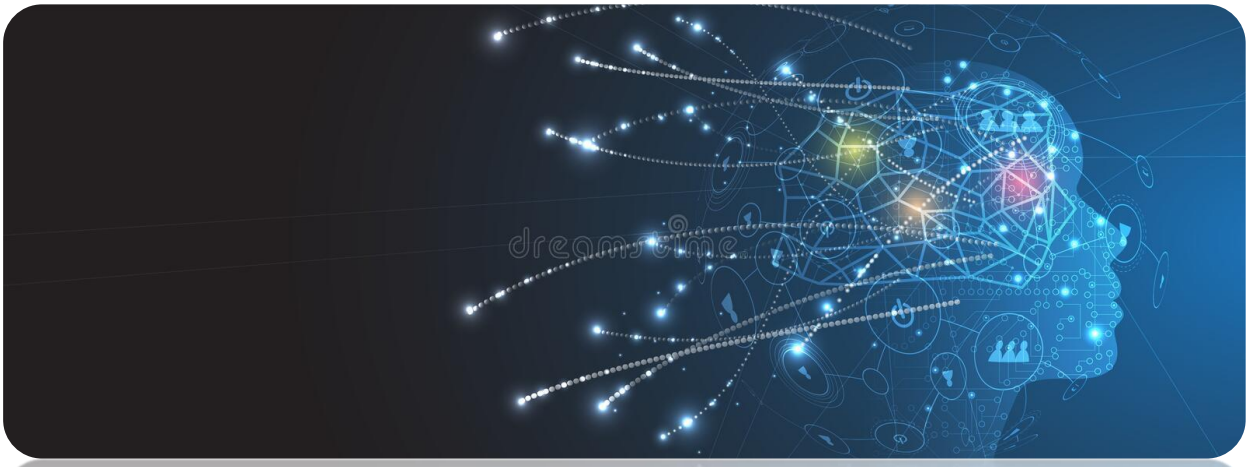
Ψηφιακές Τηλεπικοινωνίες

Πρώτο Σετ Ασκήσεων (2022-2023)



Ονοματεπώνυμο: Άγκο Μπεσιάνα

ΑΜ: 1059662

Έτος: 6ο

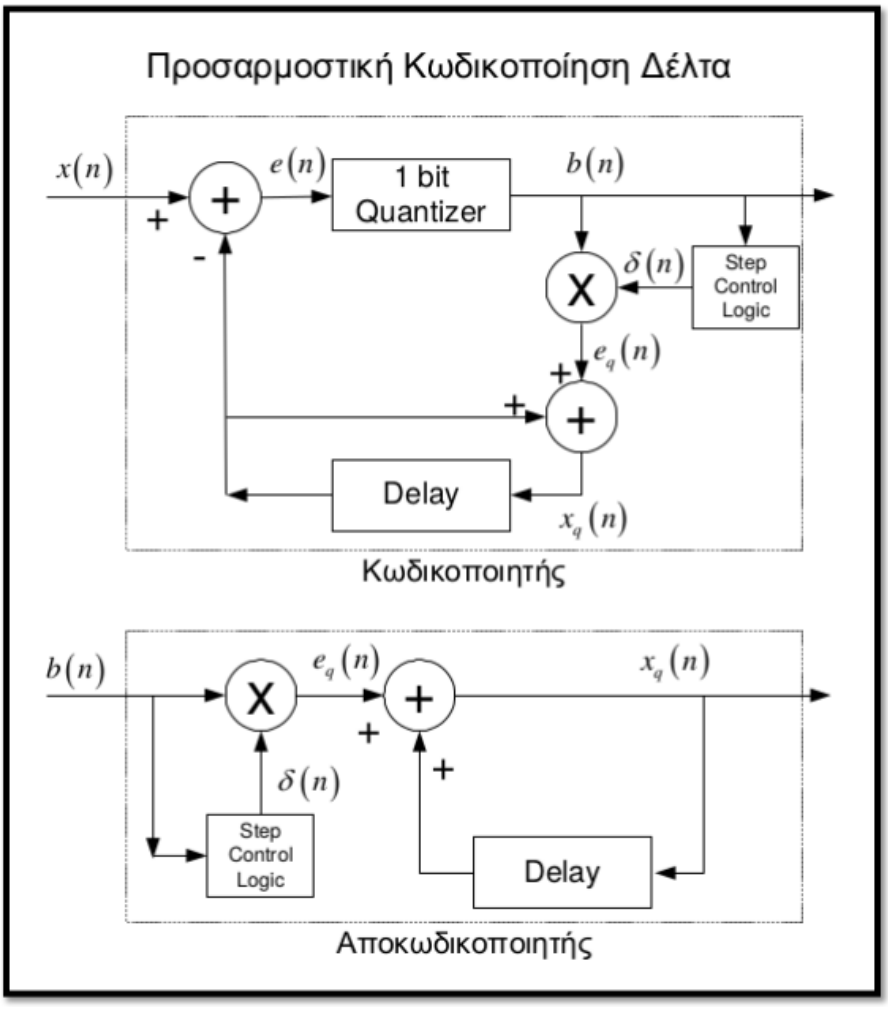
Καθηγητής: Κ. Μπερμπερίδης

Περιεχόμενα

Μέρος 1:

Στο πρώτο μέρος μας ζητήθηκε να υλοποιήσουμε έναν παλμοκωδικό διαμορφωτή (PCM) με μη ομοιόμορφη κβάντιση. Η PCM είναι μια μέθοδος κωδικοποίησης κυματομορφής, η οποία μετατρέπει ένα αναλογικό σήμα σε ψηφιακά δεδομένα. Έχει 3 βασικά μέρη: Δειγματολήπτη, Κβαντιστή και Κωδικοποιητή. Για την άσκηση υλοποιήθηκε ένας μη ομοιόμορφος κβαντιστής N bits, δηλαδή 2^N επιπέδων.

Επίσης, κληθήκαμε να υλοποιήσουμε προσαρμοστική διαμόρφωση Δέλτα (ADM). Η DM είναι μια απλοποιημένη μορφή της DPCM όπου ο κβαντιστής διαθέτει 2 στάθμες κβάντισης. Η μονή διαφορά της ADM από την DM είναι ότι η ADM χρησιμοποιεί μεταβαλλόμενο βήμα. Συγκεκριμένα, σε περιοχές που η κυματομορφή του σήματος εμφανίζει απότομη κλήση, η ADM αυξάνει το βήμα. Αντίθετα, σε περιοχές σταθερής τιμής του σήματος, το βήμα μικραίνει, ώστε να αποφευχθεί ο κοκκώδης θόρυβος.



Εικόνα 1: ADM

Ερώτημα 1:

Για τις μετρήσεις μας πρέπει να χρησιμοποιήσουμε 2 πηγές. Η πρώτη είναι μια AR(1) τυχαία διαδικασία πρώτης τάξης η οποία περιγράφεται από την εξίσωση διαφορών :

$$y(n) = a_1 y(n-1) + x(n)$$

Η πηγή Β ήταν τα εικονοστοιχείο (pixels) μιας grayscale εικόνας. Όπου κάθε εικονοστοιχείο λαμβάνει μια τιμή στο δυναμικό εύρος [0:255] η οποία αντιστοιχεί σε ένα επίπεδο από τα 256 επίπεδα φωτεινότητας.

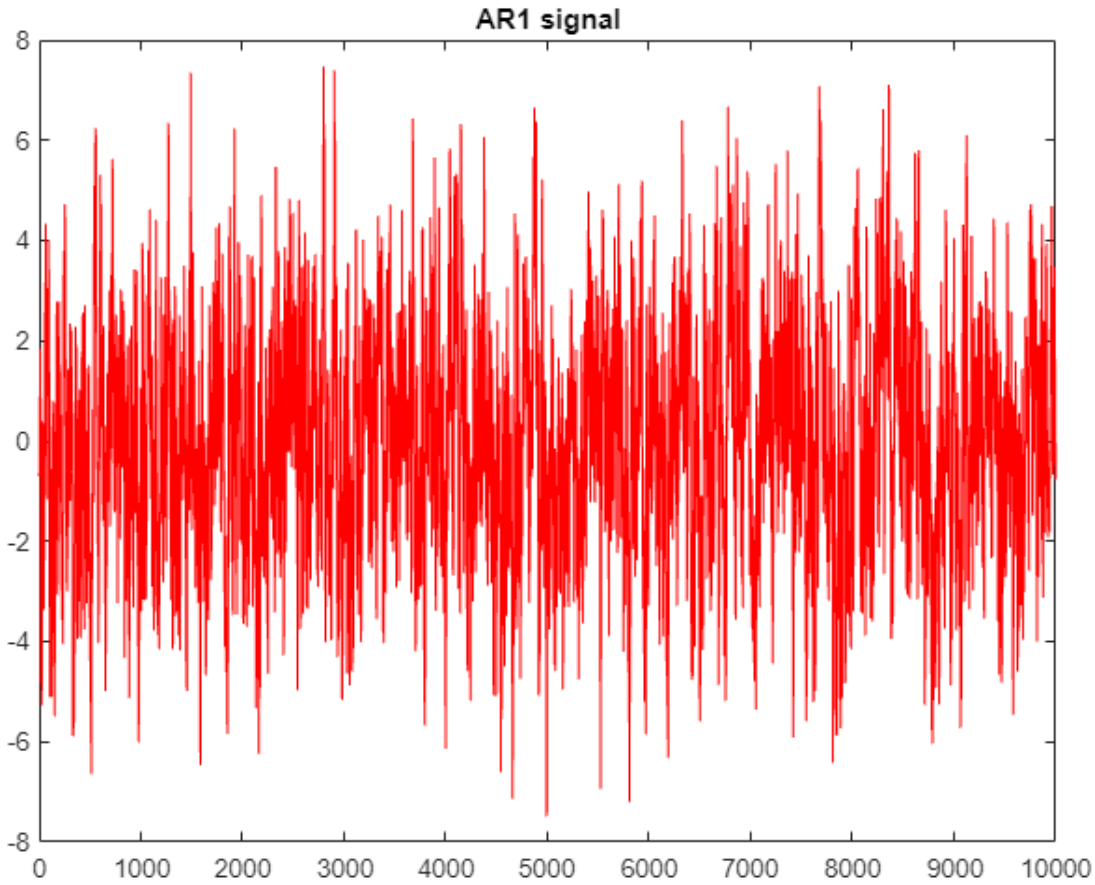
Η PCM υλοποιήθηκε στο αρχείο PCM.m , η ADM στο αρχείο ADM.m και για την λήψη όλων των απαραίτητων μετρήσεων οι συναρτήσεις καλούνται από το Script.m . Οι πηγαίοι κώδικες βρίσκονται όλοι στο παράρτημα Α.

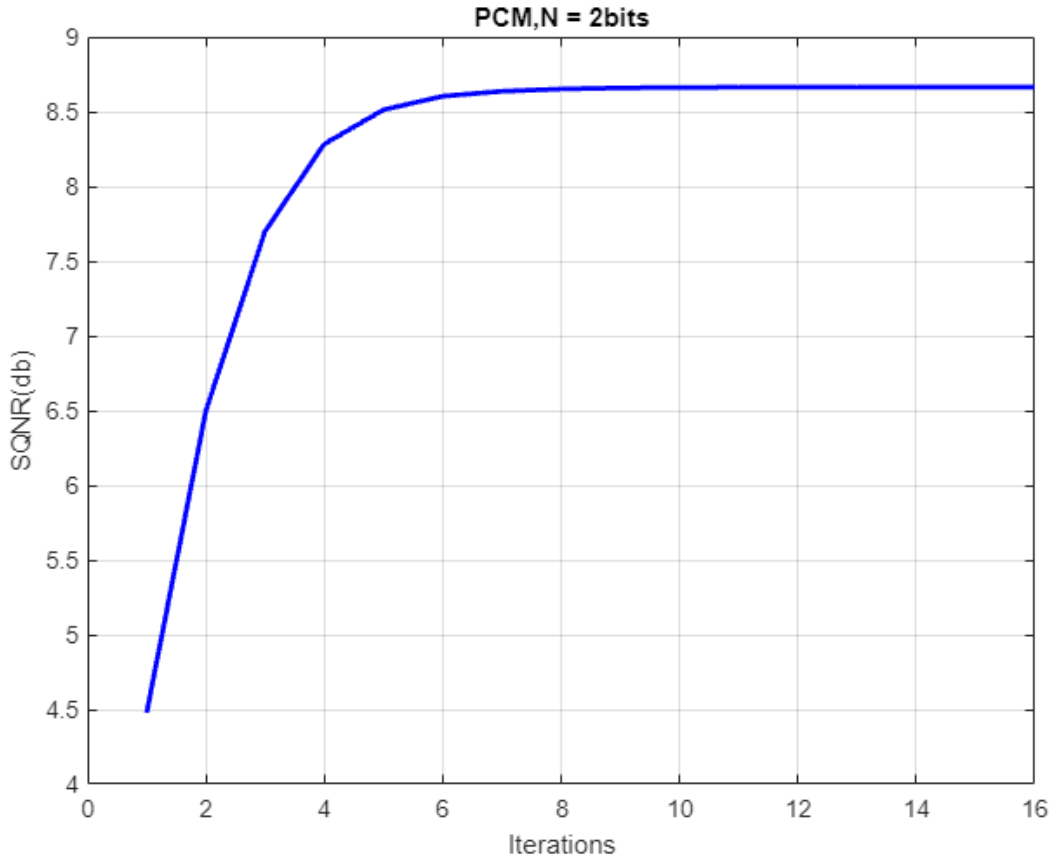
Ερώτημα 1.1:

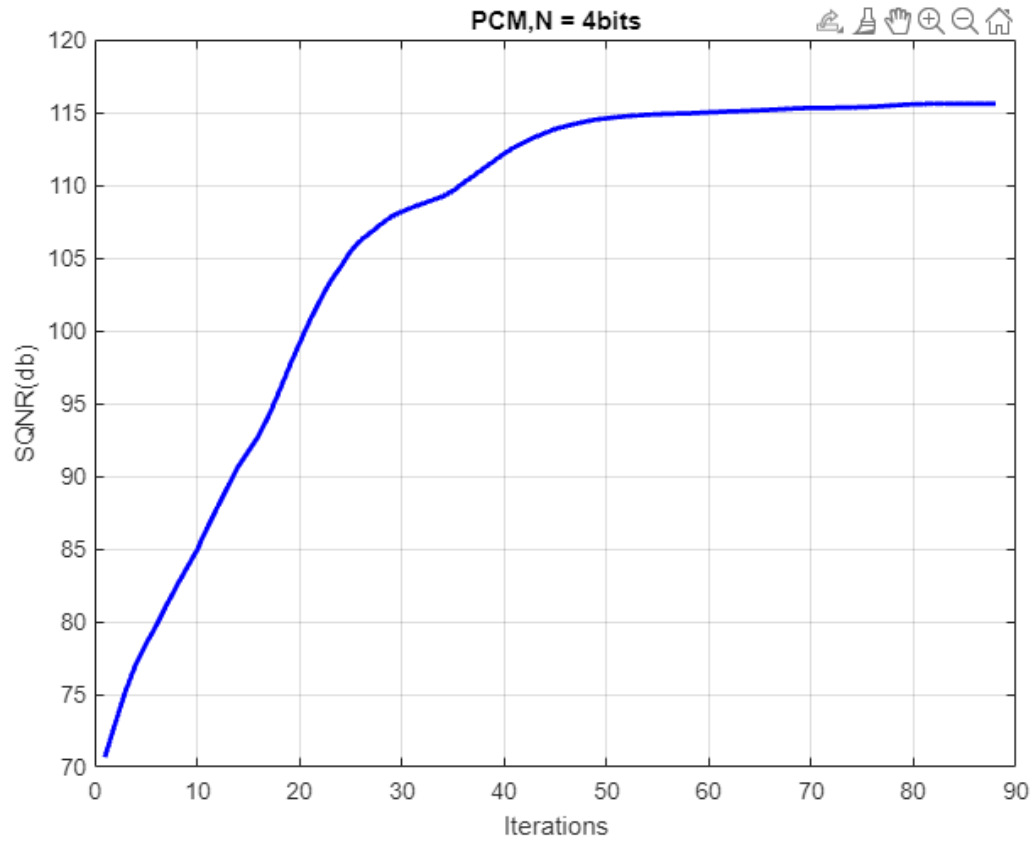
Για τη διαδικασία AR παρατίθενται οι γραφικές παραστάσεις μεταβολών του SQNR συναρτήσει του αριθμού επαναλήψεων του αλγορίθμου Lloyd-Max.

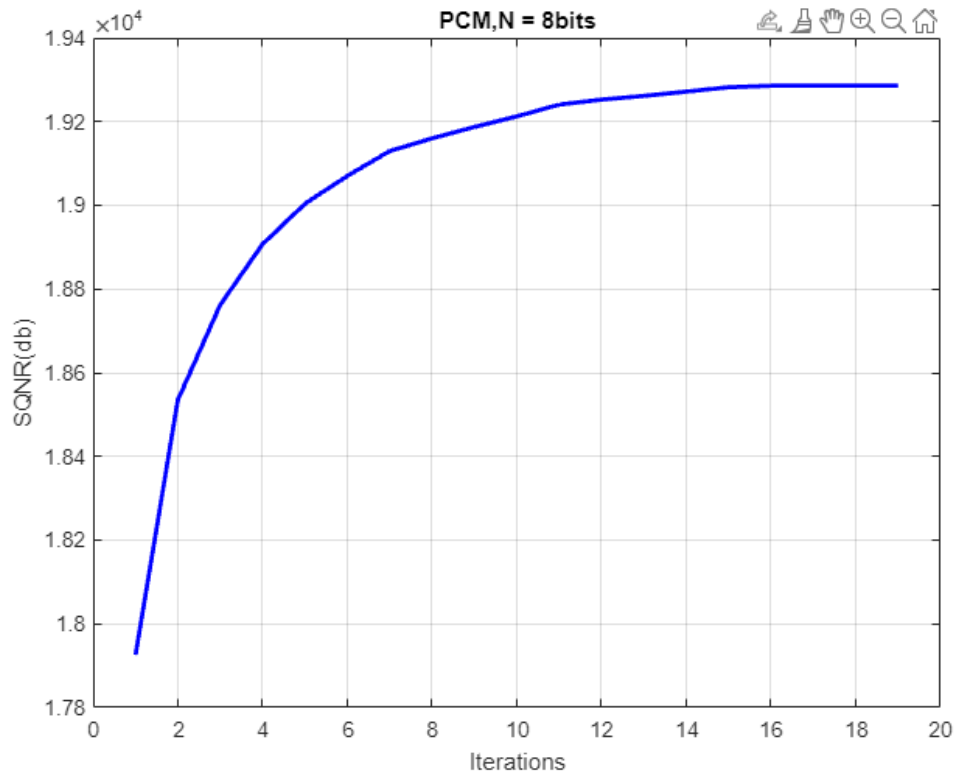
Για την AR(1) διαδικασία με τον συντελεστή $\alpha_1 = 0.9$:

Το αρχικό σήμα:



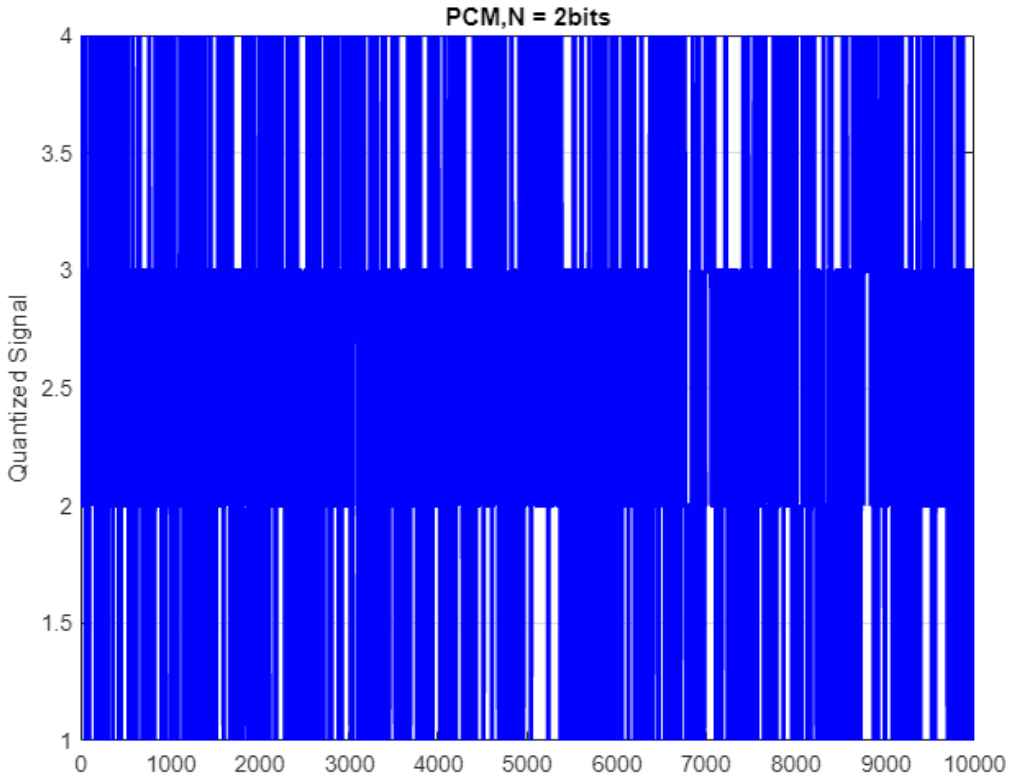

PCM,N = 2bits
SQNR(db)
Iterations
9
8.5
8
7.5
7
6.5
6
5.5
5
4.5
4
0
2
4
6
8
10
12
14
16

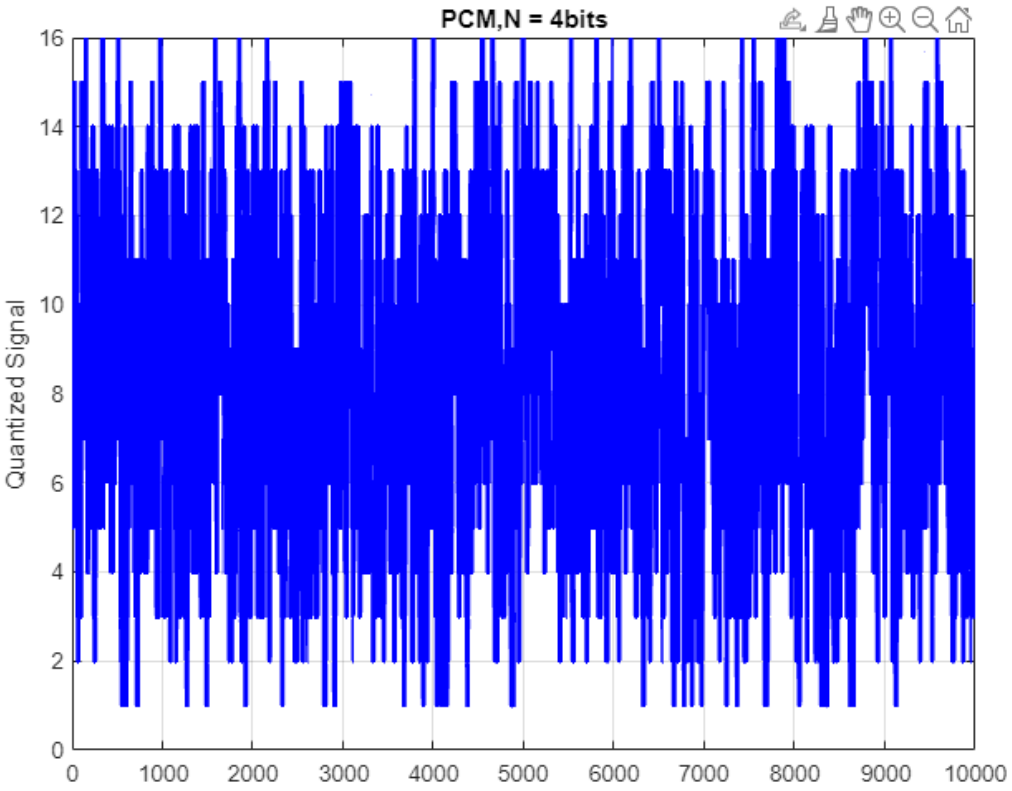

PCM,N = 4bits
SQNR(db)
Iterations
120
115
110
105
100
95
90
85
80
75
70
0
10
20
30
40
50
60
70
80
90

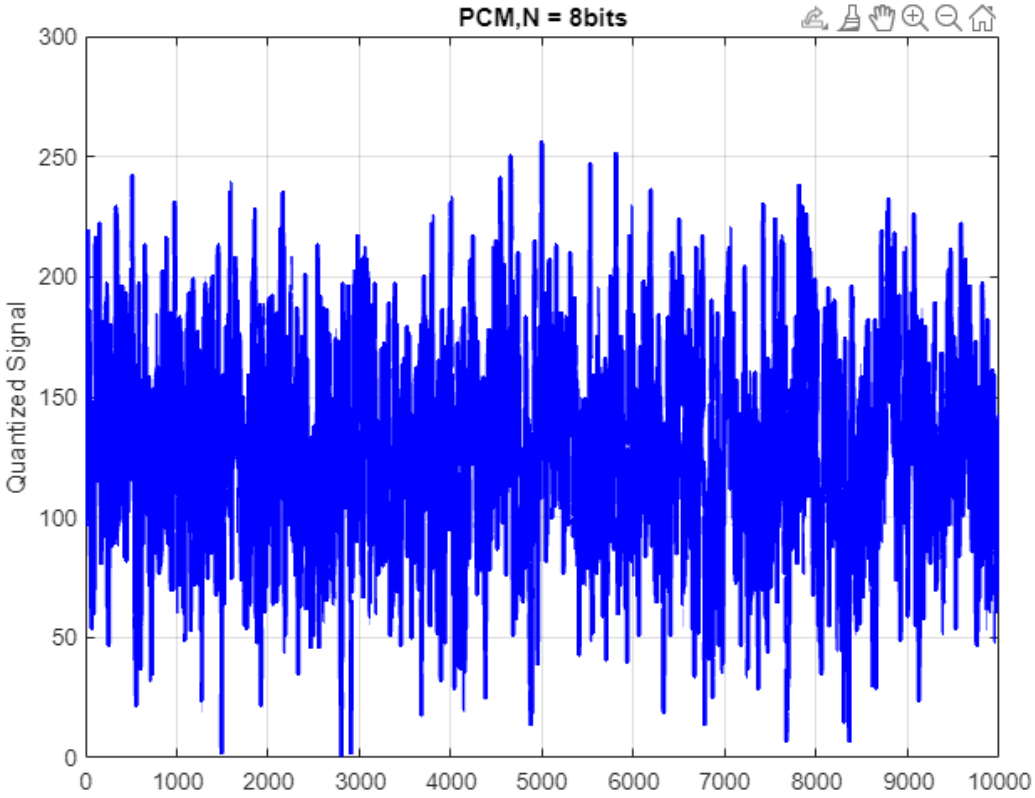


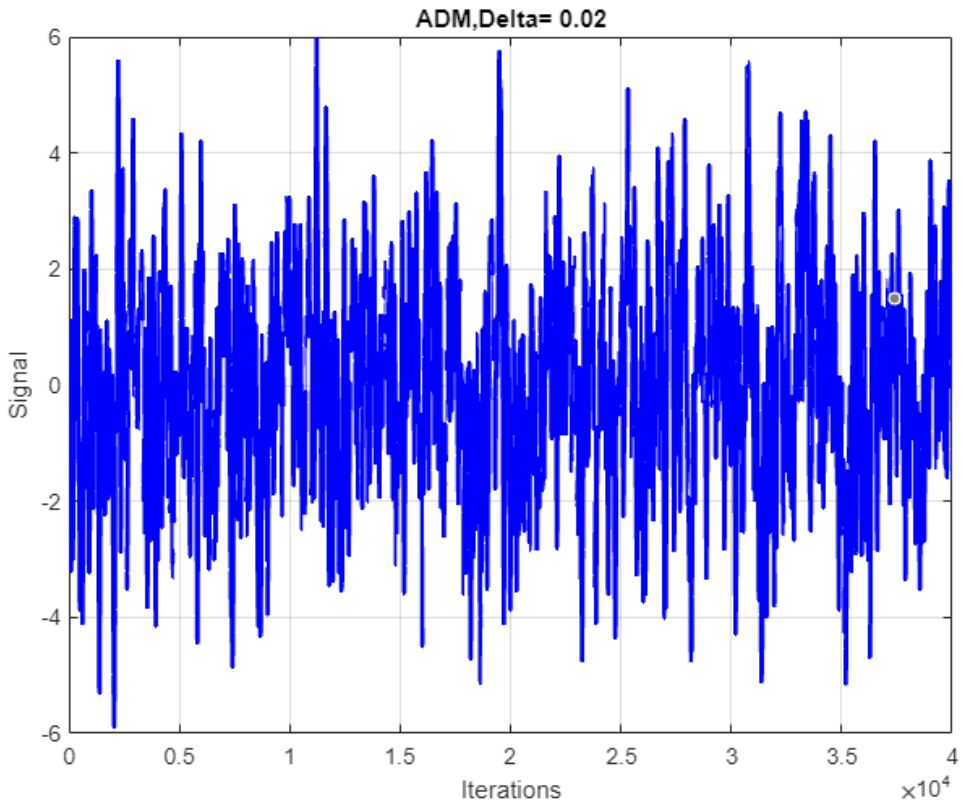
Η ADM μετρήθηκε ότι έχει SQNR -1.934991e-01

Ακολουθούν τα κβαντισμένα σήματα για PCM με N=2,4,8 και ADM.

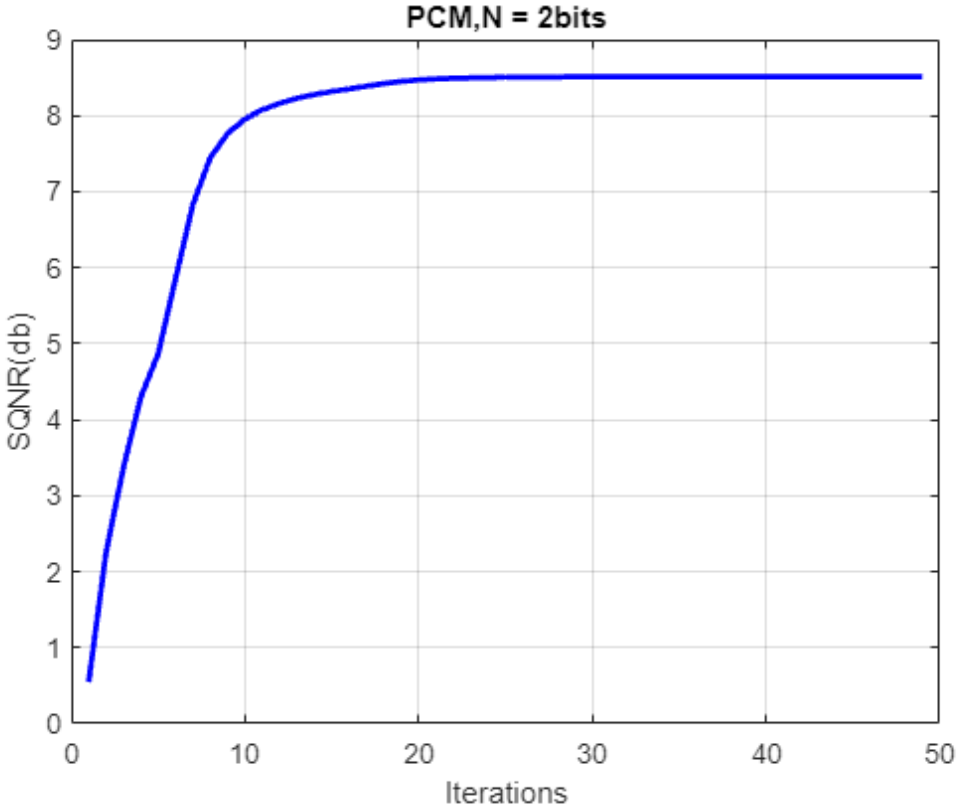


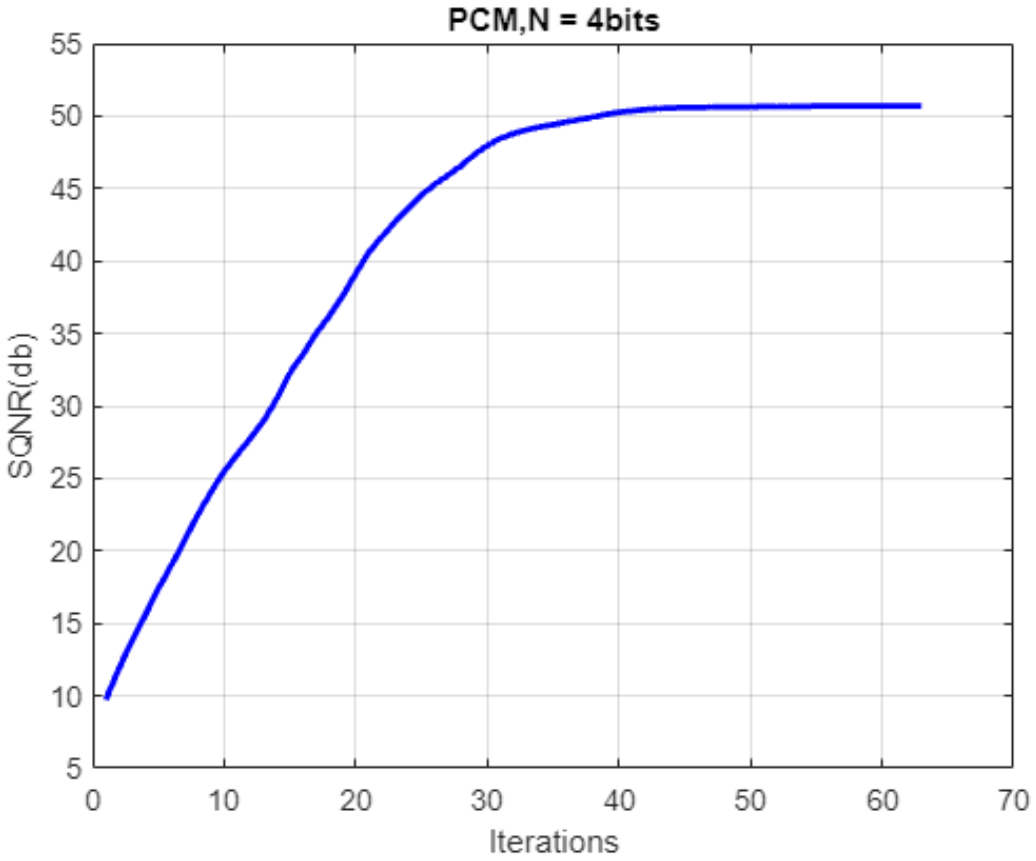

PCM,N = 4bits
Quantized Signal
0
2
4
6
8
10
12
14
16
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

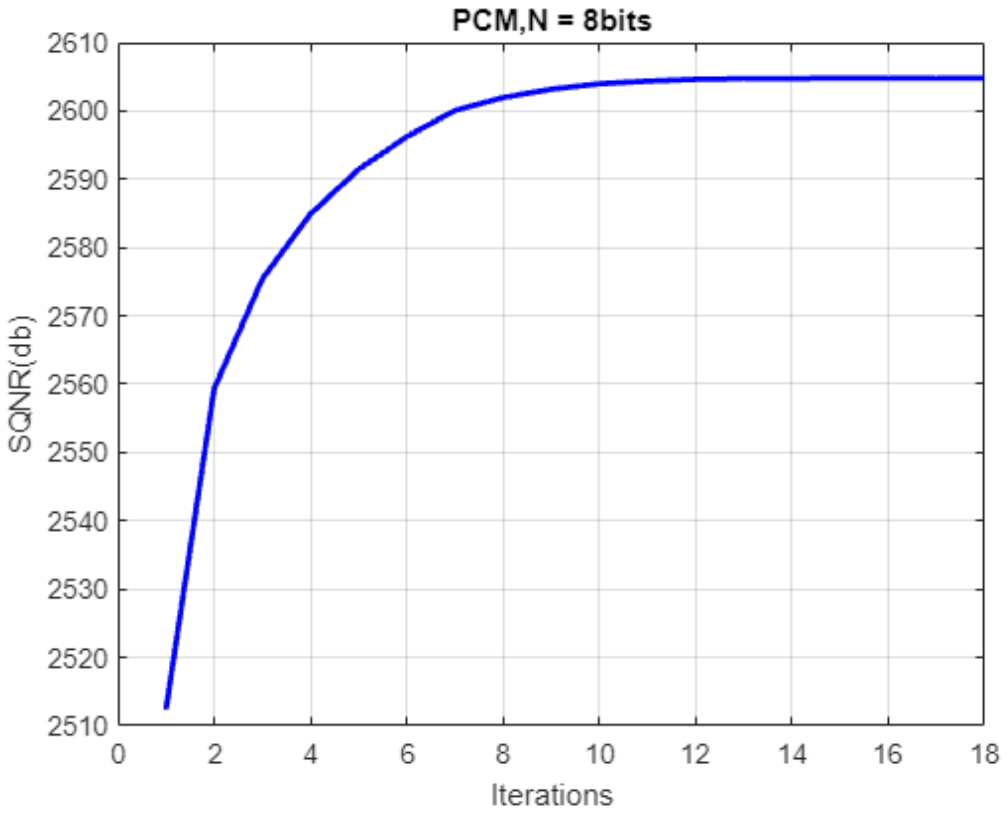

PCM,N = 8bits
Quantized Signal
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000



Για την AR(1) διαδικασία με τον συντελεστή $\alpha_2 = 0.01$:

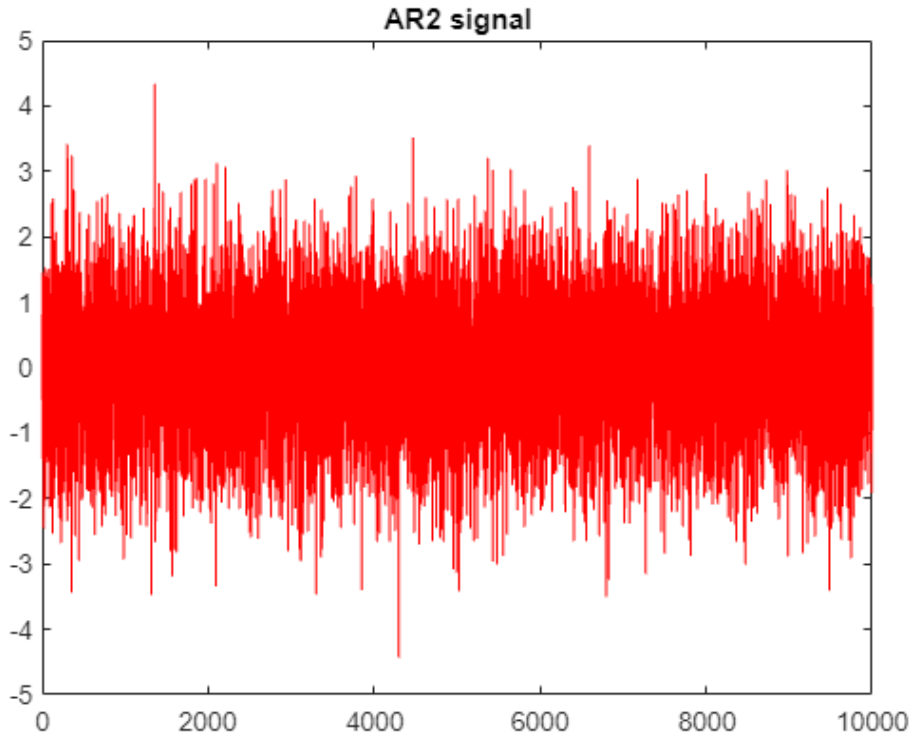




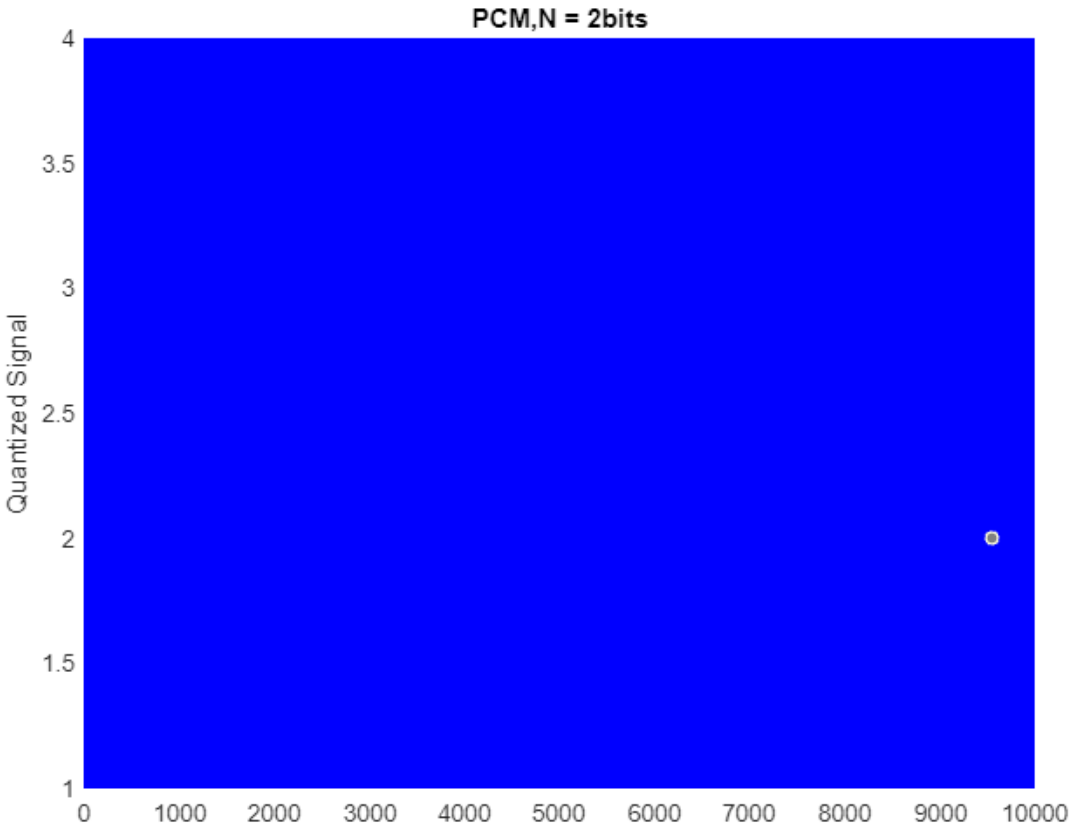


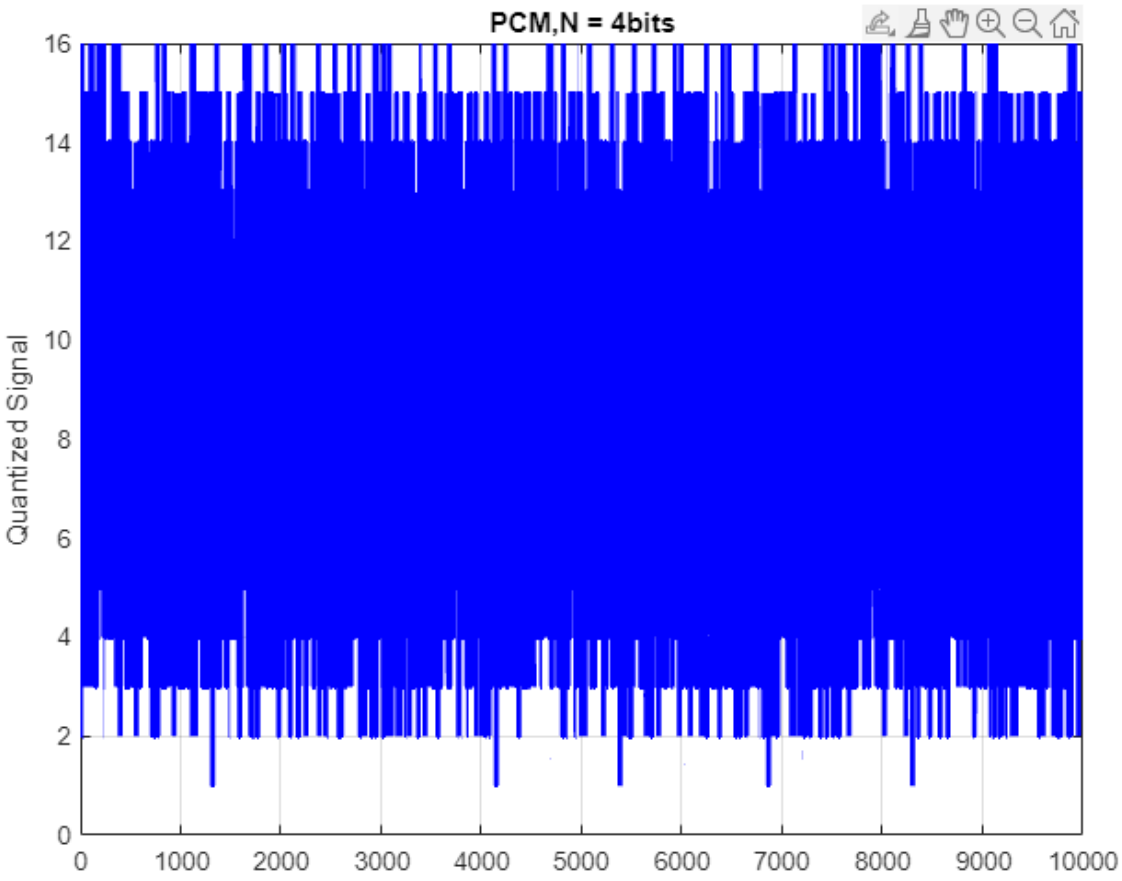
Το SQNR της ADM είναι -5.951562e-01

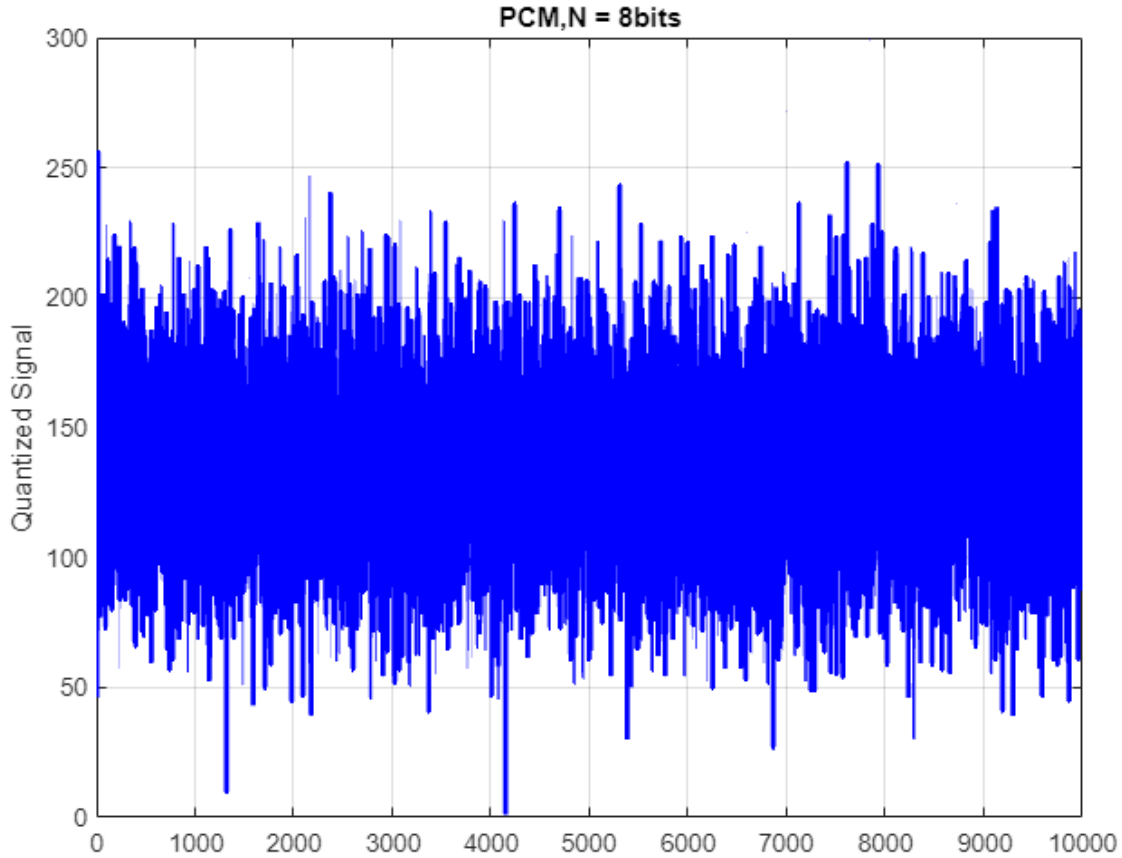
Το αρχικό σήμα για $\alpha_2 = 0.01$:



Ακολουθούν τα κβαντισμένα σήματα για PCM με N=2,4,8 και ADM.




PCM,N = 4bits
Quantized Signal
0
2
4
6
8
10
12
14
16
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


PCM,N = 8bits
Quantized Signal
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Ερώτημα 1.2:

Για την PCM υπολογίστηκε η εντροπία στην έξοδο του κβαντιστή και παρατίθεται παρακάτω:

Για a1:

Για N=2: Η εντροπία είναι 1.915972e+00

Για N=4: Η εντροπία είναι 3.745167e+00

Για N=8: Η εντροπία είναι 7.301416e+00

Για a2:

Για N=2: Η εντροπία είναι 1.911387e+00

Για N=4: Η εντροπία είναι 3.665328e+00

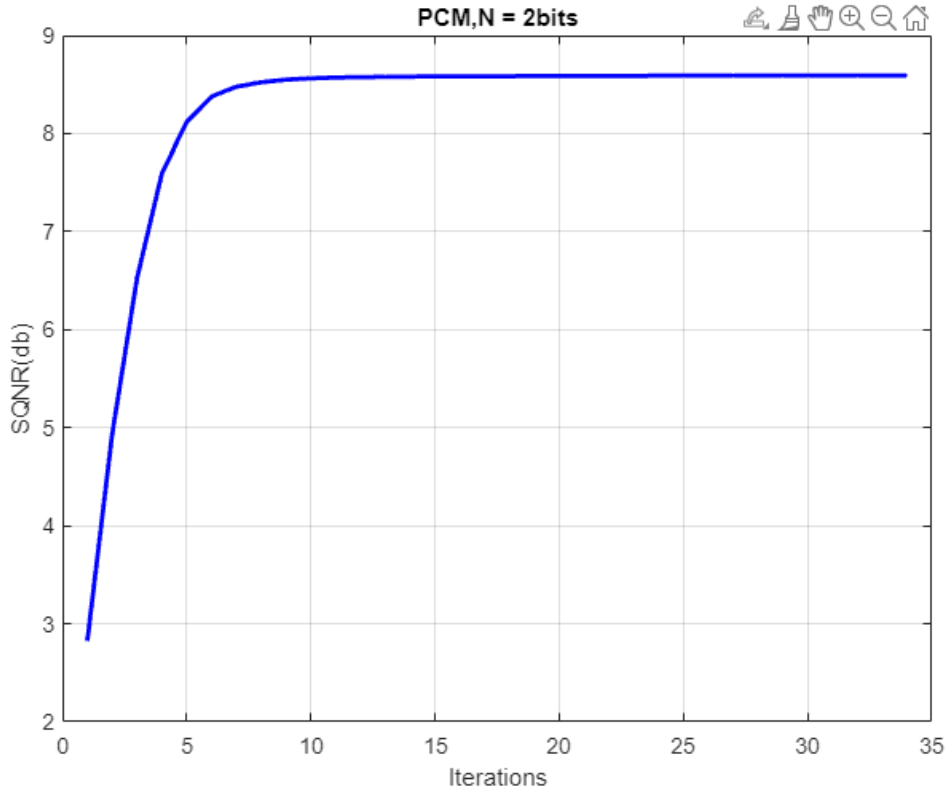
Για N=8: Η εντροπία είναι 6.984118e+00

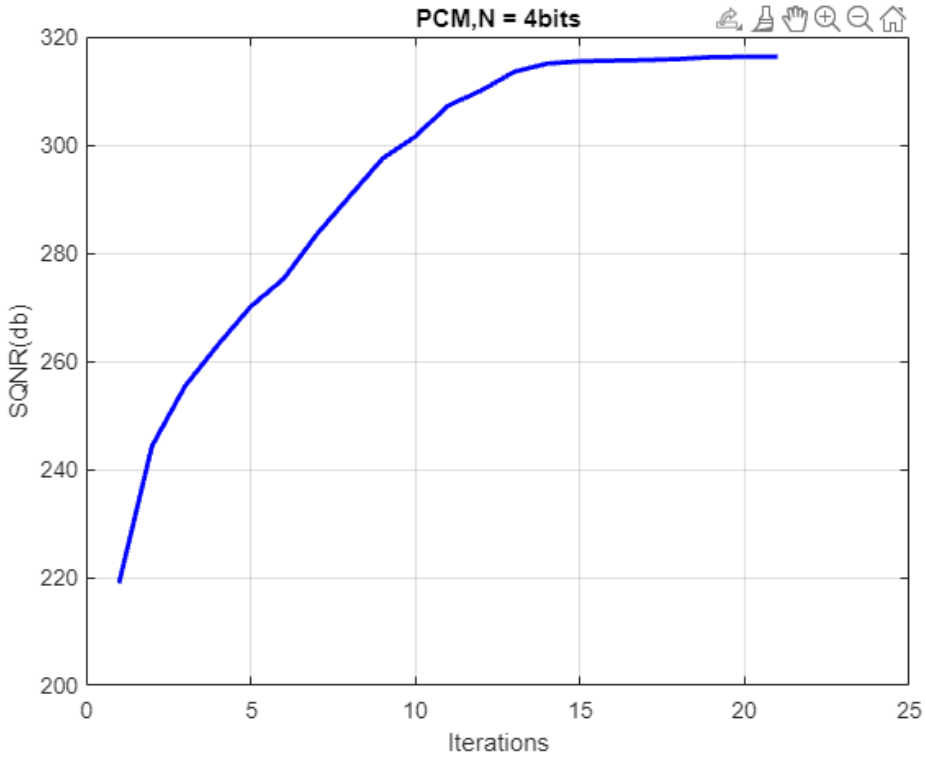
Ερώτημα 1.3:

Με βάση τις τιμές του SQNR η PCM για N= 8 bits είναι η καλύτερη, ακολουθεί η PCM για N=4 bits, μετά η PCM για N=2 bits και τέλος η ADM.

Με βάση τις κυματομορφές εξόδου οι καλύτερες είναι η ADM , μετά η PCM για N=8 , μετά η PCM για N=4 και τέλος η PCM για N=2.

Ερώτημα 2:


PCM,N = 2bits
SQNR(db)
Iterations
9
8
7
6
5
4
3
2
0
5
10
15
20
25
30
35

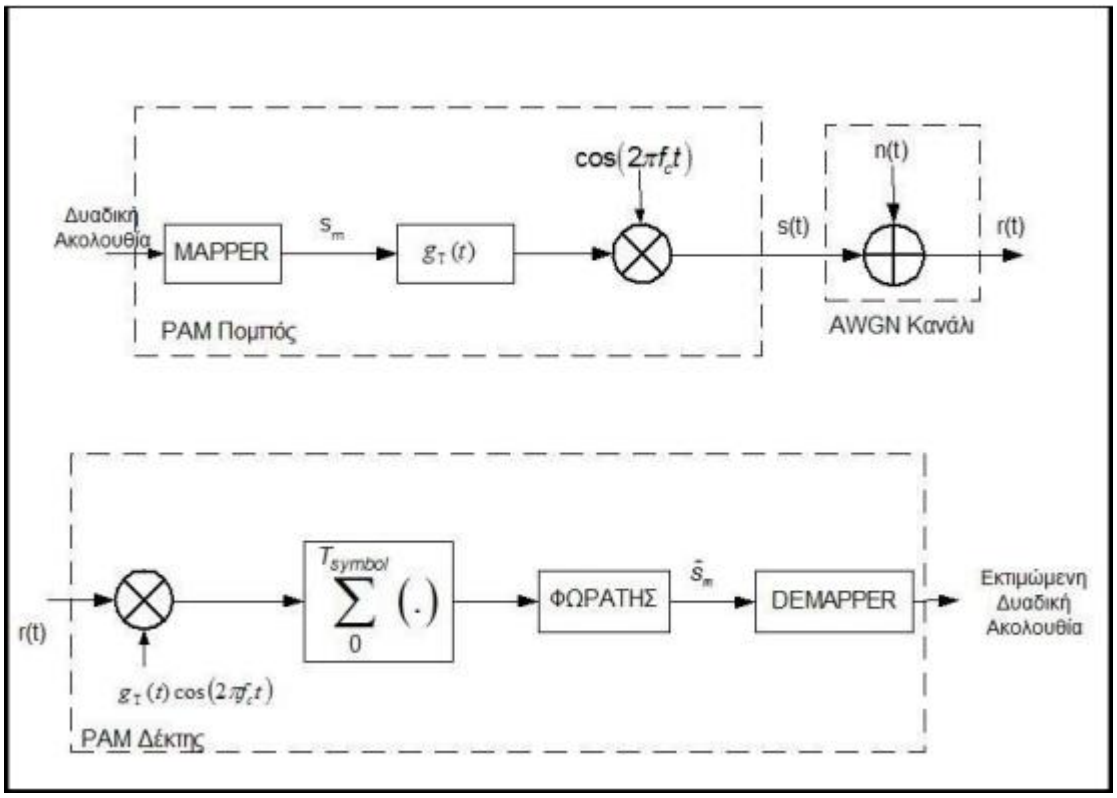

PCM,N = 4bits
SQNR(db)
Iterations
320
300
280
260
240
220
200
0
5
10
15
20
25

Με βάση το SQNR συμπεραίνουμε ότι για N=4 η PCM είναι σημαντικά καλύτερη από ότι για N=2. Αυτό επιβεβαιώνεται και από ότι το οπτικό αποτέλεσμα όταν ανακατασκευαστούν οι εικόνες.

Η εντροπία στην έξοδο του κβαντιστή για N= 2 είναι 1.968754e+00, και για N= 4 η εντροπία είναι 3.415174e+00

Μέρος 2:

Ερώτημα 1) Στην συγκεκριμένη άσκηση ζητήθηκε η υλοποίηση προσομοίωσης τηλεπικοινωνιακού συστήματος M-PAM για M = 4 και M = 8 αντίστοιχα.



Σε ένα σύστημα M-PAM ο πομπός δέχεται ως είσοδο μια δυαδική ακολουθία την οποία μετατρέπει σε σύμβολα, την πολλαπλασιάζει με τον ορθογώνιο παλμό και στην συνέχεια το σήμα μεταφέρεται στη ζώνη μετάδοσης μέσω του διαμορφωτή. Στο σήμα προστίθεται AWGN θόρυβος και φτάνει στο δέκτη του M-PAM συστήματος. Εκεί αποδιαμορφώνεται και προκύπτει ένα μονοδιάστατο διάνυσμα το οποίο εισάγεται στο φωρατή όπου και αποφασίζεται ποιο σύμβολο στάλθηκε. Τέλος, ο demapper κάνει την αντίστροφη αντιστοίχιση από σύμβολα σε bits.

Αρχικά, στο σύστημα δίνω μια ακολουθία των 10^5 δυαδικών ψηφίων που δημιουργείται τυχαία μέσω της συνάρτησης randsrc. Η ακολουθία διοχετεύεται στον mapper όπου τα δυαδικά ψηφία ανάλογα του M, αντιστοιχίζεται σε σύμβολα. Δηλαδή, ο mapper είναι ένας μετατροπέας από bits σε σύμβολα. Κάθε σύμβολο αντιστοιχεί σε μια συγκεκριμένη ακολουθία $\log_2 M$ bits. Αντίστοιχα, ο demapper δέχεται ως είσοδο σύμβολο και εξάγει μια ακολουθία $\log_2 M$ bits για κάθε σύμβολο.

Στην κωδικοποίηση Gray εάν δύο σύμβολα είναι γειτονικά τότε σε αυτά ανατίθενται διατάξεις bits που διαφέρουν μόνο κατά 1 bit μεταξύ τους.

Η μετάδοση γίνεται μέσω ορθογώνιου παλμού τον οποίο προσομοιώνουμε. Για τη δημιουργία του ορθογώνιου παλμού χρησιμοποιείται στο modulation ο τύπος g_T που έχει δοθεί στην εκφώνηση.

Το ζωνοπερατό σήμα που εκπέμπει ο πομπός διέρχεται μέσα από ένα ιδανικό κανάλι προσθετικού θορύβου. Ο θόρυβος είναι λευκός και ακολουθεί Gaussian κατανομή μέσης τιμής. Τον παράγω με κλήση της συνάρτησης randn.

Ο αποδιαμορφωτής του συστήματος M-PAM συσχετίζει το ληφθέν σήμα με τη φέρουσα και τον ορθογώνιο παλμό. Η συσχέτιση γίνεται στα χρονικά πλαίσια μιας περιόδου συμβόλου. Ο αποδιαμορφωτής συσχετίζει το σήμα με την συνιστώσα της φέρουσας οπότε προκύπτει το διάνυσμα r που είναι η εκτιμηθείσα τιμή του τρέχοντος συμβόλου.

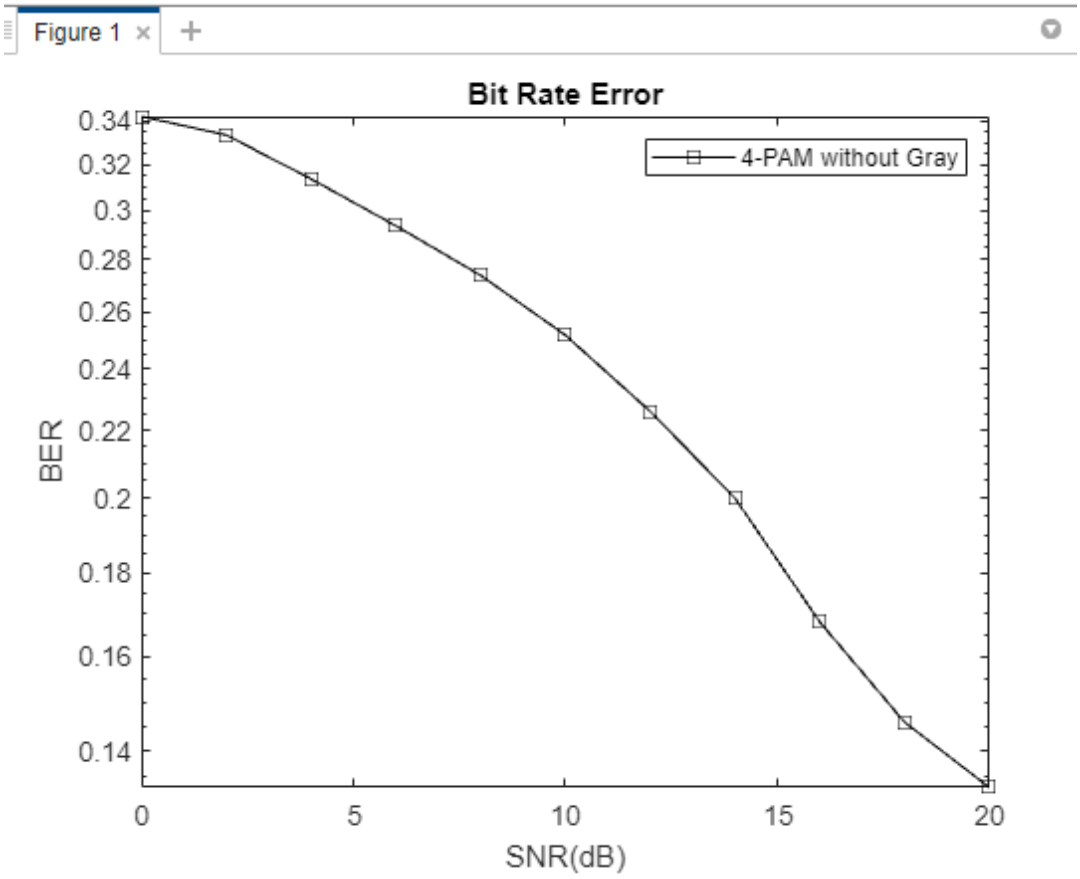
Ο φωρατής με βάση την τιμή r αποφασίζει σε ποιο σύμβολο βρίσκεται πιο κοντά. Το σύμβολο που θα έχει τη μικρότερη απόσταση από το r είναι τελικά το σύμβολο που στάλθηκε.

Η συνάρτηση simulation ενώνει τα παραπάνω και τρέχει τη προσομοίωση με βάση τις παραμέτρους που θα της ορίσουμε εμείς και μετράει το Bit Error Rate και το Symbol Error Rate.

Μπορείτε να δείτε τον πλήρη κώδικα του συστήματος στο παράρτημα Α.

Ερώτημα 2:

Η καμπύλη BER για M=4 για απλή κωδικοποίηση για τιμές του SNR = 0:2:20dB είναι:



Εικόνα 2: BER: 4-PAM without Gray

Από το παραπάνω γράφημα, όπου έχουμε 4-PAM με απλή κωδικοποίηση, μπορούμε να συμπεράνουμε ότι όσο αυξάνουμε το SNR τόσο μειώνεται το σφάλμα. Συγκεκριμένα, όσο το SNR είναι χαμηλό το πληρώνουμε σε σφάλμα.

Η πιθανότητα σφάλματος bit υπολογίζεται από τον παρακάτω κώδικα. Επίσης, όπως έχω αναφέρει πιο πάνω στο προηγούμενο ερώτημα, η συνάρτηση simulation (η οποία βρίσκεται στο παράρτημα Α) τρέχει τη προσομοίωση με βάση τις παραμέτρους και μετράει το Bit Error Rate.

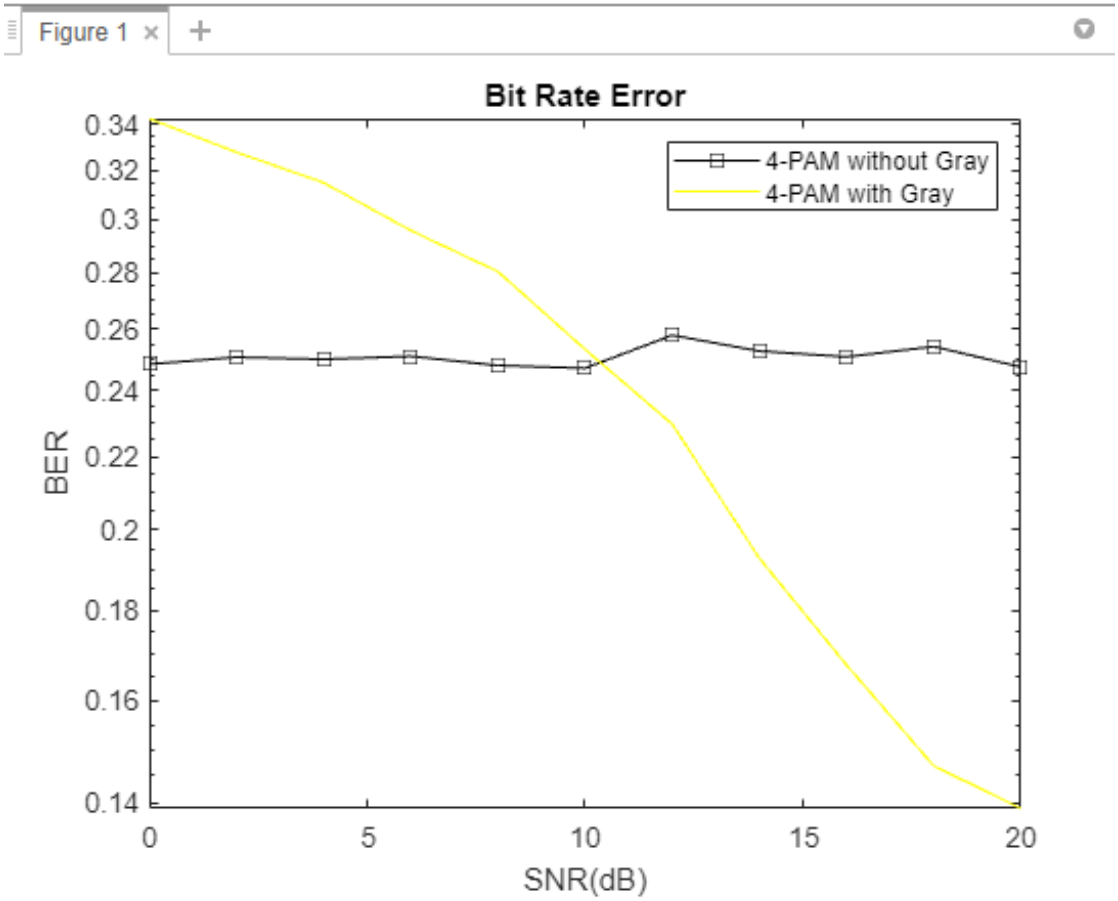
Κώδικας για Bit Error Rate:

```
function BER = ber(input, output,M)
% Calculates the bit error rate

    length_of_input = length(input);
    modular = mod(length_of_input, log2(M));

    % Recover from sending more bits
    if modular ~= 0
        % calculate the reduntant bits of the output
        further_elements_to_add = log2(M) - modular;
```

```
        input(length_of_input + further_elements_to_add) = input(length_of_input);
        input(length_of_input) = 0;
    end

    BER = sum(input ~= output)/length(output);
end
```

Ερώτημα 3:

Η καμπύλη BER για M = 4 με απλή κωδικοποίηση και με χρήση κωδικοποίησης Gray:

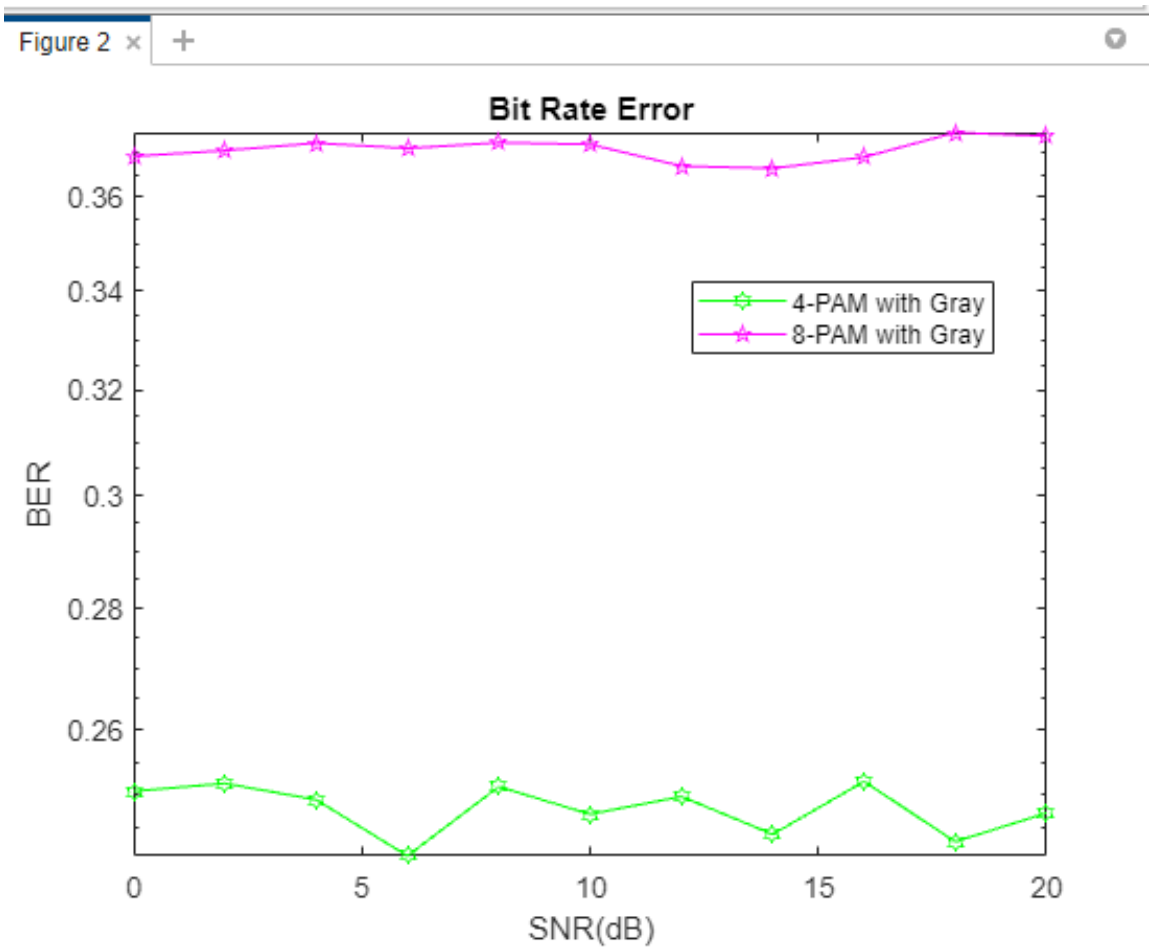


Εικόνα 3: BER: 4-PAM with and without Gray

Από τις παραπάνω καμπύλες συμπεραίνουμε ότι η χρήση της κωδικοποίησης Gray έχει νόημα. Παρατηρούμε ότι η κωδικοποίηση Gray είναι πιο αποδοτική από την απλή και ότι το σφάλμα είναι περισσότερο σταθερό.

Ερώτημα 4:

Η καμπύλη BER για M = 4 και M = 8 με κωδικοποίηση Gray:

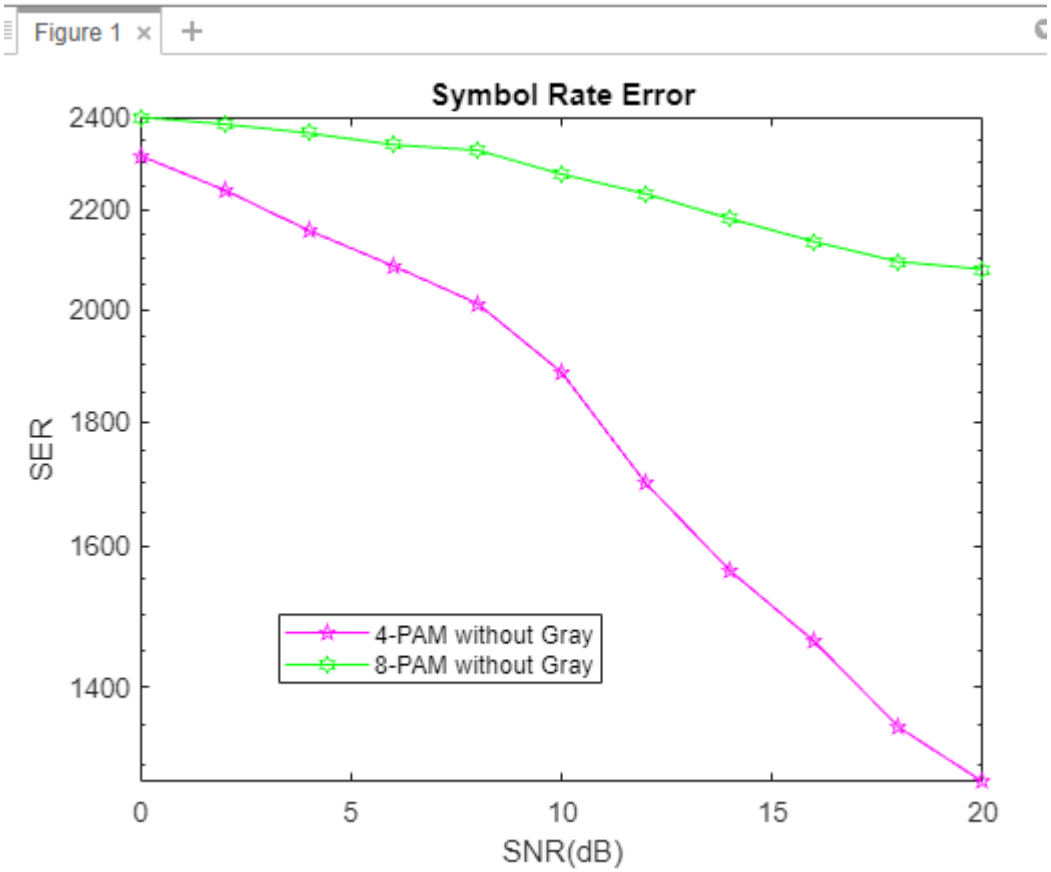


Εικόνα 4: BER: 4-PAM and 8-PAM with Gray

Παρατηρούμε ότι το σφάλμα για 4-PAM είναι μικρότερο από ότι για 8-PAM.

Ερώτημα 5:

Η καμπύλη SER για M = 4 και M = 8 με απλή κωδικοποίηση:



Εικόνα 5: SER: 4-PAM and 8-PAM without Gray

Παρατηρούμε ότι στην περίπτωση 4-PAM η πιθανότητα σφάλματος είναι μικρότερη σε σύγκριση με την 8-PAM. Αυτό συμβαίνει επειδή όσο μικρότερο είναι το M τόσο μικρότερη θα είναι και η πιθανότητα σφάλματος.

Επίσης, η πιθανότητα σφάλματος συμβόλου υπολογίζεται από τον παρακάτω κώδικα, και η συνάρτηση simulation χρησιμοποιεί τη προσομοίωση με βάση τις παραμέτρους και υπολογίζει το Symbol Error Rate.

Κώδικας για Symbol Error Rate:

```
function SER = ser(input, output)
% Calculates the symbol error rate
    SER = nnz(output' - input);
end
```

ΠΑΡΑΡΤΗΜΑ Α:

Παρακάτω παρατίθενται οι κώδικες που χρησιμοποίησα σε κάθε ερώτημα της εργασίας.

Μέρος 1ο

PCM.m κώδικας :

```
function [xq,centers,D] = PCM(x,N,min,max)
% INPUT ----------------------------------------------------------------
%   x: Input signal in vector format
%   N: Number of bits
%   min: Minimum acceptable value of input signal
%   max: Maximum acceptable value of input signal
% OUTPUT ---------------------------------------------------------------
%   xq: Encoded vector of output signal
%   centers: Centers of quantization segments
%   D: Vector of signal's distortion at every repetition
% ----------------------------------------------------------------------

% Number of bits used.
v = N;
% Levels of quantization.
quant_levels = 2^v;
% Initialization of vector xq.
xq = zeros(length(x),1);
% Quantization step.
quant_step = (abs(min)+max)/quant_levels;
% Calculation of centers.
centers = zeros(quant_levels,1);
for i=1:quant_levels
    centers(i) = max-(2*(i-1)+1)*(quant_step/2);
end
% Loop for Lloyd-Max.
T = zeros((quant_levels+1),1);
counter = 1;
previous_distortion = 0;
while 1
    %Calculation of new segments' limits.
    T(1) = max; % Higher level limit.
    for i=2:quant_levels % Middle levels.
        T(i) = (centers(i)+centers(i-1))/2;
    end
    T(quant_levels+1) = min; % Lower level limit.

    %Output signal
    q_max_value = 1; q_min_value = quant_levels;
    for i=1:length(x)
        if (x(i) >= max)
            xq(i) = q_max_value;
        elseif (x(i) <= min)
            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

    % Check if there are any zeros.
    if (all(xq)) % If not, calculate the distortion.
        D(counter) = mean((x-centers(xq)).^2);
    end
    SQNR(counter)=mean(x.^2)/D(counter);
    %Criterion check. If fulfilled, stop the loop.
    difference = abs(D(counter)-previous_distortion);
    if (difference < eps('single')&&N<=4)
        break;
    elseif (difference < eps()&&N>4)
        break;
    else % Else store distortion for the next comparison.
        previous_distortion = D(counter);
    end

    %New levels of quantization.
    temp_sum = zeros(quant_levels,1);
    temp_counter = zeros(quant_levels,1);
    for n=1:quant_levels
        for i=1:length(x)
            % Check if x(i) belongs to n level.
            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;
            % If x(i) is greater than max_value.
            elseif ((x(i) > T(n)) && (n == 1))
                temp_sum(n) = temp_sum(n) + T(n);
                temp_counter(n) = temp_counter(n) + 1;
            % If x(i) is smaller than min_value.
            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
        % Calculating the new center for n level.
        if (temp_counter(n) > 0) % If greater than zero, calculate.
            centers(n) = temp_sum(n)/temp_counter(n);
        end

    end

    counter = counter + 1;
end
stoixeio = unique(xq);


[emfaniseis,~] = hist(xq,stoixeio);

pithanotita =(emfaniseis/length(xq)) ;

Entropia = -pithanotita*log2(pithanotita)';

fprintf('H entropia gia N = %d einai %d \n',N,Entropia);
figure;
plot(SQNR,'-b','LineWidth',2);
title(['PCM,N = ',num2str(N),'bits']);
ylabel('SQNR(db)')
```

```
xlabel('Iterations')
grid on;

figure;
plot(xq,'-b','LineWidth',2);
title(['PCM,N = ',num2str(N),'bits']);
ylabel('Quantized Signal')
hold off;
grid on;
```

ADM κώδικας:

```
function [ADMout] = ADM(sig_in, Delta, td, ts)
% INPUT ----------------------------------------------------------------
%   sig_in: Input signal in vector format
%   Delta: Minimum step size
%   td: The original sampling period of the input signal, sig_in
%   ts: The required sampling period for ADM output
% OUTPUT ---------------------------------------------------------------
%   ADMout:Encoded vector of output signal

    if (round(ts/td) >= 2)
        M = round(ts/td);    %Nearest integer
        xsig = interp(sig_in,M);
        Lxsig = length(xsig);
        cnt1 = 0;
        cnt2 = 0;
        sum = 0;
        for i=1:Lxsig

            if (xsig(i) == sum)
            elseif (xsig(i) > sum)
                if (cnt1 < 2)
                    sum = sum + Delta;  %Step up by Delta
                elseif (cnt1 == 2)
                    sum = sum + 2*Delta;    %Double the step size after
                                            %first two increase
                elseif (cnt1 == 3)
                    sum = sum + 4*Delta;    %Double step size
                else
                    sum = sum + 8*Delta; %Still double and then stop

                end
                if (sum < xsig(i))
                    cnt1 = cnt1 + 1;
                else
                    cnt1 = 0;
                end
            else
                if (cnt2 < 2)
                    sum = sum - Delta;
                elseif (cnt2 == 2)
                    sum = sum - 2*Delta;
                elseif (cnt2 == 3)
                    sum = sum - 4*Delta;
                else
```

```
                    sum = sum - 8*Delta;
                end
                if (sum > xsig(i))
                    cnt2 = cnt2 + 1;
                else
                    cnt2 = 0;
                end
            end
             D(Lxsig)= mean(abs(xsig - sum).^2);
            SQNR(Lxsig-1)=mean(xsig.^2)/D(Lxsig);
            SQNR(Lxsig-1)=10*log10(SQNR(Lxsig-1));
            ADMout(((i-1)*M + 1):(i*M)) = sum;

        end
    end

fprintf('To SQNR ths ADM einai %d \n',SQNR(Lxsig-1));
figure;
plot(ADMout,'-b','LineWidth',2);
title(['ADM,Delta= ',num2str(Delta)]);
ylabel('Signal')
xlabel('Iterations')
grid on;
```

Script Κώδικας:

```
clear;
L=10000;
x=randn(L,1);
a1=0.9;
a2=0.01;

y1 = filter(1,[1; -a1], x);
y2 = filter(1,[1; -a2], x);
load cameraman.mat;
x = i(:);
x = (x-128)/128;

figure;
plot(y1,'-r');
title('AR1 signal');

[xq_2,centers_2,D_2] = PCM(y1,2,min(y1),max(y1));
[xq_4,centers_4,D_4] = PCM(y1,4,min(y1),max(y1));
[xq_8,centers_8,D_8] = PCM(y1,8,min(y1),max(y1));

[ADM]= ADM(y2,0.02,8,16);

figure;
plot(y2,'-r');
title('AR2 signal');

[xq_2,centers_2,D_2] = PCM(y2,2,min(y2),max(y2));
[xq_4,centers_4,D_4] = PCM(y2,4,min(y2),max(y2));
[xq_8,centers_8,D_8] = PCM(y2,8,min(y2),max(y2));

%[ADM]= ADM(y2,0.02,8,16);
fprintf('Source 2 \n');
```

```
figure;
plot(x,'-r');
title('Initial image signal');

[xq_2_image,centers_2_image,D_2_image] = PCM(x,2,min(x),max(x));
[xq_4_image,centers_4_image,D_4_image] = PCM(x,4,min(x),max(x));
```

Μέρος 2ο

Ερώτημα 1:

Κώδικας για Mapper:

```
function symbols = mapper(binary_sequence, M, gray)
% Takes as argument a binary sequence and transforms the elements into symbols
% The gray argument denotes if is to be used gray (1) encoding or not (0)

    size_of_binary_sequence = length(binary_sequence);


    % group the bits into groups of log2(M)
    temp = mod(size_of_binary_sequence, log2(M));

    % the sequence which is dividable by log2(M)
    new_bin_seq = binary_sequence(1 : (size_of_binary_sequence - temp), :);
    % grouping of that sequence
    reshaped_sequence = reshape(new_bin_seq, log2(M), (size_of_binary_sequence -
temp) / log2(M));

    % tranform the sequence into binary code for every group of log2(M) bits
    for i = 1 : (size_of_binary_sequence - temp) / log2(M)
        symbols(i) = bin2dec(num2str(reshaped_sequence(:, i)'));
    end

    % the rest of the bits are separately tranformed into a symbol in binary
    % code
    if temp ~= 0
        symbols(i + 1) = bin2dec(num2str(binary_sequence((size_of_binary_sequence
- temp) + 1 : size_of_binary_sequence, 1)'));
    end

    if gray == 1
        symbols = bin2gray(symbols, 'pam', M);
    end
end
```

Κώδικας για Modulator:

```
function s_m = modulator(symbols_array, M)
% Takes as arguments the symbols array that are to be transmitted and
% encodes it according to the argument M
% Returns the modulated signal
```

```
    % size of the array that has the sequence converted into symbols
    size_of_symbols_array = length(symbols_array);

    T_symbol = 4;
    T_sample = 0.1;
    T_c = 0.4; % ferousa
    F_c = 1 / T_c;
    E_s = 1; % Energy per symbol

    % orthogonal pulse
    g = sqrt(2 * E_s / T_symbol);

    % computation of the transmitted signal
    for i = 1: size_of_symbols_array
        for t = 1: T_symbol/T_sample
            s_m(i, t) = g * cos( 2*pi*F_c*t - 2*pi*symbols_array(i)/M );
        end
    end
end
```

Κώδικας για AWGN noise:

```
function received_signal = noise(s_m, SNR, M)
% The noise function takes as argument the s_m signal that is to be
% transmitted and the SNR and adds AWGN

    E_s = 1;
    E_b = E_s / log2(M);
    N_0 = E_b / (10^(SNR/10));

    % We create a gaussian distribution with mean value:
    m = 0;
    % and standard deviation
    sigma = sqrt(N_0 / 2);

    [x, y] = size(s_m);
    % produce AWGN noise
    noise = m + sigma * randn(x, y);

    % add noise to the signal
    received_signal = s_m + noise;
end
```

Κώδικας για Demodulator:

```
function r = demodulator(received_signal)
% The demodulator function takes as argument the received signal and finds
% r

    T_symbol = 4;
```

```
    T_c = 0.4; % ferousa
    F_c = 1 / T_c;

    E_s = 1; % Energy per symbol

    % orthogonal pulse
    g = sqrt(2 * E_s / T_symbol);

    [~, T_symbol] = size(received_signal);

    % demodulation
    for t = 1: T_symbol
        y1(t, 1) = g * cos(2 * pi * F_c * t);
    end

    % calculation of the 2 components
    r = (received_signal * y1);
end
```

Κώδικας για Φωρατή:

```
function symbols = foratis(r, M)
% Takes the r argument and calculates the
% binary (or gray) symbols that was to be send

    [r_lines, ~] = size(r);
    for i = 1: M
        s(i, 1) = cos( 2 * pi * i / M );
    end

    % calculates the symbol which presents the greatest propability to
    % be the sent symbol
    for j =1: r_lines
        for i = 1: M
            temp(i, 1) = norm([r(j,1)] - s(i,:));
        end
        [~, symbols(j, 1)] = min(temp);
    end

    symbols = mod(symbols,M);
end
```

Κώδικας για Demapper:

```
function received_bits = demapper(symbols, encoding, gray)
% Converts the received symbols to bits
    if gray == 1
         symbols = gray2bin(symbols, 'pam', encoding);
    end

    received_bits = dec2bin(symbols);

    [m, n] = size(received_bits);

    % reshape the matrix with the received bits to an array
    received_bits = reshape(received_bits', m*n, 1);
```

```
    % Only for binaries
    received_bits = double(received_bits) - 48;
end
```

Κώδικας για Bit Error Rate:

```
function BER = ber(input, output,M)
% Calculates the bit error rate

    length_of_input = length(input);
    modular = mod(length_of_input, log2(M));

    % Recover from sending more bits
    if modular ~= 0
        % calculate the reduntant bits of the output
        further_elements_to_add = log2(M) - modular;

        input(length_of_input + further_elements_to_add) = input(length_of_input);
        input(length_of_input) = 0;
    end

    BER = sum(input ~= output)/length(output);
end
```

Κώδικας για Symbol Error Rate:

```
function SER = ser(input, output)
% Calculates the symbol error rate
    SER = nnz(output' - input);
end
```

Κώδικας για Simulation:

```
function [BER , SER] = simulation(in_binary_seq, M, SNR, gray)
% Simulates the M-PAM mod-demod
    sa = mapper(in_binary_seq, M , gray);
    sig_m = modulator(sa, M);
    received_signal = noise(sig_m,SNR,M);
    r = demodulator(received_signal);
    E = foratis(r, M);
    out_binary_seq = demapper(E', M, gray);
    BER = ber(in_binary_seq,out_binary_seq,M);
    if nargout >= 2
        SER = ser(sa, E);
    end
end
```

Ερώτημα 2:

Κώδικας BER για M=4, με απλή κωδικοποίηση και για SNR = 0:2:20 dB:

```
% BER
clear all;
clc;
binary_seq = randsrc(10000, 1, [0,1]);

i = 1;
x = (0: 2: 20);

for SNR = 0: 2: 20
    M=4;
    BER_4_pam_without_gray(i, 1) = simulation(binary_seq, M, SNR, 0);
    i = i + 1;
end

% BER Plot
semilogy(x', BER_4_pam_without_gray, 'ks-');
legend('4-PAM without Gray');
title('Bit Rate Error');
xlabel('SNR(dB)');
ylabel('BER');
```

Ερώτημα 3:

Κώδικας BER για M=4, με απλή κωδικοποίηση και με κωδικοποίηση Gray:

```
% BER
clear all;
clc;
binary_seq = randsrc(10000, 1, [0,1]);

i = 1;
x = (0: 2: 20);

for SNR = 0: 2: 20
    M=4;
    BER_4_pam_without_gray(i, 1) = simulation(binary_seq, M, SNR, 0);
    M=4;
    BER_4_pam_with_gray(i, 1) = simulation(binary_seq, M, SNR, 1);
    i = i + 1;
end

% BER Plot
semilogy(x', BER_4_pam_with_gray, 'ks-',x', BER_4_pam_without_gray, '-y');
legend('4-PAM without Gray','4-PAM with Gray');
title('Bit Rate Error');
xlabel('SNR(dB)');
ylabel('BER');
```

Ερώτημα 4:

Κώδικας BER για M=4,8 με απλή κωδικοποίηση:

```
% BER
clear all;
clc;
binary_seq = randsrc(10000, 1, [0,1]);

i = 1;
x = (0: 2: 20);

for SNR = 0: 2: 20

    M=4;
    BER_4_pam_with_gray(i, 1) = simulation(binary_seq, M, SNR, 1);

    M = 8;
    BER_8_pam_with_gray(i, 1) = simulation(binary_seq, M, SNR, 1);

    i = i + 1;
end

% BER Plot
semilogy(x', BER_4_pam_with_gray, 'gh-',x', BER_8_pam_with_gray, 'mp-');
legend('4-PAM with Gray', '8-PAM with Gray');
title('Bit Rate Error');
xlabel('SNR(dB)');
ylabel('BER');
```

Ερώτημα 5)

Κώδικας SER για M=4,8 με απλή κωδικοποίηση για τιμές SNR = 0:2:20dB :

```
% SER
clear all;
clc;
binary_seq = randsrc(10000, 1, [0,1]);

i = 1;
x = (0: 2: 20);

for SNR = 0: 2: 20
    M = 4;
    [~, SER_4_pam_without_gray(i, 1)]  =  simulation(binary_seq, M, SNR, 0);

    M = 8;
    [~, SER_8_pam_without_gray(i, 1)] = simulation(binary_seq, M, SNR, 0);

    i = i + 1;
end

% SER Plot
semilogy(x', SER_4_pam_without_gray, 'mp-',x', SER_8_pam_without_gray, 'gh-');
legend('4-PAM without Gray','8-PAM without Gray');
title('Symbol Rate Error');
xlabel('SNR(dB)');
ylabel('SER');
```