ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Ακαδημαϊκό Έτος 2016-2017
2η Εργαστηριακή Άσκηση *ΑΜ*:5673
Φακωτάκης Ορέστης

Μέρος 1

Κωδικοποίηση PCM με Μη Ομοιόμορφη Κβάντιση

Κωδικοποίηση PCM με Μη Ομοιόμορφη Κβάντιση Η PCM είναι μια μέθοδος κωδικοποίησης κυματομορφής, η οποία μετατρέπει ένα αναλογικό σήμα σε ψηφιακά δεδομένα. Τυπικά η μέθοδος PCM αποτελείται από τρία βασικά τμήματα: Έναν δειγματολήπτη, έναν κβαντιστή, και έναν κωδικοποιητή . Στα πλαίσια της άσκησης, βασικός στόχος είναι η εξοικείωση με τη λειτουργία του κβαντιστή. Συγκεκριμένα, καλείστε να υλοποιήσετε ένα μη ομοιόμορφο κβαντιστή N bits, δηλαδή 2 N επιπέδων.

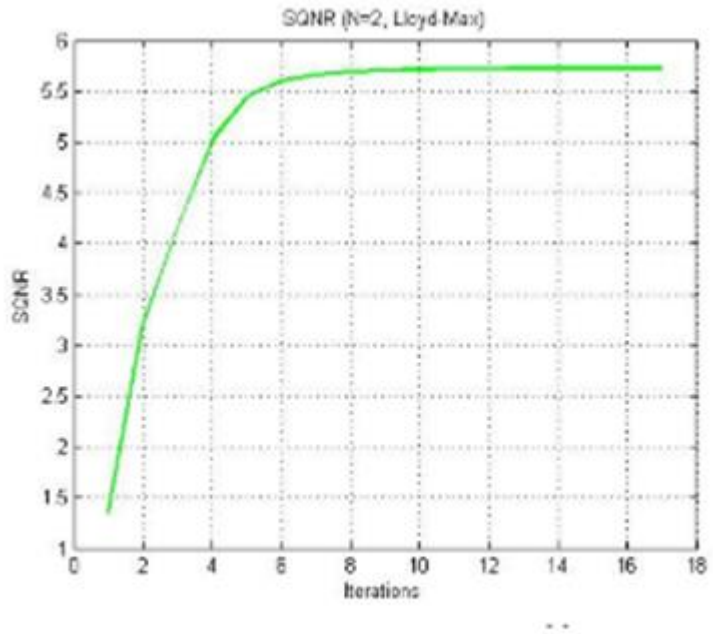
Για τη μη ομοιόμορφη κβάντιση του διανύσματος εισόδου θα χρησιμοποιηθεί ο αλγόριθμος Lloyd-Max ο οποίος επιτρέπει την σχεδίαση βέλτιστου κβαντιστή για οποιοδήποτε αριθμό επιπέδων. Καλείστε να υλοποιήσετε στη MATLAB την παρακάτω συνάρτηση: [xq, $centers$,D] = $LloydMax(x, N, xmin, xmax)$;

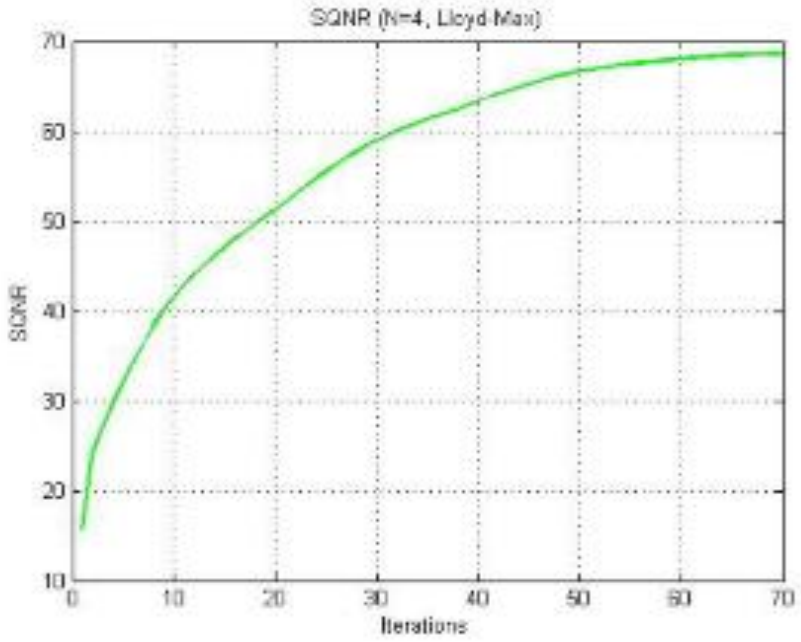
Ήχος

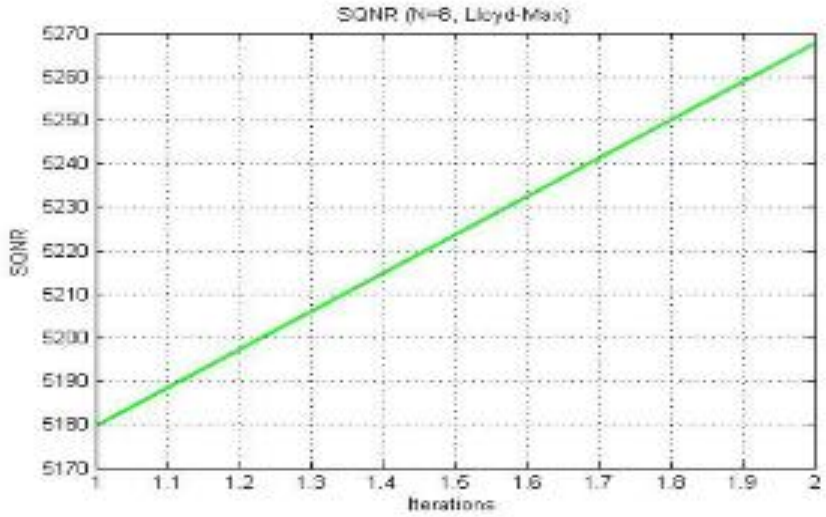
Οι τιμές του sqnr, για τις τιμές του N, του ομοιόμορφου κβαντιστή φαίνονται στον παρακάτω πίνακα:

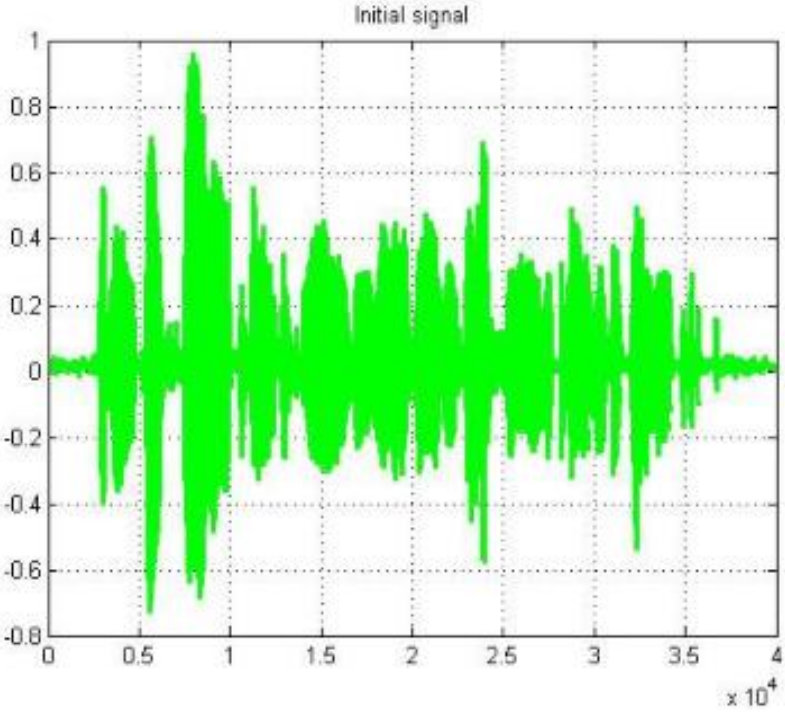
SQNR	***N=2***	***N=4***	***N=8***
Bits	1,357	15,621	5.179,563

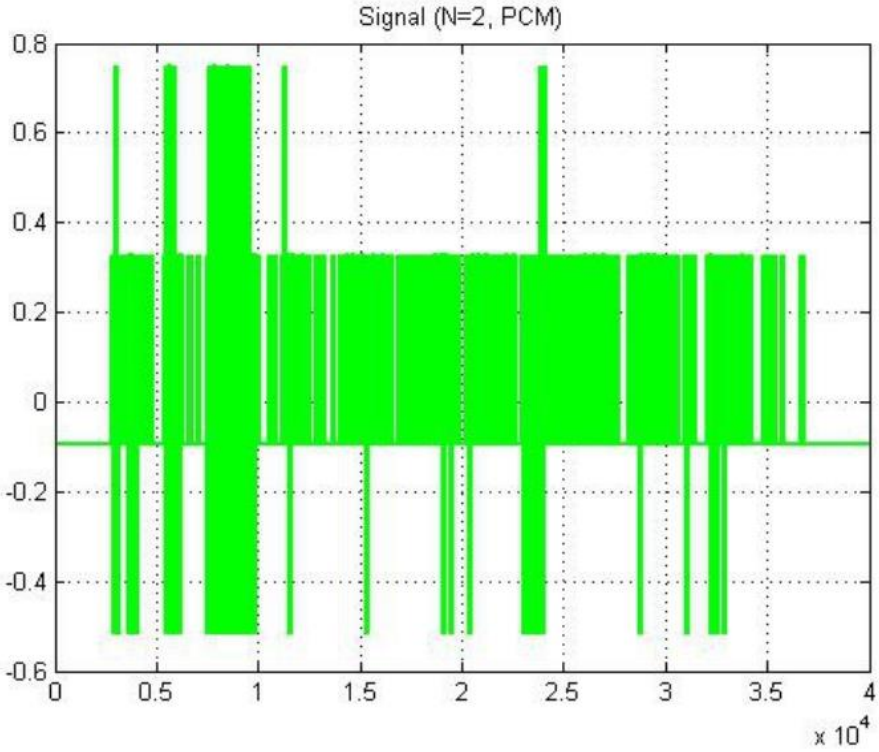
Το sqnr μας δείχνει την σχέση μεταξύ του αρχικού σήματος και της παραμόρφωσης του αρχικού σήματος. Παρατηρούμε ότι όσο μεγαλύτερο είναι το sqnr ,τόσο πιο πετυχημένη είναι κβάντιση.

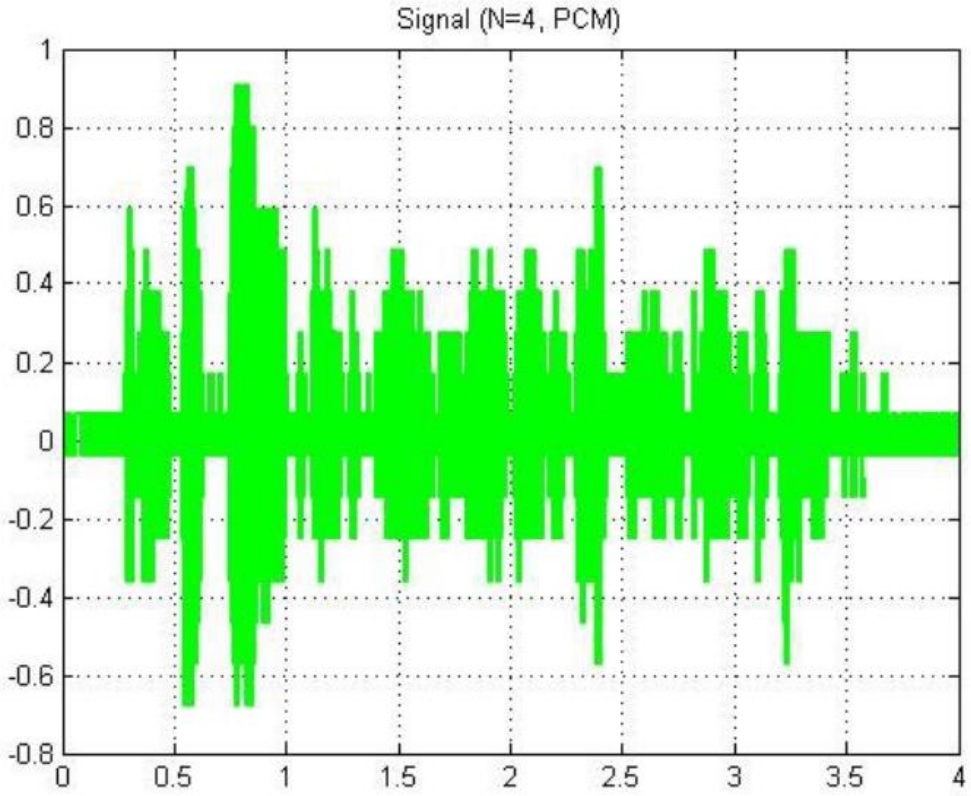

SQNR (N=2, Lloyd Max)
SQNR
Iterations
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0
2
4
6
8
10
12
14
16
18

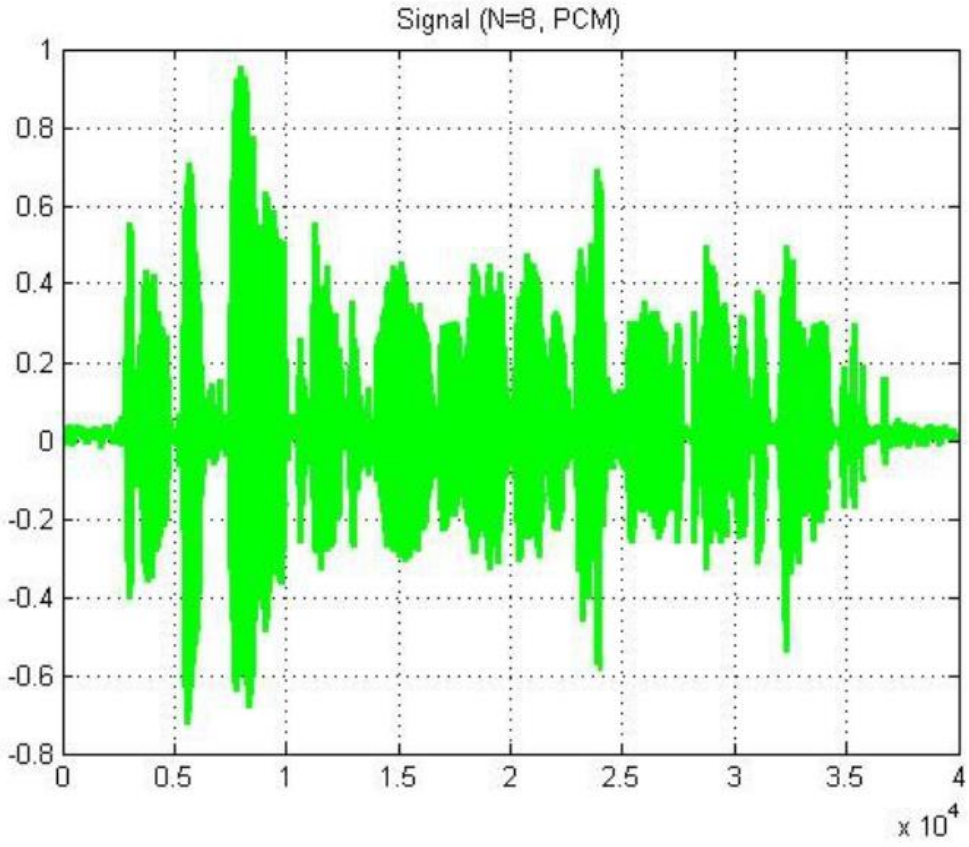

SQNR (N=4, Lloyd Max)
SQNR
Iterations
70
60
50
40
30
20
10
0
10
20
30
40
50
60
70

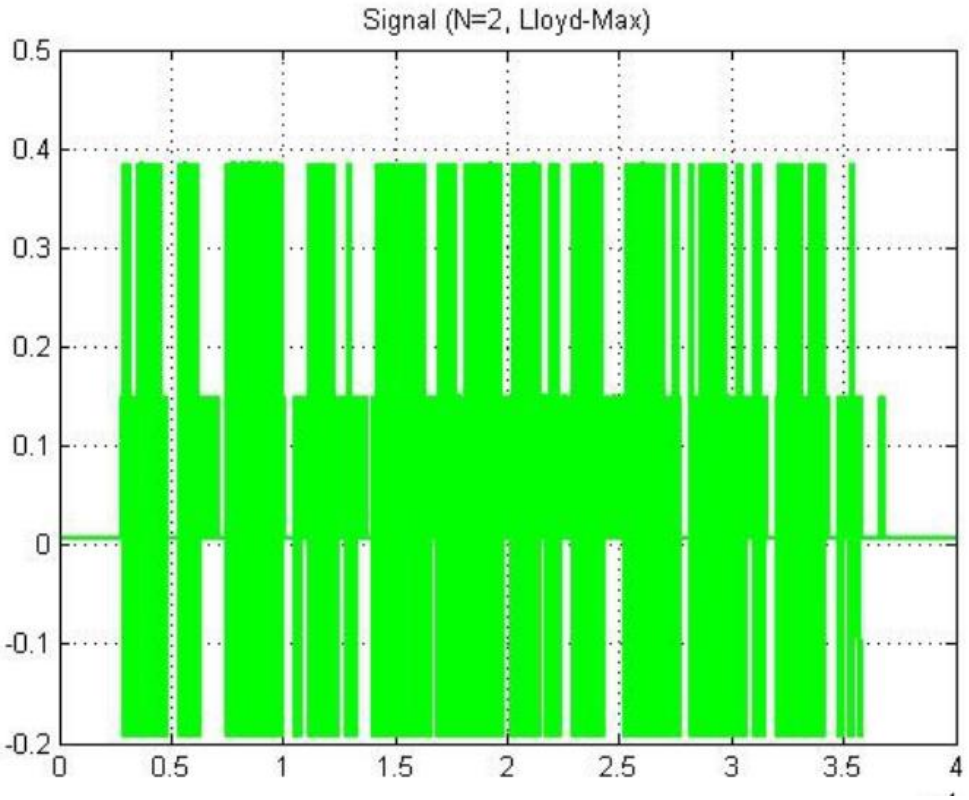

SQNR (N=6, Lloyd-Max)
SQNR
Iterations
5270
5260
5250
5240
5230
5220
5210
5200
5190
5180
5170
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2

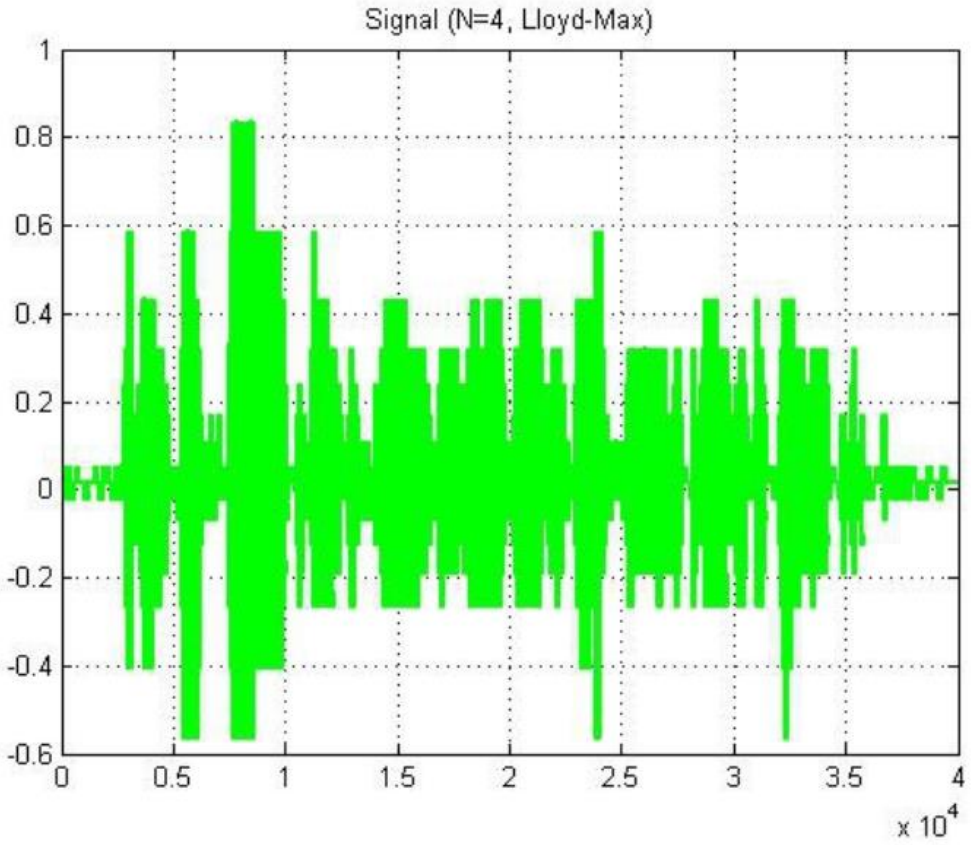

Initial signal
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4

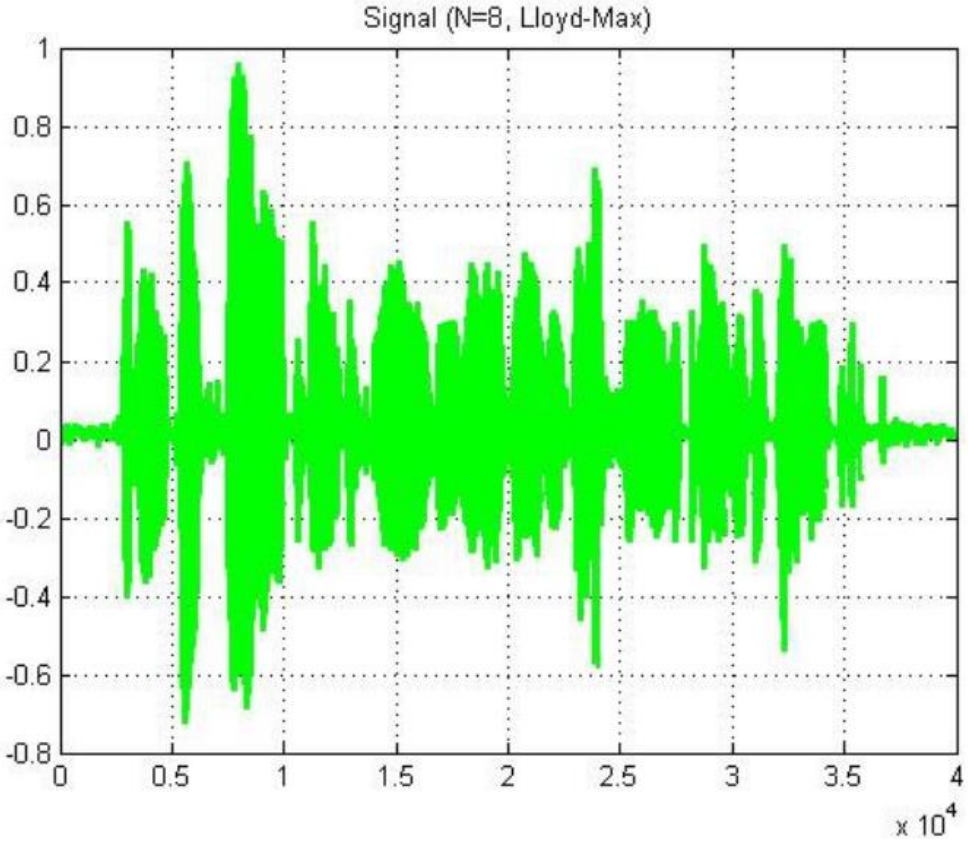

Signal (N=2, PCM)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4


Signal (N=4, PCM)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.5
1
1.5
2
2.5
3
3.5
4


Signal (N=8, PCM)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4


Signal (N=2, Lloyd-Max)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
0
0.5
1
1.5
2
2.5
3
3.5
4


Signal (N=4, Lloyd-Max)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4

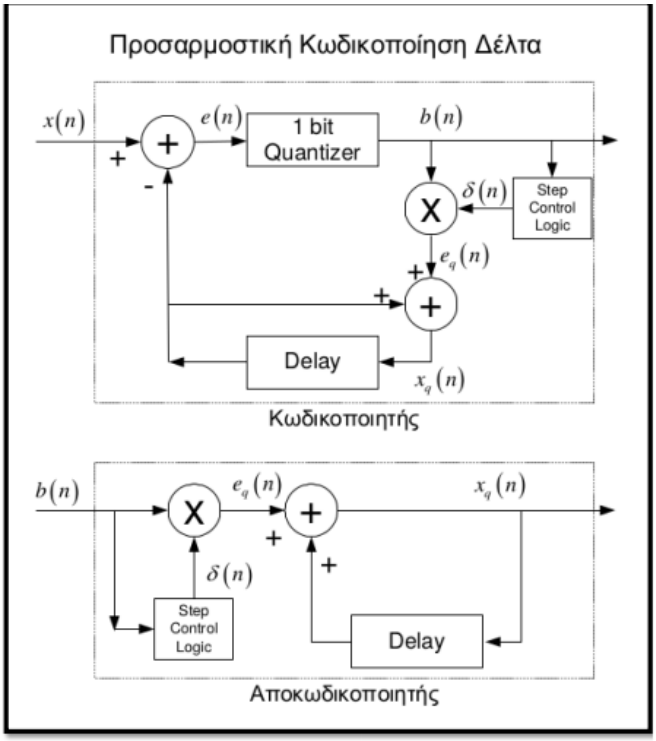

Signal (N=8, Lloyd-Max)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4

Όπως φαίνεται από τα διαγράμματα, όσο αυξάνονται τα bit που χρησιμοποιούμε στην κβάντιση τόσο περισσότερες επαναλήψεις χρειαζόμαστε για να τερματιστεί ο αλγόριθμος. Στα λίγα bits κβαντισης έχουμε μεγάλη παραμόρφωση αρά ο αλγόριθμος τερματίζει σε λίγες επαναλήψεις.

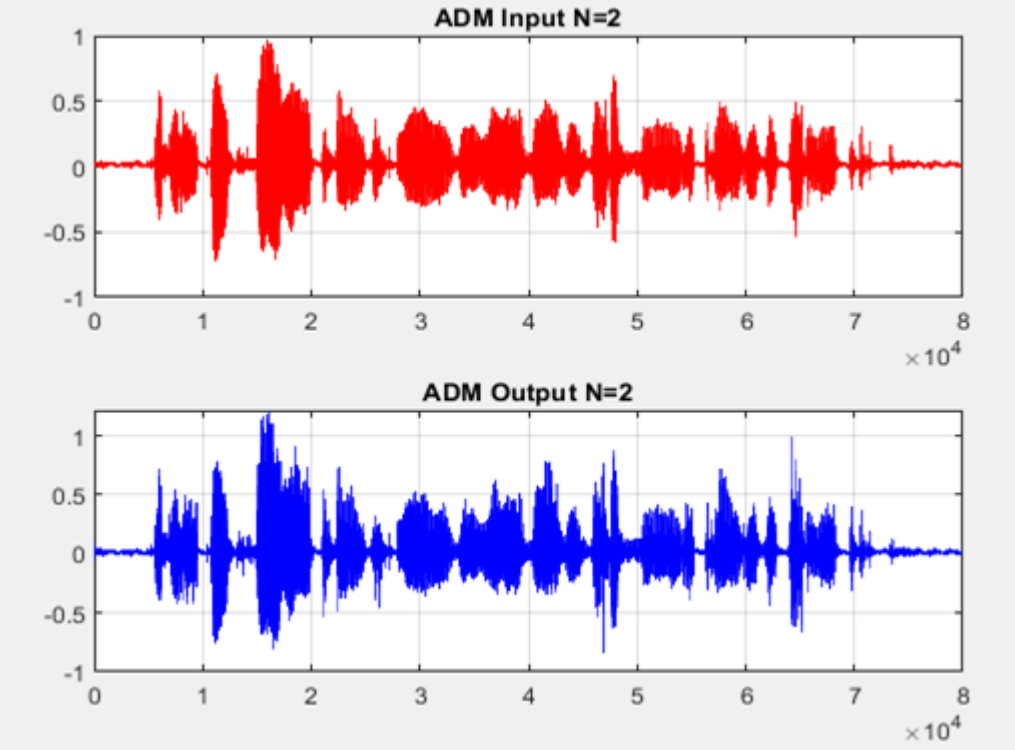
Με βάση το ηχητικό αποτέλεσμα διακρίνουμε ότι όσο αυξάνονται τα bits της κβαντισης ελαττώνεται ο θόρυβος. Όταν κβαντίζεται για N=2 είτε για N=4 το σήμα ακούγεται με αρκετό θόρυβο, ενώ για N=8 το σήμα ακούγεται παρόμοιο με το αρχικό.

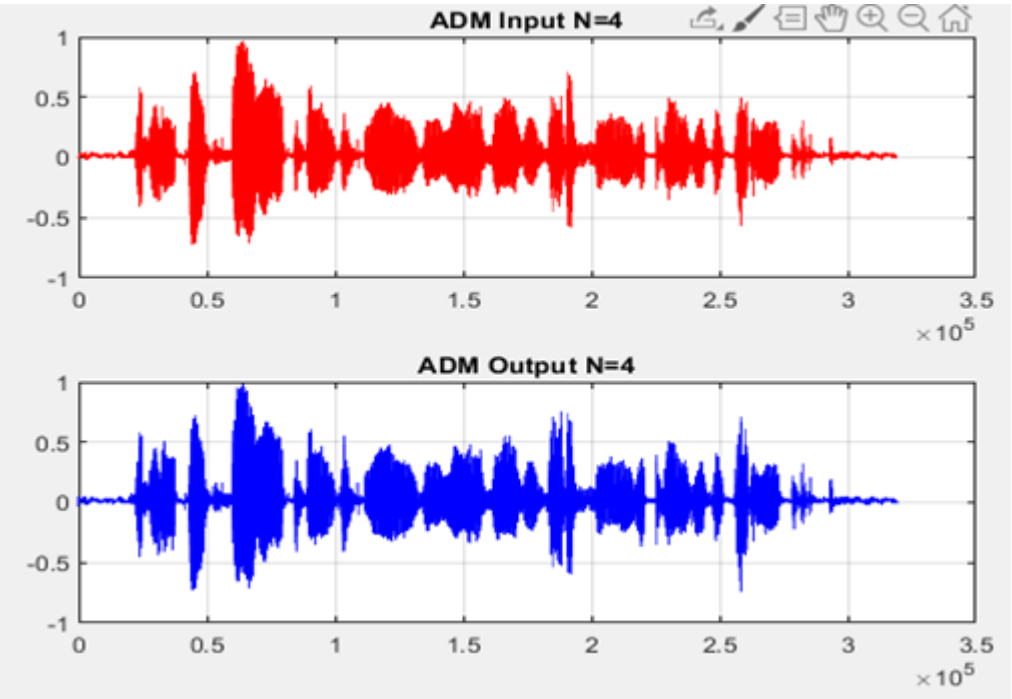
Προσαρμοστική Διαμόρφωσή Δέλτα (ADM)

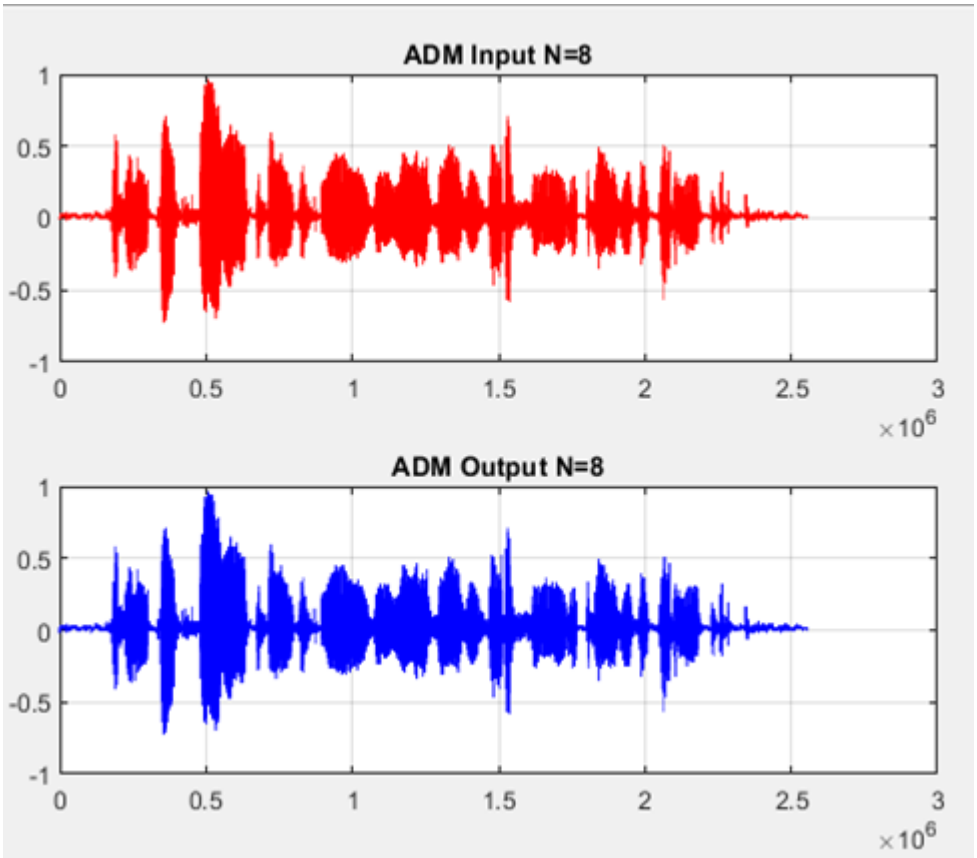
Η ADM χρησιμοποιεί μεταβαλλόμενο βήμα. Συγκεκριμένα, σε περιοχές που η κυματομορφή του σήματος εμφανίζει απότομη κλήση, η ADM αυξαίνει το βήμα. Αντίθετα, σε περιοχές σταθερής τιμής του σήματος, το βήμα μικραίνει, ώστε να αποφευχθεί ο κοκκώδης θόρυβος.



Παρακάτω παρουσιάζουμε τα διαγράμματα μεταξύ του ADM INPUT και του ADM output για N=2,4,8:


ADM Input N=2
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
×10^4
ADM Output N=2
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
×10^4


ADM Input N=4
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
2.5
3
3.5
×10^5
ADM Output N=4
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
2.5
3
3.5
×10^5



Εικόνα

Η πηγή Β είναι τα εικονοστοιχεία (pixels) μιας grayscale εικόνας. Μια τέτοια εικόνα μπορεί να θεωρηθεί ως ένας πίνακας με εικονοστοιχεία, όπου κάθε εικονοστοιχείο αντιστοιχεί σε ένα byte πληροφορίας. Κάθε εικονοστοιχείο επομένως λαμβάνει μια τιμή στο δυναμικό εύρος [0:255] η οποία αντιστοιχεί σε ένα από τα 256 επίπεδα φωτεινότητας (το 0 αντιστοιχεί στο μαύρο και το 255 στο λευκό). Στο αρχείο cameraman.mat περιέχεται ένα σήμα εικόνας.

Χρησιμοποιώντας τον κβαντιστή που υλοποιήσαμε προηγούμενος κωδικοποιούμε την πηγή Β (αρχείο εικόνας) για N=2,4.

bits	N=2	N=4
sqnr	23.88	308,56

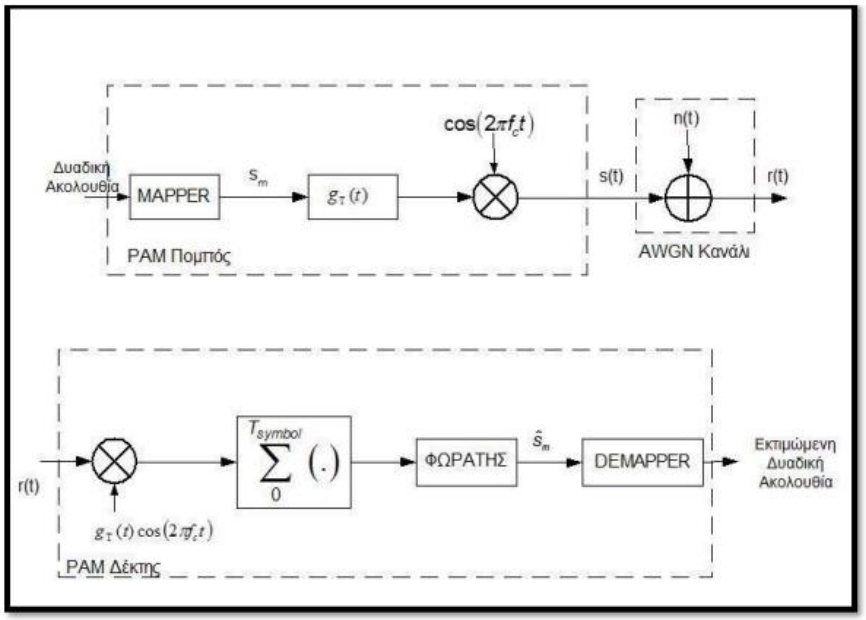
Η αρχική εικόνα ήταν:

Με N=2 έχουμε το παρακάτω αποτέλεσμα

Ενώ για N=4 παίρνουμε:

Μέρος 2

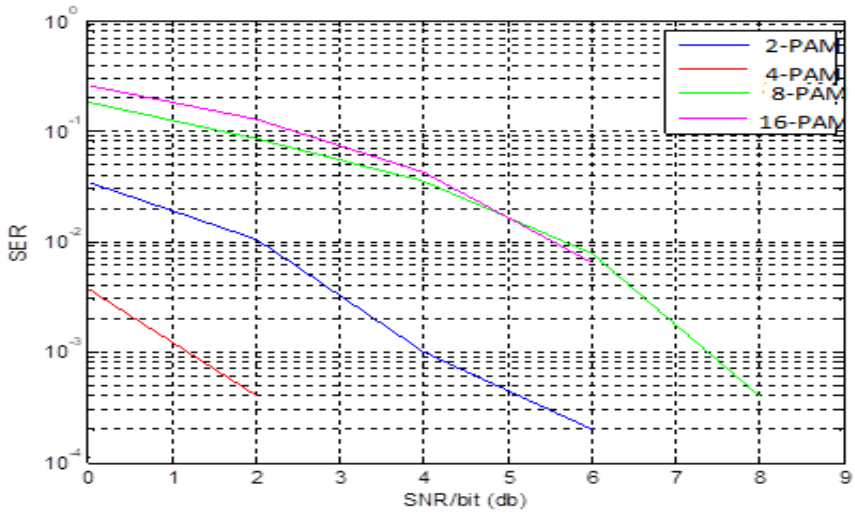
Μελετάμε την διαμόρφωση M-PAM και ως προς την απόδοσή τους, για M=2 και M=4 και M=8. Η σύγκριση θα βασιστεί στην πιθανότητα σφάλματος συμβόλου (SER) και bit (BER), που θα πραγματοποιηθούν σε ομόδυνα ζωνοπερατά συστήματα με ορθογώνιο παλμό. M-PAM Στο παρακάτω σχήμα παρουσιάζονται τα βασικά μέρη του πομπού και του δέκτη που χρησιμοποιούνται στην M-PAM διαμόρφωση:



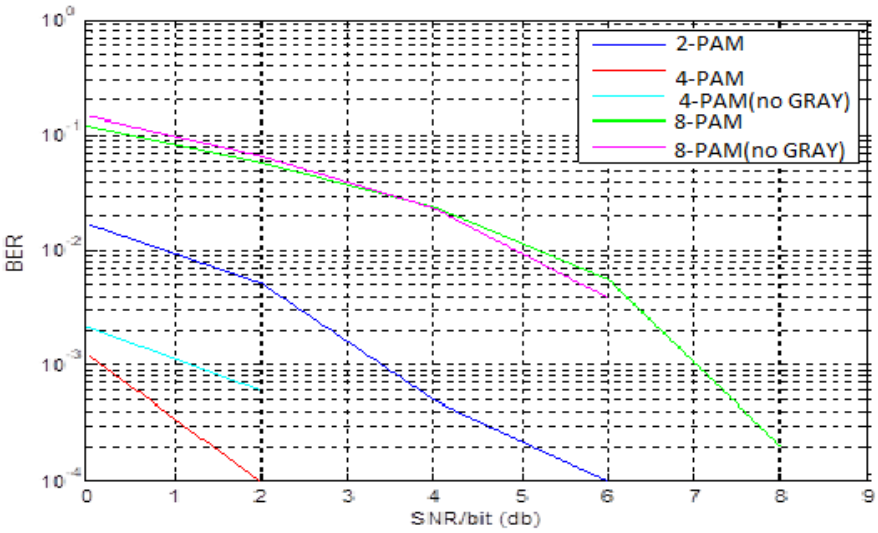
Κατασκευάζουμε μια ακολουθία από δυαδικά ψηφία. Αυτή την ακολουθία πλέον απαιτείται να αντιστοιχίσουμε σε σύμβολα. Μετατρέπουμε μια αλληλουχία από δυαδικά ψηφία, στην αντίστοιχη δεκαδική της ποσότητα. Στην συνέχεια ακολουθεί η μορφοποίηση του σήματος χωρίς θόρυβο κατά M-PAM όπου M ∈2,4,8 . Δημιουργούμε τη συνάρτηση make_signal, και επιλέγουμε PAM διαμόρφωση. Η περίοδος δειγματοληψίας είναι 1, και αυτό σημαίνει πως για την κωδικοποίηση ενός συμβόλου χρειάζονται TS = 40 δείγματα. Αυτά παράγονται εφαρμόζοντας τη σχέση που δίνεται στην εκφώνηση για κατάλληλες τιμές. Τελικά στο μορφοποιημένο σήμα προστίθεται μια πηγή λευκού θορύβου η οποία εξαρτάται από το SNR. Όσο μικρότερο είναι αυτό τόσο περισσότερος θόρυβος προστίθεται.. Τελικά παράγεται το μεταδιδόμενο σήμα το οποίο γίνεται είσοδος στην αποκωδικοποίηση, Για να μετρήσουμε τα σφάλματα μετάδοσης bit και συμβόλων πρέπει να εφαρμόσουμε την ανάστροφη διαδικασία που ακολουθήσαμε για την κωδικοποίηση της ακολουθίας. μετατρέπουμε το σήμα που λάβαμε σε προσεγγίσεις συμβόλων, αθροίζοντας όλα τα TS δείγματα που το απαρτίζουν σύμφωνα με την σχέση που δίνεται στις οδηγίες. Παίρνοντας τις προσεγγίσεις συμβόλων που προκύπτουν τις τροφοδοτούμε στην unmod_symbols η οποία αποκόπτει-στρογγυλοποιεί τις τιμές αυτές σε ακέραιες ποσότητες στο διάστημα που περιορίζονται τα σύμβολα ανάλογα του M. Τελικά μετατρέπουμε κάθε σύμβολο σε δυαδικά ψηφία και ανακατασκευάζουμε την ακολουθία

δυαδικών ψηφίων. Βρίσκουμε σε πόσα σημεία διαφέρουν οι ακολουθίες συμβόλων και δυαδικών ψηφίων και υπολογίζουμε τα SER (Symbols Error Rate) και BER (Bits Error Rate).

Λαμβάνουμε τις τιμές των SER και BER, για M=2, M=4,M=8 με κωδικοποίηση Gray ή χωρίς κωδικοποίηση Gray για τις τιμές του SNR στο διάστημα [0:2:20]. Οι γραφικές παραστάσεις του SER που πήραμε είναι οι εξής.



Σχεδιάζουμε τις καμπύλες της πιθανότητας σφάλματος bit (BER) για τιμές του SNR από 0 μέχρι 20 db χρησιμοποιώντας βήμα 2 db. Για τον υπολογισμό του δείκτη BER αρκεί, αντίστοιχα με πριν, να μετρήσουμε πόσα bits δεν στάλθηκαν σωστά και να διαιρέσουμε με το συνολικό αριθμό των bits.



Κώδικες

```
function [xq,centers] = my_quantizer(x,N,min_value,max_value)
v = N;
quant_levels = 2^v;
xq = zeros(length(x),1);
quant_step = (abs(min_value)+max_value)/quant_levels;
centers = zeros(quant_levels,1);
for i=1:quant_levels
   centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
end
q_max_value = 1; q_min_value = quant_levels;
for i=1:length(x)
   if (x(i) >= max_value), xq(i) = q_max_value;
   elseif (x(i) < min_value), xq(i) = q_min_value;
   else
      for n=1:quant_levels
         if (x(i) >= (centers(n)-(quant_step/2)) &&
            x(i) < (centers(n)+(quant_step/2)))
            xq(i) = n;
         end
      end
   end
end
```

```
function [xq,centers,D] = Lloyd_Max(x,N,min_value,max_value)
v = N;

quant_levels = 2^v;

xq = zeros(length(x),1);

quant_step = (abs(min_value)+max_value)/quant_levels;

centers = zeros(quant_levels,1);
for i=1:quant_levels
   centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
end

T = zeros((quant_levels+1),1);
counter = 1;
previous_distortion = 0;
while 1

   T(1) = max_value;
   for i=2:quant_levels
      T(i) = (centers(i)+centers(i-1))/2;
   end
   T(quant_levels+1) = min_value;

   q_max_value = 1; q_min_value = quant_levels;
   for i=1:length(x)

      if (x(i) >= max_value)
         xq(i) = q_max_value;

      elseif (x(i) <= min_value)
         xq(i) = q_min_value;
      else %
         for n=1:quant_levels
            if ((x(i) <= T(n)) && (x(i) > T(n+1)))
               xq(i) = n;
            end
         end
      end
   end

   if (all(xq))
      D(counter) = mean((x-centers(xq)).^2);
   else
```

```
    fprintf('Error: there are zeros as index.\n');
  end


  difference = abs(D(counter)-previous_distortion);
  if (difference < eps('single'))
    break;
  else % Else, store current distortion for the next comparison.
    previous_distortion = D(counter);
  end


  temp_sum = zeros(quant_levels,1);
  temp_counter = zeros(quant_levels,1);
  for n=1:quant_levels
    for i=1:length(x)

      if (x(i) <= T(n) && x(i) > T(n+1))
        temp_sum(n) = temp_sum(n) + x(i);
        temp_counter(n) = temp_counter(n) + 1;

      elseif ((x(i) > T(n)) && (n == 1))
        temp_sum(n) = temp_sum(n) + T(n);
        temp_counter(n) = temp_counter(n) + 1;

      elseif ((x(i) < T(n+1)) && (n == quant_levels))
        temp_sum(n) = temp_sum(n) + T(n+1);
        temp_counter(n) = temp_counter(n) + 1;
      end
    end

    if (temp_counter(n) > 0)
      centers(n) = temp_sum(n)/temp_counter(n);
    end;
  end


  counter = counter + 1;
end

fprintf('(%d bits) Kmax = %d\n',N,counter);
```

Φόρτωση Εικόνας

```
M=256;
N=256;
load cameraman.mat
figure;
imshow(uint8(i));
x=i(:);
x=(x-128)/128;
length_x = length(x);
title_of_plot=['SQNR N=2';'SQNR N=4'];
SQNR_Lloyd=zeros(2,1);
xq=zeros( length_x, 2);
for i= 2:2:4
[xq, C, D] = Lloyd_Max(x, i, -1, 1);
x1=128*xq+128;
M=256;
N=256;
output = reshape(x1,M,N);
figure;
imshow(uint8(output));
kmax = length(D);
loops = (1:1:kmax);
Sqnr = zeros(kmax,1);
for j = 1: 1: kmax
Sqnr(j,1)=sqnr(A,D(j,1));
end
figure;
plot(loops,Sqnr,'.-')
title( title_of_plot(i/2,:) );
xlabel('loops');
ylabel('SQNR');
% last SQNR value
SQNR_Lloyd(i/2,1) = Sqnr(kmax,1);
end
```

Φόρτωση ήχου

```
[fs,N]=audioread('speech.wav');
length_x = length(A);
axis_x=[1:1:length_x];
plot( axis_x, A, '.');
title('input signal');
SQNR_Lloyd=zeros(3,1);
title_of_plot=['SQNR N=2';'SQNR N=4';'SQNR N=8'];
```

```
xq=zeros( length_x, 3);

c=1;
for i = 2: 2:6
   if i==6
      i=8;
   end

[xq( :, c ), C, D] = Lloyd_Max(A, i,-1, 1);
kmax = length(D);
loops = (1:1:kmax);
Sqnr = zeros(kmax,1);
for j = 1: 1: kmax
Sqnr(j,1)=sqnr(A,D(j,1));
end
figure;
plot(loops,Sqnr,'.-')
title( title_of_plot(c,:) );
xlabel('loops');
ylabel('SQNR');
SQNR_Lloyd(c,1) = Sqnr(kmax,1);
wavplay(A,fs);
disp(i)
disp(c)
c=c+1;
end
title_of_plot=['Lloyd Max N=2';'Lloyd Max N=4';'Lloyd Max N=8'];
for i = 1: 1: 3
figure;
plot( axis_x, xq( :, i ), '.');
title( title_of_plot(i,:) );
end
```

```
[xq2_pcm,centers2_pcm] = my_quantizer(y,2,min(y),max(y));

[xq4_pcm,centers4_pcm] = my_quantizer(y,4,min(y),max(y));

[xq8_pcm,centers8_pcm] = my_quantizer(y,8,min(y),max(y));

[xq2_lm,centers2_lm,D2_lm] = Lloyd_Max(y,2,min(y),max(y));
```

```
[xq4_lm,centers4_lm,D4_lm] = Lloyd_Max(y,4,min(y),max(y));
[xq8_lm,centers8_lm,D8_lm] = Lloyd_Max(y,8,min(y),max(y));


% Calculating SQNR for PCM and plotting it for Lloyd-Max.
SQNR2_pcm = mean(y.^2)/mean((y-centers2_pcm(xq2_pcm)).^2);
fprintf('SQNR2_pcm: %f\n',SQNR2_pcm);
SQNR4_pcm = mean(y.^2)/mean((y-centers4_pcm(xq4_pcm)).^2);
fprintf('SQNR4_pcm: %f\n',SQNR4_pcm);
SQNR8_pcm = mean(y.^2)/mean((y-centers8_pcm(xq8_pcm)).^2);
fprintf('SQNR8_pcm: %f\n',SQNR8_pcm);
SQNR2_lm = mean(y.^2)./D2_lm;
figure; plot(SQNR2_lm,'g','LineWidth',2); title('SQNR
(N=2, Lloyd-Max)');
xlabel('Iterations'); ylabel('SQNR'); grid on;
SQNR4_lm = mean(y.^2)./D4_lm;
figure; plot(SQNR4_lm,'g','LineWidth',2); title('SQNR
(N=4, Lloyd-Max)');
xlabel('Iterations'); ylabel('SQNR'); grid on;
SQNR8_lm = mean(y.^2)./D8_lm;
figure; plot(SQNR8_lm,'g','LineWidth',2); title('SQNR
(N=8, Lloyd-Max)');
xlabel('Iterations'); ylabel('SQNR'); grid on;


% Saving audio objects for each quantization case.
initial_track = audioplayer(y,fs);
track2_pcm = audioplayer(centers2_pcm(xq2_pcm),fs);
track4_pcm = audioplayer(centers4_pcm(xq4_pcm),fs);
track8_pcm = audioplayer(centers8_pcm(xq8_pcm),fs);
track2_lm = audioplayer(centers2_lm(xq2_lm),fs);
track4_lm = audioplayer(centers4_lm(xq4_lm),fs);
track8_lm = audioplayer(centers8_lm(xq8_lm),fs);


% Plotting all signals.
figure; plot(y,'g','LineWidth',2); title('Initial
signal'); grid on;
```

```
figure; plot(centers2_pcm(xq2_pcm),'g','LineWidth',2);
title('Signal (N=2, PCM)'); grid on;
figure; plot(centers4_pcm(xq4_pcm),'g','LineWidth',2);
title('Signal (N=4, PCM)'); grid on;
figure; plot(centers8_pcm(xq8_pcm),'g','LineWidth',2);
title('Signal (N=8, PCM)'); grid on;
figure; plot(centers2_lm(xq2_lm),'g','LineWidth',2);
title('Signal (N=2, Lloyd-Max)'); grid on;
figure; plot(centers4_lm(xq4_lm),'g','LineWidth',2);
title('Signal (N=4, Lloyd-Max)'); grid on;
figure; plot(centers8_lm(xq8_lm),'g','LineWidth',2);
title('Signal (N=8, Lloyd-Max)'); grid on;
```

Β ΜΕΡΟΣ

metr.m

```
N_BITS = 100000;
[ber_nogray, ser_nogray] = run_pam(N_BITS, 0);
[ber_gray, ser_gray] = run_pam(N_BITS, 1);
f = figure;
x = [0:2:12]';
semilogy(x', ber_nogray(1,:), 'g-');
hold on;
grid on;
semilogy(x', ber_nogray(2,:), 'b-');
semilogy(x', ber_gray(2,:), 'm-');
title('Bits error rate');
xlabel('SNR'); ylabel('BER');
legend('2-pam no-gray', '4-pam no-gray', '2-pam gray', '4-pam
gray',8-pam );
legend('2-pam no-gray', '4-pam no-gray', '4-pam gray',8-pam);
hold;
% Grafike SER vs SNR
f = figure;
semilogy(x', ser_nogray(1,:), 'g-');
hold on;
grid on;
semilogy(x', ser_nogray(2,:), 'b-');
title('Symbols error rate');
xlabel('SNR'); ylabel('SER');
legend('2-pam', '4-pam', '8-pam', '16-pam');
```

hol**d;**

run_pam.m

```
function [ber, ser] = run_pam(bits, gray)
 i = 1;
 fprintf('\n\nPPM\tSNR\tGray\tBER\tSER\n');
 fprintf('---\t---\t----\t---\t---\n');
 for M=2:2:4
 j = 1;
 for SNR=0:2:12
 bit_stream = randsrc(bits, 1, [0,1]);
 sym = make_symbols(bit_stream, M, gray, 'pam');
 sig = make_signal(sym, M, 'pam');
 sig = add_gwn_noise(sig, M, SNR);
 rec_sig = unmod_signal(sig, M, 'pam');
 rec_sym = unmod_symbols(rec_sig, M, 'pam');
 rec_bit = make_bits(rec_sym, M, gray, 'pam');
 ber(i, j) = sum(bit_stream ~= rec_bit) / bits;
 ser(i, j) = sum(sym ~= rec_sym)/ size(sym,
gstr = 'false';
 if gray == 1
 gstr = 'true';
 end
 fprintf('%d\t%d\t%s\t%g\t%g\n', M, SNR, gstr,ber(i, j), ser(i,j));
 j = j + 1;
 end
 i = i + 1;
 end
end
```

make_symbols.m

```
 function [symbols] = make_symbols(bit_stream, M, gray,
modtype)
k = 1;
for i=1:M/2:length(bit_stream)
symbols(k, 1) = 0;
for b=1:M/2
symbols(k, 1) = symbols(k, 1) + bit_stream(i+b-1)*2^(M/2-b);
 end k = k + 1;
end if (gray == 1)
symbols = bin2gray(symbols, modtype, M);
end
end
```

add_gwn_noise.m

```
 function [sig] = add_gwn_noise(signal, M, SNR)
EB = 2/M;
N0 = EB * 10^((-SNR)/10);
```

```
DS = sqrt(N0/2);
sig = signal + DS*randn(size(signal));
end
```

unmod_symbols.m

```
function [rec_sym] = unmod_symbols(rec_sig, M)
 N = size(rec_sig, 1);

 rec_sym = round(1/2*(M+rec_sig-1));
 else
 rec_sym = zeros(N,1);
 for s=1:N
 rec_sym(s,1) = 0;
 max = rec_sig(s,1);
 for m=1:M
 if (rec_sig(s,m) > max)
 max = rec_sig(s,m);
 rec_sym(s,1) = m-1;
 end
 end
 end
 end
 rec_sym(rec_sym<0) = 0;
 rec_sym(rec_sym>M-1) = M-1;
end
```

make_bits.m

```
function [rec_bit] = make_bits(rec_sym, M, gray)
 N = length(rec_sym);
 rec_bit = zeros(N*M/2,1);
 if gray == 1
 rec_sym = gray2bin(rec_sym, M);
 end
 bit = 1;
 for i=1:N
 for b=1:M/2
 rec_bit(bit) = bitget(rec_sym(i), M/2-b+1);
 bit = bit + 1;
 end
 end
end
```