of_plot=['my quantizer N=2';'my quantizer N=4';'my quantizer N=6'];

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 2: 2: 6
    [xq, C, P, D] = my_quantizer(A, i, 1);
    SQNR_my_quantizer(i/2,1) = sqnr(A,D);

    % make a figure in a new window
    figure;

    % a plot for each quantized signal
    plot( axis_x, xq, '.');
    title( title_of_plot(i/2,:) );

    % play the quantized signal
    sound(xq)
end

% print the three different SQNRs for the corresponding different quantized
% signal
SQNR_my_quantizer

%-------------------------------------------------------------------------
%COMPRESSOR_M

% initialization of the vector that keeps the SQNRs of the
% quantization with compressor_m->my_quantizer->decompressor_m
SQNR_compressor_m = zeros(3,1);

% titles that we give to our plots
title_of_plot=['compressor m N=2';'compressor m N=4';'compressor m N=6'];

% the i is actually the N of the exercise, e.g. the bits that we use for
% the quantization of the signal
for i = 2: 2: 6
    xc =compressor_m(A,255);
    [xq, F, P, D] = my_quantizer(xc, i, 1);
    xd = decompressor_m(xq,255);
    D = distortion(A,xd);
    SQNR_compressor_m(i/2,1)=sqnr(A,D);

    % make a figure in a new window
    figure;

    % a plot for each quantized signal
    plot( axis_x, xd, '.');
    title( title_of_plot(i/2,:) );

    % play the quantized signal
    sound(xd)
end

% print the three different SQNRs for the corresponding different quantized
% signal
SQNR_compressor_m


%-------------------------------------------------------------------------
%COMPRESSOR_A

% initialization of the vector that keeps the SQNRs of the
```

```
% quantization with compressor_a->my_quantizer->decompressor_a
SQNR_compressor_a = zeros(3,1);

% titles that we give to our plots
title_of_plot=['compressor a N=2';'compressor a N=4';'compressor a N=6'];

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 2: 2: 6
    xc =compressor_a(A,87.6);
    [xq, F, P, D] = my_quantizer(xc, i, 1);
    xd = decompressor_a(xq,87.6);
    D = distortion(A,xd);
    SQNR_compressor_a(i/2,1)=sqnr(A,D);

    % make a figure in a new window
    figure;

    % a plot for each quantized signal
    plot( axis_x, xd, '.');
    title( title_of_plot(i/2,:) );

    % play the quantized signal
    sound(xd)
end

% print the three different SQNRs for the corresponding different quantized
% signal
SQNR_compressor_a


%-------------------------------------------------------------------------
%LLOYD_MAX

% initialization of the vector that keeps the SQNRs of after the
% quantization with Lloyd_Max
SQNR_Lloyd=zeros(3,1);

% titles that we will give to our plots
title_of_plot=['SQNR N=2';'SQNR N=4';'SQNR N=6'];

% initialization of the quantized signals
xq=zeros( length_x, 3);

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 2: 2: 6
    [xq( :, i/2 ), C, D] = Lloyd_Max(A, i, 1);

    kmax = length(D);
    loops = [1:1:kmax];

    % initialization of a vector that will keep all the SQNRs from the
    % iterations
    Sqnr = zeros(kmax,1);

    % calculate the SQNRs from the iterations
    for j = 1: 1: kmax
        Sqnr(j,1)=sqnr(A,D(j,1));
    end

    % make a figure in a new window
    figure;

    % make a plot of each of the collection of SQNRs for N=[2:2:6]
```

```
    plot(loops,Sqnr,'.-')
    title( title_of_plot(i/2,:) );
    xlabel('loops');
    ylabel('SQNR');

    % last SQNR value
    SQNR_Lloyd(i/2,1) = Sqnr(kmax,1);

    % play the quantized signal
    sound(xd)
end

% titles that we will give to our plots
title_of_plot=['Lloyd Max N=2';'Lloyd Max N=4';'Lloyd Max N=6'];

for i = 1: 1: 3
    % make a figure in a new window
    figure;

    % a plot for each quantized signal
    plot( axis_x, xq( :, i ), '.');
    title( title_of_plot(i,:) );
end

% print the three different SQNRs for the corresponding different quantized
% signal
SQNR_Lloyd
```

4 Θεωρητική μελέτη πειραματικών αποτελεσμάτων

Για την περίπτωση του ομοιόμορφου PCM και για $N = 2, 4, 6$ bits, προσπαθήσαμε να εξάγουμε κάποια επιπλέον συμπεράσματα πάνω στα πειραματικά δεδομένα κάνοντας την υπόθεση ότι η κατανομή των δειγμάτων ομιλίας μπορεί να προσεγγιστεί από την κανονική κατανομή με μέση τιμή $m = -0.04$ και διασπορά $\sigma^2 = 0.11$. Με βάση αυτή την υπόθεση, συγκρίναμε τα θεωρητικά αποτελέσματα με τα πειραματικά στις εξής κατηγορίες:

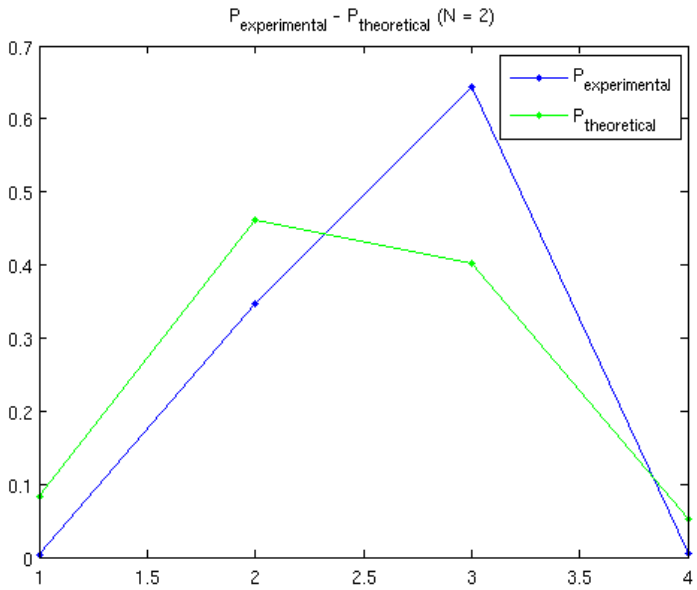
1. Πιθανότητα εμφάνισης κάθε στάθμης

2. Μέση παραμόρφωση (distortion)

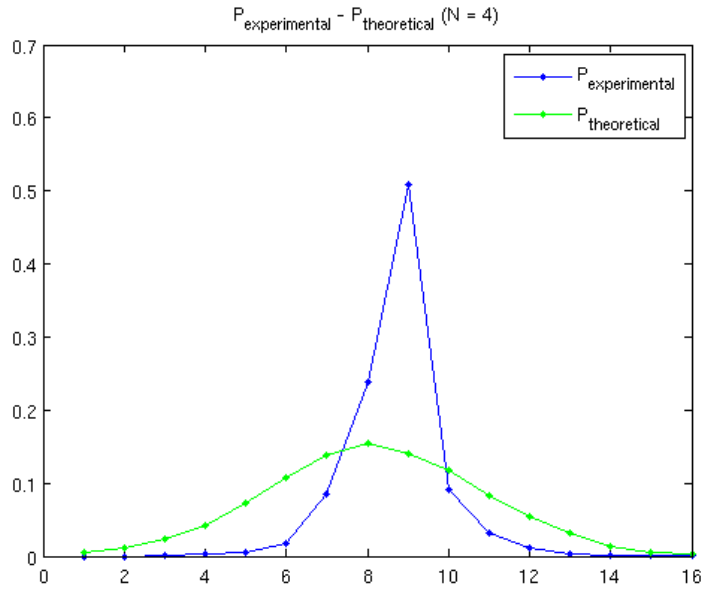
Για να βρούμε τις αντίστοιχες θεωρητικές τιμές στις δυο παραπάνω κατηγόριες, χρησιμοποιήσαμε μια γεννήτρια τυχαίων αριθμών υπό την κανονική κατανομή *randn()* που μας παρέχεται από την Matlab. Με βάση αυτή, και αφού κβαντίσαμε τις τιμές που πήραμε, υπολογίσαμε την πιθανότητα σε κάθε στάθμη να βρεθεί ένα δείγμα. Στην συνέχεια, με βάση τα δεδομένα που εξάγαμε έπειτα από την κβάντιση, δημιουργήσαμε τις αντίστοιχες γραφικές παραστάσεις για τις αντίστοιχες πιθανότητες σύμφωνα με την θεωρητική προσέγγιση και τα πραγματικά δεδομένα. Τέλος, τυπώνουμε και τους πίνακες που περιέχουν την θεωρητική και την πραγματική παραμόρφωση που υφίσταται το σήμα μας ώστε να τα συγκρίνουμε στην συνέχεια.

4.1 Αντιπαραβολή θεωρητικών και πραγματικών μετρήσεων

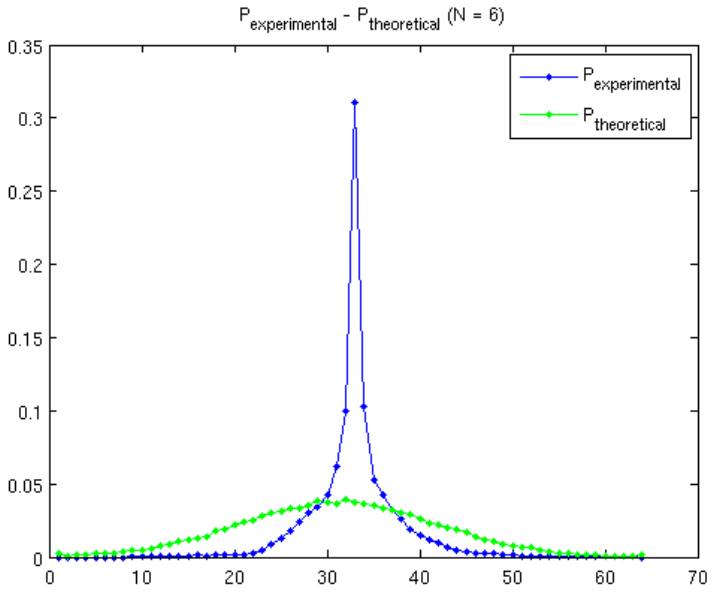
Για τα θεωρητικά αποτελέσματα χρησιμοποιήσαμε μια κανονική κατανομή με μέση τιμή $m = -0.04$ και διασπορά $\sigma^2 = 0.11$.



Σχήμα 21: Θεωρητική και πραγματική πιθανότητα κάθε στάθμης για ομοιόμορφη κβάντιση 2-bit



Σχήμα 22: Θεωρητική και πραγματική πιθανότητα κάθε στάθμης για ομοιόμορφη κβάντιση 4-bit



Σχήμα 23: Θεωρητική και πραγματική πιθανότητα κάθε στάθμης για ομοιόμορφη κβάντιση 6-bit

Παρατηρούμε από τα παραπάνω σχήματα, πως το σήμα εισόδου που είχαμε (ομιλία) προσεγγίζεται σε ικανοποιητικό βαθμό από μια κανονική κατανομή.

Όπως βλέπουμε, για περισσότερες στάθμες κβάντισης, θα χρειαζόταν να χρησιμοποιήσουμε διαφορετική κατανομή με μικρότερη διασπορά. Παρόλα αυτά και η συγκεκριμένη μας προσφέρει μια ικανοποιητική προσέγγιση.

Η πιθανότητα να βρεθεί ένα δείγμα σε μια συγκεκριμένη περιοχή κβάντισης και η παραμόρφωση σήματος υπολογίζονται απευθείας από την my_quantizer όπως την έχουμε υλοποιήσει.

Η παραμόρφωση του σήματος που έχουμε παρουσιάζεται στον Πίνακα 2.

	$N = 2$	$N = 4$	$N = 6$
Θεωρητική Παραμόρφωση Σήματος	0.0213	0.0014	0.0001
Πραγματική Παραμόρφωση Σήματος	0.0362	0.0016	0.0001

Πίνακας 2: Θεωρητική/Πραγματική παραμόρφωση σήματος

Παρατηρούμε πως οι θεωρητικές τιμές της παραμόρφωσης του σήματος που προβλέψαμε μέσω της κανονικής κατανομής με τα χαρακτηριστικά που αναφέρθηκαν προηγουμένως συγκλίνουν όπως ο αριθμός των bit κβάντισης αυξάνεται. Αυτό γίνεται γιατί δημιουργούνται περισσότερες στάθμες κβάντισης και το δείγματα του κβαντισμένου σήματος εισόδου τείνουν να προσεγγίσουν αυτά μιας κανονικής κατανομής.

4.2 Υλοποίηση σε Matlab

```
% SCRIPT FOR THE EXERCISE 3

%we print the results with long format
format long;

% we read the input signal
A=wavread('speech');

%the titles of the plot that we design
plot_titles = ['P_{experimental} - P_{theoretical} (N = 2)',
    'P_{experimental} - P_{theoretical} (N = 4)',
    'P_{experimental} - P_{theoretical} (N = 6)'];

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 2: 2: 6
    [xq, C, P, D(i/2,1)] = my_quantizer(A, i, 1);

    % theoretical mean distortion

    % normal distribution with mean value = -0.04 and standerd deviation =
    % = sqrt(0.11)
    X = -0.04 + sqrt(0.11) * randn( length(A), 1 );
    [xq_theoretical, C_theoretical, P_theoretical, D_theoretical(i/2,1)] = my_quantizer(X, i, 1);

    % make a different figure
    figure;
    % make a plot with the theoretical and experimental probabilities
    axis_x=[1:length(P)];
    plot(axis_x,P,'.-');
```

```
    hold on;
    plot(axis_x,P_theoretical,'.-g');
    title( plot_titles( i/2, : ) );
    legend('P_{experimental}','P_{theoretical}');
    hold;
end

% print the experimental and theoretical values of distortion
D
D_theoretical
```

5 Κωδικοποίηση πηγής κατά Huffman

5.1 Κώδικας Huffman

Όπως γνωρίζουμε, η εντροπία H της πηγής μας δίνει ένα άνω φράγμα για την μέγιστη συμπίεση που μπορούμε να επιτύχουμε ώστε να έχουμε επακριβή ανακατασκευή του σήματος με μηδενική πιθανότητα σφάλματος.

Στην κωδικοποίηση Huffman, που είναι κωδικοποίηση από σταθερό σε μεταβλητό μήκος, απεικονίζουμε μπλοκ σταθερού μήκους σε μεταβλητού μήκους μπλοκ δυαδικών συμβόλων. Ένα κύριο πρόβλημα σε περιπτώσεις κωδικοποίησης μεταβλητού μήκους είναι ο συγχρονισμός του δέκτη με την πηγή. Για αυτό τον λόγο απαιτούμε οι κώδικες που χρησιμοποιούμε να είναι άμεση και μονοσήμαντα αποκωδικοποιήσιμοι. Ικανή και αναγκαία συνθήκη για να ικανοποιούνται οι δυο παραπάνω απαιτήσεις σε έναν κώδικα, είναι αυτός να ικανοποιεί την συνθήκη του προθέματος. Την συνθήκη αυτή την ικανοποιεί ο κώδικας Huffman και σε συνδυασμό με την ιδιότητα του να παράγει το μικρότερο μήκος λέξεων, τον τοποθετούν στην κατηγορία των βέλτιστων κωδικοποιητών ως προς το μέσο μήκος λέξης.

Την αποδοτικότητα του αλγορίθμου την υπολογίζουμε από την εξίσωση 8 [1].

$$\eta = \frac{H(S)}{L} \tag{8}$$

Όπου η εντροπία της πηγής $H(S) = -\sum P_i \log(P_i)$ και το μέσο μήκος λέξης $L = \sum P_i L_i$. Σε περίπτωση όπου το $P_i = 0$, ορίζουμε ως $H(S_i) = 0$.

5.2 Πειραματική Αξιολόγηση

Κωδικοποιήσαμε την πηγή για $N = 4, 6$ bit μέσω της ρουτίνας Huffman. Στη συνέχεια μετρήσαμε την αποδοτικότητα του κώδικα Huffman για κάθε ένα από τα σχήματα κωδικοποίησης που αναφέραμε παραπάνω. Τα αποτελέσματα φαίνονται στον Πίνακα 3.

Όπως αναμέναμε, βλέπουμε πως ο κώδικας Huffman φθάνει σχεδόν το βέλτιστο σε απόδοση πλησιάζοντας την μονάδα.

	$N=4$	$N=6$
Ομοιόμορφο PCM	0.9912	0.9914
Συμπιεστής τύπου-μ	0.9868	0.9926
Συμπιεστής τύπου-Α	0.9877	0.9936
Μη ομοιόμορφος κβαντιστής με Lloyd-Max	0.9876	0.9942

Πίνακας 3: Αποδοτικότητα Κώδικα Huffman

5.3 Υλοποίηση σε Matlab

```
% SCRIPT FOR THE EXERCISE 2

% we print the results in long format
format long;

% we read the input signal
 A=wavread('speech');

%-------------------------------------------------------------------------
% MY_QUANTIZER

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 4: 2: 6
    [xq, C, P, D] = my_quantizer(A, i, 1);

    [encoded_A, efficiency_my_quantizer((i-2)/2,1)] = encoder(A,P,i);
end
efficiency_my_quantizer

%-------------------------------------------------------------------------
%COMPRESSOR_M

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 4: 2: 6
    xc =compressor_m(A,255);

    [xq, F, P, D] = my_quantizer(xc, i, 1);
    [encoded_A, efficiency_compressor_m((i-2)/2, 1)] = encoder(A,P,i);
end
efficiency_compressor_m

%-------------------------------------------------------------------------
%COMPRESSOR_A

% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 4: 2: 6
    xc =compressor_a(A,87.6);

    [xq, F, P, D] = my_quantizer(xc, i, 1);
    [encoded_A, efficiency_compressor_a((i-2)/2, 1) ]= encoder(A,P,i);
end
efficiency_compressor_a

%-------------------------------------------------------------------------
%LLOYD_MAX
```

```
% the i is actually the N of the exercise, the bits that we use for the
% quantization of the signal
for i = 4: 2: 6
    [xq, C, D] = Lloyd_Max(A, i, 1);

    [encoded_A, efficiency_lloyd_max((i-2)/2, 1) ]= encoder_lloyd_max(A, C, i);
end
efficiency_lloyd_max
```

6 Εργαλεία ανάπτυξης

Η συγγραφή και ανάπτυξη του κώδικα έγινε σε GNU/Linux, Debian Lenny . Χρησιμοποιήθηκαν:

- **Matlab R2008a** (7.6.0.324)
- **LaTeX** για την συγγραφή της αναφοράς.

7 Παράρτημα

7.1 Παράθεση κώδικα Matlab

7.1.1 Binary Search

```
function index = binarysearch(val, vector_of_intervals)
% Find the interval where the val belongs
% val: The value that is being searched into vector_of_intervals
% vector_of_intervals: The vector that holds the intervals
% index: The index of the corresponding interval

% find the length of the vector_of_intervals
num_of_intervals = length(vector_of_intervals);

% set the limits in where the function searches for the index in the
% vector_of_intervals
low = 1;
high = num_of_intervals + 1;

% find the expected value
index = fix((low+high)/2);
% search for the index through binary search
while (low < high) && (index~=1)
    if (val >= vector_of_intervals(index-1,1)) && (val<vector_of_intervals(index,1))
        return;
    elseif val >= vector_of_intervals(index,1)
        low = index;
    else
        high = index - 1;
    end

    index = fix((low+high)/2);

    % we have found the interval where val belongs
    if (high - low) == 1
        index = high;
```

```
        return;
    end
end

index = low;
return;
```

7.1.2 Συνάρτηση εύρεσης Παραμόρφωσης

```
function D = distortion(x, xq)
%function that calculate the destortion of the input signal x in contrast
%to the quantized signal xq

D=mean( (x - xq).^2 );
```

7.1.3 Συνάρτηση εύρεσης SQNR

```
function SQNR = sqnr(x, D)
% sqnr is a function that calculate the ratio of our signal divided from
% the noise of the quantization
% Input arguments
% x: input signal, already in discrete form
% D: the distortion
% Output arggguments
% SQNR: the "sqnr"

SQNR=mean(x.^2)/D;

return;
```

7.1.4 Κωδικοποιητής Huffman

```
function [code,len]=huffman(p);
% Huffman Coding.
%                 [code,len]=huffman(p),
%                 INPUTS
%       p(vector): contains the probabilities of each symbol
%       OUTPUTS
%       code(vector): the code for each symbol (in ascii format)
%       len(vector): the number of bits needed for each code

p = p(:)';

if length(find(p<0))~=0,
    error('Not a probability vector, negative component(s)')
end;

if abs(sum(p)-1)>10e-10,
    error('Not a probability vector, components do not add up to 1')
end;
n=length(p);
q=p;
m=zeros(n-1,n);

for i=1:n-1,
    [q,l]=sort(q);
    m(i,:)=[l(1:n-i+1),zeros(1,i-1)];
    q=[q(1)+q(2),q(3:n),1];
```

```
end;

for i=1:n-1,
    c(i,:)=blanks(n*n);
end;

c(n-1,n)='0';
c(n-1,2*n)='1';

for i=2:n-1,
    c(n-i,1:n-1)=c(n-i+1,n*(find(m(n-i+1,:)==1))-(n-2):n*(find(m(n-i+1,:)==1)));
    c(n-i,n)='0';
    c(n-i,n+1:2*n-1)=c(n-i,1:n-1);
    c(n-i,2*n)='1';
    for j=1:i-1,
        c(n-i,(j+1)*n+1:(j+2)*n)=c(n-i+1,...
            n*(find(m(n-i+1,:)==j+1)-1)+1:n*find(m(n-i+1,:)==j+1));
    end;
end;

for i=1:n,
    code(i,1:n)=c(1,n*(find(m(1,:)==i)-1)+1:find(m(1,:)==i)*n);
    len(i)=length(find(abs(code(i,:))~=32));
end;
```

7.1.5 Κωδικοποιητής

```
function [encoded_x, efficiency] = encoder(x, P, i)
% The ecoder is a function that uses the huffman algorithm in order to
% encode our quantized signal
% Input arguments:
% x: the quantized signal
% P: the probability of appearance of each level
% i: the number of bits that we use to quantize our signal
% Output arguments:
% encoded_x: the signal x encoded
% efficiency: the efficiency of algorithm huffman for the specific encoding

% length of vector x
length_x = length(x);

max_value = 1;
num_of_regions = 2^i;

% the limits of each region without the "infinities"
vector_of_intervals = zeros(num_of_regions - 1, 1);

% length of each region
% each region has the same length
length_of_region = 2 * max_value / num_of_regions;

% vector that keeps the bounds of intervals
num_of_intervals = num_of_regions - 1;
for j=1:1:num_of_intervals
    vector_of_intervals(j, 1) = - max_value + j * length_of_region;
end

% encode the quantized signal
[code,len]=huffman(P);
for j = 1: 1: length_x
    % find in which level the sample belongs
    pos = binarysearch( x(j,1), vector_of_intervals);
```

```
    % encode each of the quantized samples of the input signal
    encoded_x(j, 1) = cellstr(code(pos, :));
end

% initialization of the entropy
entropy = 0;

% the length o P vector
length_P = length(P);

%calculate the entropy
for j = 1: 1: length_P
    if P(j,1) ~= 0
        entropy = entropy - P(j, 1) * log2( P(j, 1) );
    end
end

% calculate the efficiency of the encoding
efficiency = entropy / sum(P .* len');

return;
```

7.1.6 Κωδικοποιητής Lloyd Max

```
function [encoded_x, efficiency] = encoder_lloyd_max(x, C, i)
% The ecoder is a function that uses the huffman algorithm in order to
% encode our quantized signal
% Input arguments:
% x: the quantized signal
% C: centers of quantized region
% i: the number of bits that we use to quantize our signal
% Output arguments:
% encoded_x: the signal x encoded
% efficiency: the efficiency of algorithm huffman for the specific encoding

% length of vector x
length_x = length(x);

max_value = 1;
num_of_regions = 2^i;

num_of_intervals = num_of_regions - 1;

% initialization of the vector of intervals
vector_of_intervals = zeros(num_of_intervals, 1);

% calculate the bounds of each region
for j = 1: 1: num_of_intervals
    vector_of_intervals(j, 1) = ( C(j, 1) + C( j + 1, 1) ) / 2;
end
% initialization of the vector with the occurences
occurrences = zeros(num_of_regions, 1);

for j = 1: 1: length_x
    pos = binarysearch(x(j, 1), vector_of_intervals);
    occurrences(pos, 1) = occurrences(pos, 1) + 1;
end

% initialization of the vector with the probabilities
P = zeros(num_of_regions, 1);

for j = 1: 1: num_of_regions
```

```
    P(j, 1) = occurrences(j, 1) / length_x;
end

% encode the quantized signal
[code,len]=huffman(P);
for j = 1: 1: length_x
    % find in which level the sample belongs
    pos = binarysearch( x(j,1), vector_of_intervals);
    % encode each of the quantized samples of the input signal
    encoded_x(j, 1) = cellstr(code(pos, :));
end

% initialization of the entropy
entropy = 0;

% the length o P vector
length_P = length(P);

%calculate the entropy
for j = 1: 1: length_P
    if P(j,1) ~= 0
        entropy = entropy - P(j, 1) * log2( P(j, 1) );
    end
end

% calculate the efficiency of the encoding
efficiency = entropy / sum(P .* len');

return;
```

References

[1] Peter Fenwick. Huffman code efficiencies for extensions of sources. Technical report, Department of Computer Science, The University of Auckland, February 1994.

[2] Masoud Salehi Proakis, John G. *Communication systems engineering*. Prentice Hall, 2002.