Ψηφιακές Τηλεπικοινωνίες

Πρώτο Σετ - 2022/2023

Ονοματεπώνυμο: Μάριος Στεφανίδης
ΑΜ: 106758
Έτος: 5ο

ΜΕΡΟΣ Α

Μέρος 1.1 Κωδικοποίηση PCM

Αρχικά, έχει δημιουργηθεί κώδικας για την υλοποίηση ενός **ομοιόμορφου κβαντιστή** (uniform_quantizer). Ειδικότερα, πραγματοποιείται έλεγχος, ώστε κάθε στοιχείο του πίνακα x να βρίσκεται μεταξύ των δοθέντων ορίων (δηλ. min_value < x < max_value). Στη συνέχεια, υπολογίζεται το βήμα σύμφωνα με το οποίο ουσιαστικά θα προκύψουν οι διάφορες υποπεριοχές που θα χωριστεί το διάστημα min_value έως max_value και επομένως προκύπτουν τα κέντρα των παραπάνω υποπεριοχών ως οι μέσοι όροι των άκρων τους.
Έπειτα, έχει δημιουργηθεί κώδικας για την υλοποίηση του **αλγορίθμου Lloyd-Max** (LloydMax). Μέσα στον κώδικα καλείται η συνάρτηση uniform_quantizer, ενώ στη συνέχεια ανάλογα με τα δείγματα που εντοπίζονται σε κάθε υποπεριοχή (quantization interval), υπολογίζεται ο αντίστοιχος μέσος όρος της, ο οποίος ορίζεται πλέον ως το επίπεδο κβάντισης της συγκεκριμένης υποπεριοχής (quantization level). Σύμφωνα με τον αλγόριθμο, η παραπάνω διαδικασία επαναλαμβάνεται μέχρις ότου $|Di - Di{-}1| < \varepsilon$.

Μέρος 1.2 Προσαρμοστική Διαμόρφωσή Δέλτα (ADM)

Για το συγκεκριμένο μέρος έχει υλοποιηθεί η **προσαρμοστική διαμόρφωση Δέλτα** (ADM). Ουσιαστικά, έχει ακολουθηθεί το διάγραμμα που υπάρχει στην εκφώνηση της άσκησης. Για την υλοποίηση του 1 bit Quantizer θεωρείται πως σε περίπτωση που το δείγμα είναι μη αρνητικό αντικαθίσταται από την τιμή 1, ενώ διαφορετικά από την τιμή -1. Η "μονάδα" Step Control Logic υλοποιείται σύμφωνα με την εκφώνηση και την παρακάτω συνάρτηση που δίνεται:

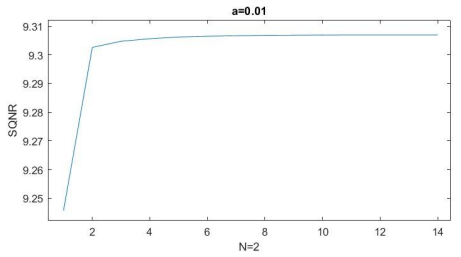
$$\delta(n) = \begin{cases} \delta(n-1)K, & b(n) = b(n-1) \\ \dfrac{\delta(n-1)}{K}, & b(n) \neq b(n-1) \end{cases}$$

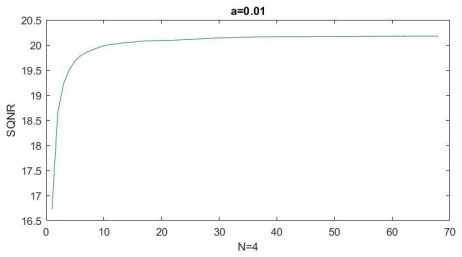
Σημείωση: Έχει υλοποιηθεί η συνάρτηση find_closest_center, η οποία χρησιμοποιείται, προκειμένου κάθε δείγμα να αντιστοιχίζεται στην σωστή υποπεριοχή και στο αντίστοιχο επίπεδο κβαντισμού που

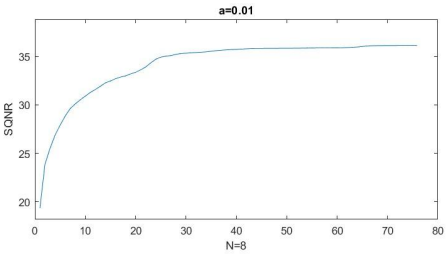
ανήκει. Επιπλέον, έχει ακόμη υλοποιηθεί η συνάρτηση get_actual_prob, προκειμένου να υπολογιστεί η ζητούμενη εντροπία.

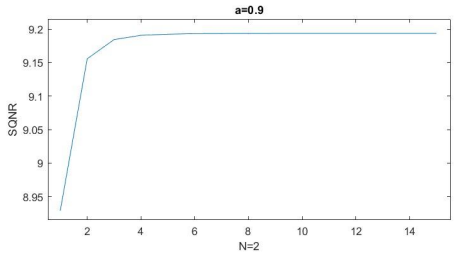
Ερωτήσεις – Ζητούμενα για Μέρος 1

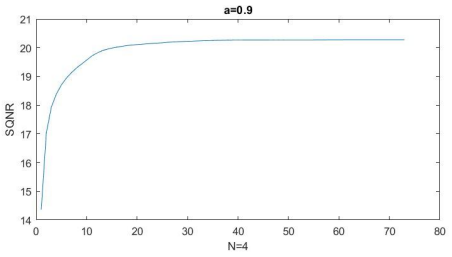
1.1α Παρακάτω παρατίθενται τα διαγράμματα, τα οποία υποδεικνύουν τον τρόπο με τον οποίο μεταβάλλεται το SQNR σε σχέση με τον αριθμό των επαναλήψεων του αλγορίθμου Lloyd-Max.

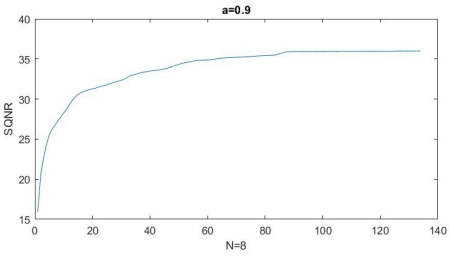






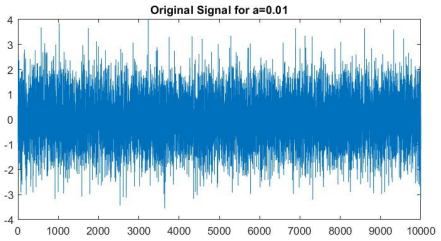


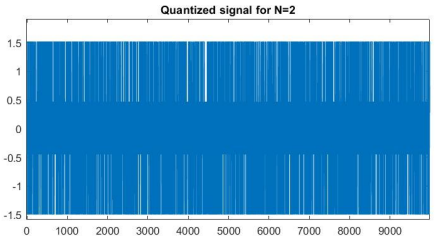


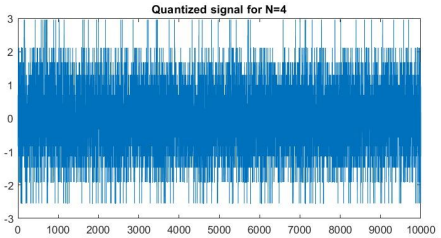


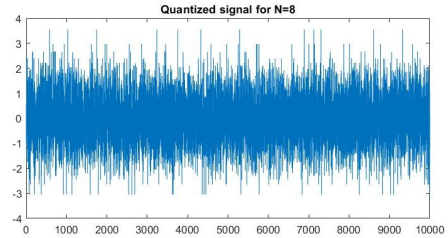
1.1b Παρακάτω παρατίθενται οι κυματομορφές εξόδου για κάθε πηγή AR ξεχωριστά για τα σχήματα PCM και ADM αντίστοιχα.

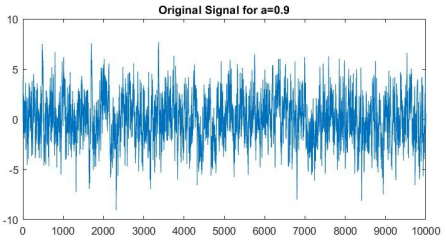
➔ Σχήμα PCM για N = 2, 4 και 8

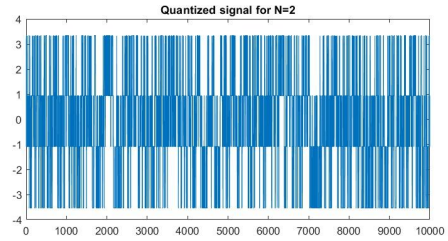


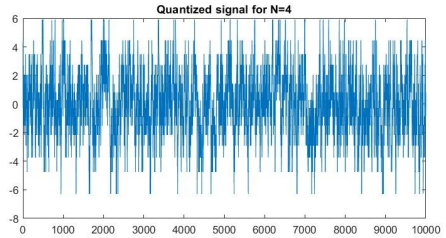


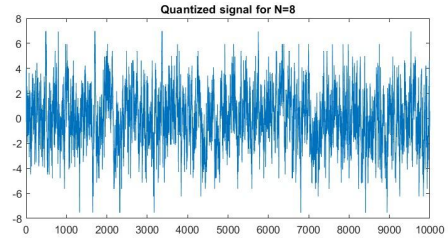




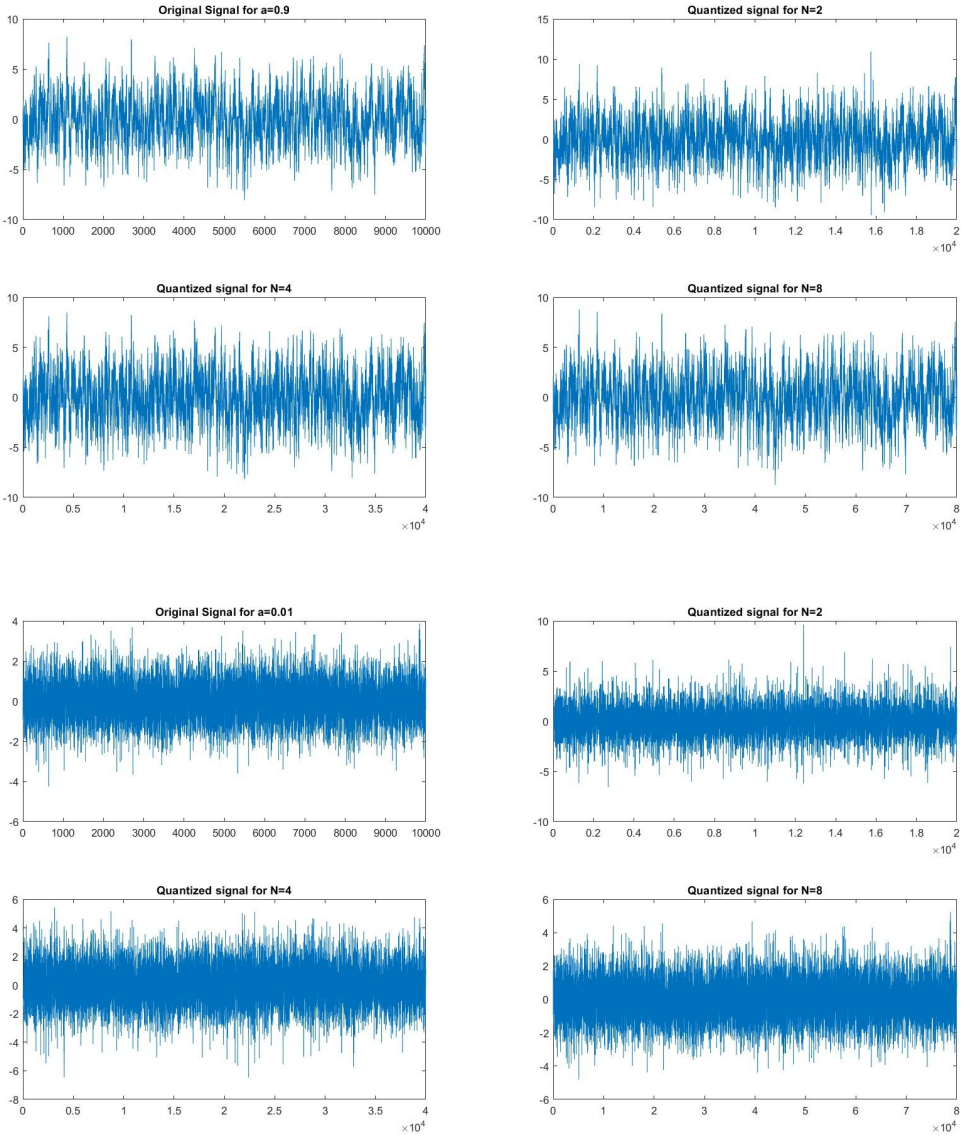








➔ Σχήμα ADM για N = 2, 4 και 8

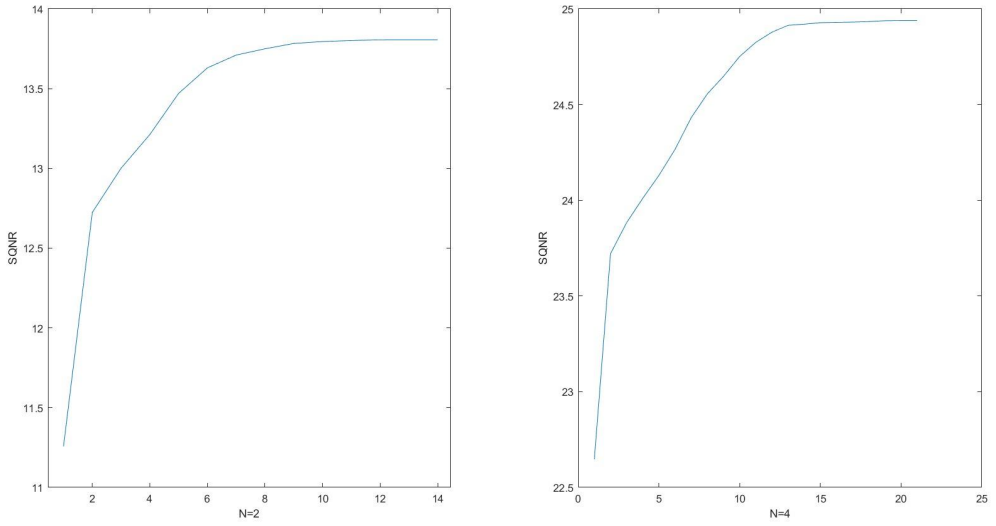


1.2 Για τα σχήματα PCM (N = 2, 4 και 8 bits) έχει υπολογιστεί η εντροπία στην έξοδο του κβαντιστή και τα αποτελέσματα φαίνονται παρακάτω:

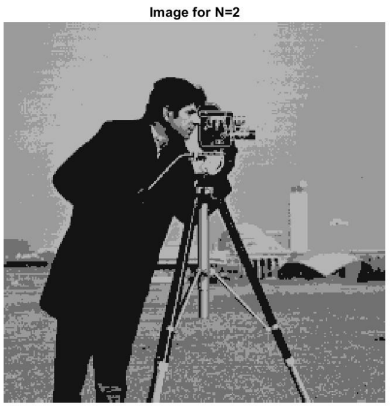
```
LloydMax Entropy for N=2 and a=0.01: 1.9169
LloydMax Entropy for N=4 and a=0.01: 3.8111
LloydMax Entropy for N=8 and a=0.01: 7.8489
-----------------------------------------
LloydMax Entropy for N=2 and a=0.9: 1.9148
LloydMax Entropy for N=4 and a=0.9: 3.791
LloydMax Entropy for N=8 and a=0.9: 7.8994
```

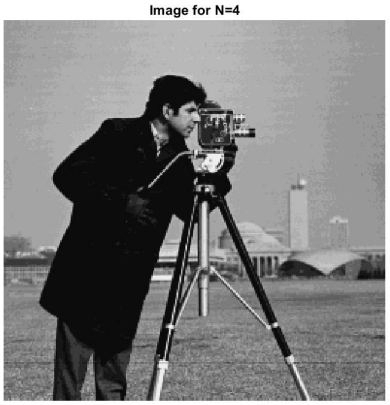
1.3 Σύμφωνα με τα παραπάνω αποτελέσματα, δηλαδή τις κυματομορφές εξόδου των σχημάτων PCM για N = 2, 4 και 8 bits και ADM και σε συνδυασμό με τα αποτελέσματα SQNR, γίνεται αντιληπτό πως το PCM όσον αφορά στην κωδικοποίηση με N = 2 bits, πρόσθεσε αρκετή παραμόρφωση στην τελική έξοδο με αποτέλεσμα να εντοπίζεται μεγάλη απόκλιση από την αρχική πηγή εισόδου. Αντιθέτως, το σχήμα PCM για N = 4 bits και το σχήμα ADM επέστρεψαν αποτελέσματα που είχαν μικρή διαφορά σε σύγκριση με την αρχική πηγή. Όσον αφορά στο σχήμα PCM για N = 8 bits, παρατηρείται πως είναι αυτό που έχει την μικρότερη επίδραση θορύβου, αφού το τελικό αποτέλεσμα είναι πιο ακριβές και πιο κοντά με την αρχική πηγή εισόδου.
Λαμβάνοντας υπόψιν λοιπόν τα παραπάνω αποτελέσματα επαληθεύεται το πόρισμα, σύμφωνα με το οποίο κατά την κωδικοποίηση μίας εισόδου, ο αριθμός των bits που χρησιμοποιούνται είναι αντιστρόφως ανάλογος της παραμόρφωσης που εντοπίζεται στο τελικό αποτέλεσμα.

2.a Παρακάτω παρατίθεται το διάγραμμα, το οποίο υποδεικνύει τον τρόπο με τον οποίο μεταβάλλεται το SQNR σε σχέση με τον αριθμό των επαναλήψεων του αλγορίθμου Lloyd-Max για την πηγή εισόδου.



2.b Παρακάτω παρατίθενται οι εικόνες μετά την έξοδο τους από τον κβανιστή.





Συμπερασματικά, με βάση την ποιότητα των τελικών εικόνων και τις τιμές SQNR, το σχήμα PCM για N = 2 bits αύξησε την επίδραση του θορύβου στην εικόνα, με αποτέλεσμα η τελευταία να είναι θολή, δυσανάγνωστη και ξεθωριασμένη σε σύγκριση με την αρχική. Όσον αφορά στο σχήμα PCM για N = 4 bit, η επίδραση του θορύβου είναι αρκετά μικρότερη, καθιστώντας την παραμόρφωση στην τελική εικόνα μηδαμινή. Στη συγκεκριμένη κωδικοποίηση, το τελικό αποτέλεσμα μοιάζει σε μεγάλο βαθμό με την αρχική εικόνα. Παρομοίως, και σε αυτή την περίπτωση επαληθεύεται το πόρισμα που αναφέρθηκε στο ερώτημα 1.

2.2 Για τα σχήματα PCM (N = 2 και 4 bits) έχει υπολογιστεί η εντροπία στην έξοδο του κβαντιστή και τα αποτελέσματα φαίνονται παρακάτω:

```
>> main2_2
LloydMax Entropy for N=2: 1.9126
LloydMax Entropy for N=4: 3.7998
```

ΜΕΡΟΣ Β

1. Τα βασικά στοιχεία του συστήματος **M-PAM** περιγράφονται αναλυτικά παρακάτω. Αρχικά, δημιουργείται ένα μητρώο διαστάσεων 1xLb, το οποίο αποτελείται από bits 0 και 1. Στη συνέχεια, το παραπάνω μητρώο κωδικοποιείται μέσω του Mapper σε απλή κωδικοποίηση ή Gray κωδικοποίηση, δημιουργώντας τα αντίστοιχα σύμβολα από $\log_2 M$ εκφράσεις bit κάθε φορά. Σειρά έχει ο διαμορφωτής, ο οποίος πολλαπλασιάζει κάθε σύμβολο με τον ορθογώνιο παλμό ενώ ταυτόχρονα το μεταφέρει στη ζώνη μετάδοσης. Κάθε σύμβολο δειγματοληπτείται 40 φορές (Tsymbol/Tsample) σύμφωνα με τη περίοδο του συμβόλου, συνεπώς προκύπτει το αντίστοιχο διάνυσμα. Μετά την ολοκλήρωση της δειγματοληψίας, προστίθεται λευκός γκαουσιανός θόρυβος. Έπειτα, αφού ολοκληρωθεί η λήψη των δειγμάτων από το κανάλι μετάδοσης, το παραπάνω διάνυσμα δέχεται επεξεργασία από τον αποδιαμορφωτή, με αποτέλεσμα να δημιουργηθεί ένα νέο διάνυσμα, το οποίο περιέχει την εκτιμηθείσα τιμή για κάθε σύμβολο. Τα σύμβολα αυτά, με τη βοήθεια του φωρατή, αντιστοιχίζονται στα σύμβολα

με τη μικρότερη διαφορά σύμφωνα με τη δεκαδική τιμή των συμβόλων που χρησιμοποιήθηκαν κατά τη διαδικασία της κωδικοποίησης. Τέλος, το διάνυσμα αποκωδικοποιείται με βάση τον Demapper και την αρχική κωδικοποίηση, με αποτέλεσμα να προκύψει μητρώο όπου κάθε γραμμή του περιέχει την αλληλουχία από bits που αντιστοιχεί στο κάθε σύμβολο.

Για το σύστημα στη παρούσα αναφορά έχουν υλοποιήθηκαν οι παρακάτω συναρτήσεις:

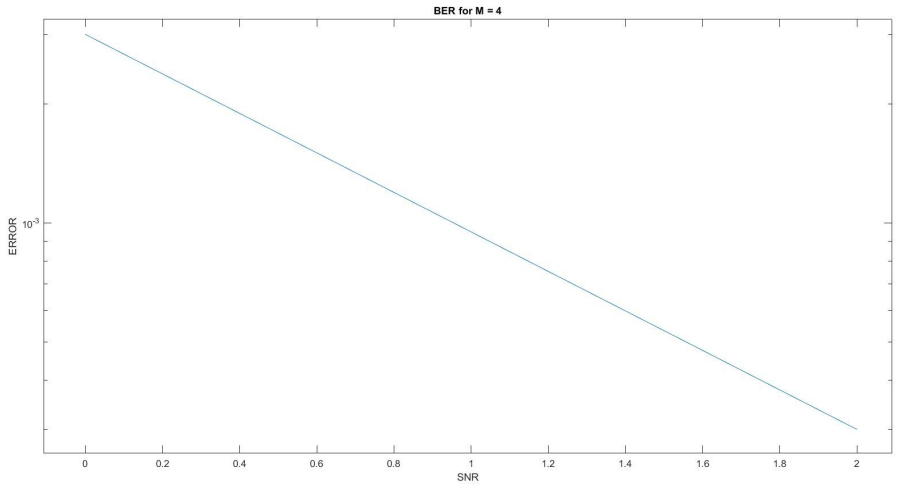
- Mapper → Μετατρέπει το stream των bits που δίνεται ως είσοδο σε μία ακολουθία από σύμβολα. Έχει χρησιμοποιηθεί η συνάρτηση της matlab, **bi2de**, η οποία μετατρέπει δυαδικές εκφράσεις στις αντίστοιχες του δεκαδικού συστήματος καθώς και η συνάρτηση **pammod**
- Modulator → Πολλαπλασιάζει κάθε σύμβολο που έχει προκύψει από την παραπάνω συνάρτηση με τον ορθογώνιο παλμό και την ημιτονοειδή συνάρτηση cos, λαμβάνοντας υπόψιν τις χρονικές μονάδες προσομοίωσης που παρέχονται από την εκφώνηση της άσκησης
- AWGN → Προσθήκη λευκού γκαουσιανού θορύβου μέσω της χρήσης της συνάρτησης matlab, **awgn**
- Demodulator → Ομαδοποιεί τις τιμές που έχουν προκύψει αφού έχει πραγματοποιηθεί η δειγματοληψία κάθε συμβόλου, τις πολλαπλασιάζει με τη φέρουσα και τον ορθογώνιο παλμό και τέλος τις προσθέτει μεταξύ τους
- Demapper → Χρησιμοποιείται η συνάρτηση της matlab, **pamdemod** καθώς και η συνάρτηση **de2bi**, η οποία μετατρέπει κάθε σύμβολο στην αντίστοιχη ακολουθία $\log_2 M$ δυαδικών ψηφίων

Για τον υπολογισμό των σφαλμάτων που ζητούνται, έχουν ληφθεί υπόψιν οι παρακάτω τύποι:

$$\mathrm{SER} = \frac{\text{No. of symbols in error}}{\text{Total no. of transmitted symbols}}$$

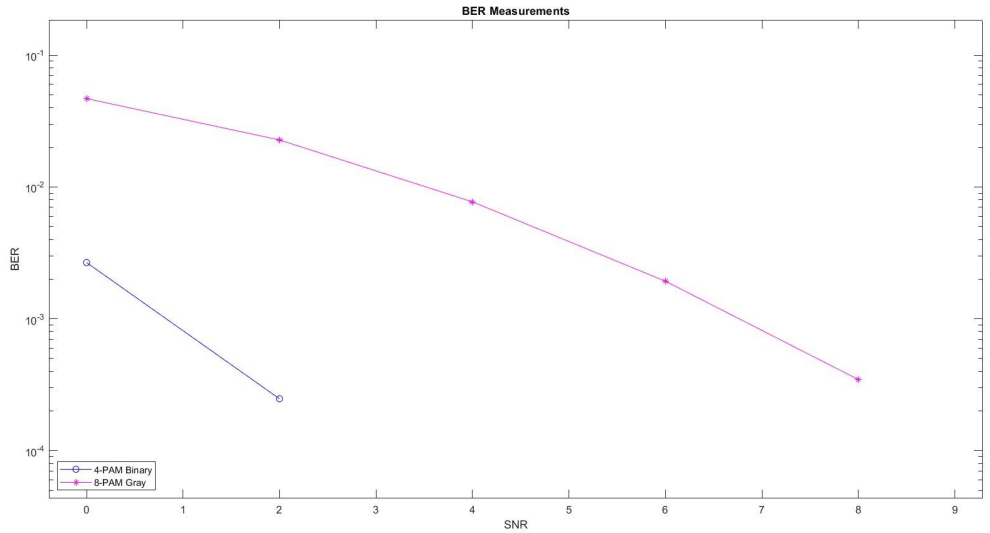
$$\mathrm{BER} = \frac{\text{No. of bits in error}}{\text{Total no. of transmitted bits}}$$

2. Παρακάτω έχει σχεδιαστεί η πιθανότητα σφάλματος bit (BER) για M = 4 - σε λογαριθμική κλίμακα - για απλή κωδικοποίηση (bin) και τιμές SNR = 0:2:20dB.

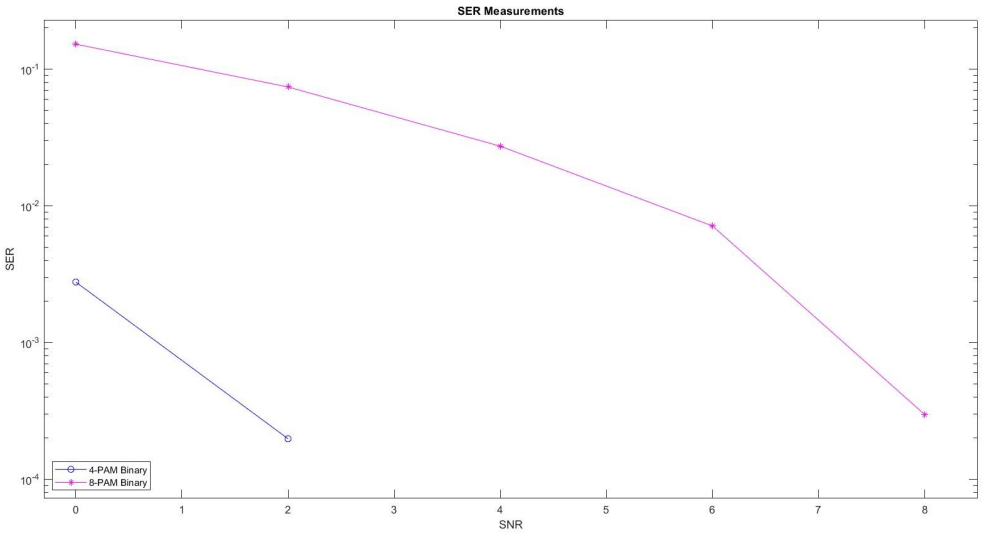


3. Η **κωδικοποίηση Gray** εφαρμόζεται με τέτοιο τρόπο, ώστε τα γειτονικά διαδοχικά σύμβολα να διαφέρουν μόνο κατά ένα bit. Παρακάτω αναφέρεται ο λόγος που η συγκεκριμένη κωδικοποίηση έχει νόημα να χρησιμοποιηθεί στη διαμόρφωση M-PAM.
   Γνωρίζουμε πως κατά τη μετάδοση ενός σήματος προστίθεται θόρυβος με αποτέλεσμα ο δέκτης να λαμβάνει σύμβολα συνοδευόμενα και από θόρυβο. Συνεπώς, αυτό σημαίνει πως το ληφθέν σημείο μετατοπίζεται στη γεωμετρική αναπαράσταση του δέκτη. Η μετατόπιση αυτή βέβαια συνήθως πραγματοποιείται στη γειτονιά του πραγματικού συμβόλου που έχει σταλεί εξαρχής. Ωστόσο, δεν αλλάζει το γεγονός πως ο δέκτης μπορεί να λάβει εσφαλμένη απόφαση για το σύμβολο που στάλθηκε. Άρα, με τη χρήση της παραπάνω κωδικοποίησης προκύπτει πως μία εσφαλμένη απόφαση συνεπάγεται ότι ένα μόνο bit θα είναι λάθος (και επομένως BER < SER).

4. Παρακάτω έχουν σχεδιαστεί στο ίδιο γράφημα η πιθανότητα σφάλματος bit (BER) για M = 4 και 8 - σε λογαριθμική κλίμακα - για απλή κωδικοποίηση (bin) και Gray κωδικοποίηση και τιμές SNR = 0:2:20dB.



5. Παρακάτω έχουν σχεδιαστεί στο ίδιο γράφημα η πιθανότητα σφάλματος συμβόλου (SER) για M = 4 και 8 - σε λογαριθμική κλίμακα - για απλή κωδικοποίηση (bin) και τιμές SNR = 0:2:20dB.



Παράρτημα

Μέρος Α

1. sourceA

```
function x = sourceA(L, a1)
  b = 1;
  k = randn(L, 1);
  a = transpose([b, -a1]);
  x = filter(b, a, k);
end
```

2. sourceB

```
function y = sourceB()
  img = load ("cameraman.mat");
```

```
  img = img.i;
  y = img(:);
  y = (y-128)/128;
end
```

3. find_closest_center

```
function index = find_closest_center(centers, value)
  % initialize the min variable and the needed output -> position
  % with the first index of the centers matrix, so I have an output in
  % case that the if won't activate
  min = abs(centers(1)-value);
  index = 1;
  for i = 2:length(centers)
    if abs(centers(i)-value) < min
      min = abs(centers(i) - value);
      index = i;
    end
  end
end
```

4. uniform_quantizer

```
function [Xq, centers] = uniform_quantizer(x, N, min_value, max_value)
  % normalize the range of the signal
  x(x > max_value) = max_value;
  x(x < min_value) = min_value;

  % initialize the output
  Xq = zeros(length(x), 1);

  % find the step that will define the quantization
  % interval between max_value and min_value
  step = (max_value-min_value)/2^N;

  % initialize the matrix in which the quantization
  % levels will be stored
  centers = zeros(2^N, 1);

  % calculate the first interval
  left = min_value;
  right = min_value + step;

  % find the total quantization levels
  for i = 1:length(centers)
    centers(i) = (left + right) / 2;
```

```
        left = left + step;
        right = right + step;
    end

    % every element of Xq contains the index of the centers matrix
    % that indicates in which quantization interval it belongs to
    for i = 1:length(x)
        Xq(i) = find_closest_center(centers, x(i));
    end
end
```

5. LloydMax

```
function [Xq, centers, D] = LloydMax(x, N, min_value, max_value)
    % call the function uniform_quantizer for the step of the algorithm
    [~, centers] = uniform_quantizer(x, N, min_value, max_value);
    % initialize the matrix that will contain the distortion
    D = [];
    % repeat steps 2, 3 and 4 as long as the below condition is true
    while length(D) < 2 || abs(D(end) - D(end-1)) >= eps
        distortion = 0;
        % initialize the matrices needed
        sum_var = zeros(length(centers), 1);
        Xq = zeros(length(x), 1);
        counter_var = zeros(length(centers), 1);

        for k=1:length(x)
            % find the quantization level that the element x(k) belongs to
            % and return the corresponding index of the centers matrix
            index = find_closest_center(centers, x(k));
            % every element of the counter_val matrix is an index for the centers
            % matrix which represents each quantization level and it adds
            % the current quantization level with every element of the
            % source x
            sum_var(index) = sum_var(index) + x(k);
            % every element of the counter_val matrix is an index for the centers
            % matrix which represents each quantization level, so counter_val
            % counts how many times, the quantization level has appeared
            counter_var(index) = counter_var(index) + 1;
            % calculates every time the current distortion
            distortion = distortion + (x(k) - centers(index))^2;
            Xq(k) = index;
        end

        % calculates the final distortion after each iteration of the
        % algorithm has been completed
        D(end+1) = distortion/length(x);
```

```
        % calculates the new quantization levels after each iteration of
        % the algorithm has been completed
        for k=1:length(centers)
            if counter_var(k) ~= 0
                centers(k) = sum_var(k)/counter_var(k);
            end
        end
    end
end
```

6. ADM

```
function [Xqn] = ADM(Xn, M)

    K = 1.5; % Constant K based on the exercise
    Xn = interp(Xn, M); % Over-Sampling Xn based on the exercise
    step = zeros(1, length(Xn)); % Initialize step
    step(1) = 0.001; % initialize using a really small step
    % Initialize the matrices needed based on
    % exercise's figure
    En = zeros(1, length(Xn));
    Bn = zeros(1, length(Xn));
    Bn(1) = sign(Xn(1));
    Eqn = zeros(1, length(Xn));
    Xqn = zeros(1, length(Xn));
    Xqn(1) = Xn(1);
    Delay = zeros(1, length(Xn));
    Delay(1) = Xn(1);
    % Encoder Implementation
    for index = 2:length(Xn)
        En(index) = Xn(index) - Delay(index-1);
        % 1 bit Quantizer implementation
        if En(index)>= 0
            Bn(index) = 1;
        else
            Bn(index) = -1;
        end
        % Step Control Logic implementation
        if (Bn(index) == Bn(index-1))
            step(index) = step(index-1) * K;
        else
            step(index) = step(index-1) / K;
        end
        Eqn(index) = step(index) * Bn(index);
        Xqn(index) = Eqn(index) + Delay(index-1);
        Delay(index) = Xqn(index);
```

```
    end
    % Decoder Implementation
    for index = 2:length(Xn)
        % Step Control Logic
        if (Bn(index) == Bn(index-1))
            step(index) = step(index-1) * K;
        else
            step(index) = step(index-1) / K;
        end
        Eqn(index) = step(index) * Bn(index);
        Xqn(index) = Eqn(index) + Delay(index-1);
    end
end
```

7. get_actual_prob

```
function actual_prob = get_actual_prob(Xq)
    Xq = tabulate(Xq);
    actual_prob = Xq(:, 3) ./ 100;
end
```

8. Ερώτημα 1a

```
a = [0.01, 0.9];
Lb = 10000;
x1 = sourceA(Lb, a(1));
plotIndex = 1;
for N = [2, 4, 8]
    % execute the LloydMax function
    [Xq, centers, D] = LloydMax(x1, N, -2, 2);
    SQNR1 = 10*log10(mean(x1.^2)./D);

    figure(1)
    subplot(2, 2, plotIndex)
    plot(SQNR1)
    title(['a=' num2str(a(1))])
    xlabel(['N=' num2str(N)])
    ylabel('SQNR')
    plotIndex = plotIndex + 1;
end
x2 = sourceA(Lb, a(2));
plotIndex = 1;
for N = [2, 4, 8]
    % execute the LloydMax function
    [Xq, centers, D] = LloydMax(x2, N, -4, 4);
    SQNR2 = 10*log10(mean(x2.^2)./D);
    figure(2)
    subplot(2, 2, plotIndex)
```

```
  plot(SQNR2)
  title(['a=' num2str(a(2))])
  xlabel(['N=' num2str(N)])
  ylabel('SQNR')
  plotIndex = plotIndex + 1;
end
```

9. Ερώτημα 1b (Γραφικές Παραστάσεις για PCM)

```
% initialize a matrix for the coefficients
a = [0.01, 0.9];
Lb = 10000;
x1 = sourceA(Lb, a(1));
for N = [2, 4, 8]
  % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x1, N, -2, 2);
  figure(1)
  subplot(2,2,1)
  plot(x1)
  title('Original Signal for a=0.01')

  if N==2
     subplot(2,2,2)
     plot(centers(Xq))
     title('Quantized signal for N=2')
  elseif N==4
     subplot(2,2,3)
     plot(centers(Xq))
     title('Quantized signal for N=4')
  else
     subplot(2,2,4)
     plot(centers(Xq))
     title('Quantized signal for N=8')
  end

end
x2 = sourceA(Lb, a(2));
for N = [2, 4, 8]
   % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x2, N, -4, 4);
  figure(2)
  subplot(2,2,1)
  plot(x2)
  title('Original Signal for a=0.9')

  if N==2
     subplot(2,2,2)
     plot(centers(Xq))
     title('Quantized signal for N=2')
  elseif N==4
     subplot(2,2,3)
```

```
        plot(centers(Xq))
        title('Quantized signal for N=4')
    else
        subplot(2,2,4)
        plot(centers(Xq))
        title('Quantized signal for N=8')
    end
end
```

10. Ερώτημα 1.2

```
% initialize a matrix for the coefficients
a = [0.01, 0.9];
Lb = 10000;
entropyA1 = zeros(3, 1);
indexA1 = 1;
entropyA2 = zeros(3, 1);
indexA2 = 1;
x1 = sourceA(Lb, a(1));
for N = [2, 4, 8]
  % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x1, N, -2, 2);

  probA1 = get_actual_prob(Xq);
  probsA1 = probA1(probA1~=0);
  entropyA1(indexA1) = -sum(probsA1.*log2(probsA1));
  fprintf("LloydMax Entropy for N=%s and a=%s: %s\n", num2str(N), num2str(a(1)),
num2str(entropyA1(indexA1)));
  indexA1 = indexA1 + 1;
end
x2 = sourceA(Lb, a(2));
fprintf("-----------------------------------------\n");
for N = [2, 4, 8]
   % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x2, N, -4, 4);

  probA2 = get_actual_prob(Xq);
  probsA2 = probA2(probA2~=0);
  entropyA2(indexA2) = -sum(probsA2.*log2(probsA2));
  fprintf("LloydMax Entropy for N=%s and a=%s: %s\n", num2str(N), num2str(a(2)),
num2str(entropyA2(indexA2)));
  indexA2 = indexA2 + 1;
end
```

11. Ερώτημα 2a

```
x = sourceB();
plotIndex = 1;
for N = [2, 4]
  % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x, N, -1, 1);
  SQNR = 10*log10(mean(x.^2)./D);
  figure(1)
  subplot(1,2, plotIndex)
  plot(SQNR)
  xlabel(['N=' num2str(N)])
  ylabel('SQNR')
  plotIndex = plotIndex + 1;
end
```

12. Ερώτημα 2b

```
x = sourceB();
for N = [2, 4]
  % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x, N, -1, 1);
  img = centers(Xq);
  img = 128*img + 128;
  image = reshape(img, 256, 256);
  imshow(uint8(image));
  caption = sprintf('Image for N=%s', num2str(N));
  title(caption, 'FontSize', 14);
  drawnow;
end
```

13. Ερώτημα 2.2

```
entropy = zeros(2, 1);
index = 1;
x = sourceB();
for N = [2, 4]
  % execute the LloydMax function
  [Xq, centers, D] = LloydMax(x1, N, -1, 1);

  prob = get_actual_prob(Xq);
  probs = prob(prob~=0);
  entropy(index) = -sum(probs.*log2(probs));
  fprintf("LloydMax Entropy for N=%s: %s\n", num2str(N), num2str(entropy(index)));
  index = index + 1;
end
```

14. Ερώτημα 1b (Γραφικές Παραστάσεις για ADM)

```
% initialize a matrix for the coefficients
a = [0.01, 0.9];
Lb = 10000;
x1 = sourceA(Lb, a(1));
for N = [2, 4, 8]
  % execute the ADM function
  [Xqn] = ADM(x1, N);
  figure(1)
  subplot(2,2,1)
  plot(x1)
  title('Original Signal for a=0.01')
  if N==2
     subplot(2,2,2)
     plot(Xqn)
     title('Quantized signal for N=2')
  elseif N==4
     subplot(2,2,3)
     plot(Xqn)
     title('Quantized signal for N=4')
  else
     subplot(2,2,4)
     plot(Xqn)
     title('Quantized signal for N=8')
  end
end

x2 = sourceA(Lb, a(2));
for N = [2, 4, 8]
  % execute the ADM function
  [Xqn] = ADM(x2, N);
  figure(2)
  subplot(2,2,1)
  plot(x2)
  title('Original Signal for a=0.9')

  if N==2
     subplot(2,2,2)
     plot(Xqn)
     title('Quantized signal for N=2')
  elseif N==4
     subplot(2,2,3)
     plot(Xqn)
     title('Quantized signal for N=4')
  else
     subplot(2,2,4)
     plot(Xqn)
     title('Quantized signal for N=8')
  end
end
```

Μέρος Β

15. source

```
function y = source(Lb)
  bits = [0 1];
  y = randsrc(1, Lb, bits);
end
```

16. mapper

```
function symbols = mapper(symbols_order, signal, M)
  size = log2(M);
  % matrix that contains the symbols that will be produced
  val_decimal = zeros(1, length(signal)/size);
  for i = 1:length(val_decimal)
     % calculate the indexes in every iteration in order to get
     % each time every log2(M) bits from the signal matrix
     start_index = (i-1) * size + 1;
     end_index = i * size;
     % transform every log2(M) bits to the corresponding symbol
     value = bi2de(signal(start_index:end_index), 'left-msb');
     % add to the symbols matrix
     val_decimal(i) = value;
  end
  % using the matlab's function pammod to specify the type of code
  % mapping (gray or binary) that will be applied
  symbols = real(pammod(val_decimal, M, 0, symbols_order));
end
```

17. modulator

```
function sm = modulator(symbols)
  % initialize the time units needed
     Tsymbol = 4 * 10^(-6);
     fc = 2.5 * 10^6;
     Tsample = 10^(-7);
     % rectangular pulse
     g = sqrt(2/Tsymbol);
     % initialize sampling
     samples = Tsymbol / Tsample;
     sm = zeros(samples*length(symbols), 1);
     t = (1:samples) * Tsample;
     for i = 1:length(symbols)
        s = symbols(i) * g * cos(2*pi*fc*t);
```

```
        start_index = (i-1) * samples + 1;
        end_index = i * samples;
        sm(start_index:end_index) = s;
    end
end
```

18. AWGN

```
% apply noise using matlab's function
function noise = AWGN(sm, SNR)
  noise = awgn(sm, SNR, 'measured');
end
```

19. demodulator

```
function rt = demodulator(received)
  % initialize the time units needed
  Tsymbol = 4 * 10^(-6);
  fc = 2.5 * 10^6;
  Tsample = 10^(-7);
  % rectangular pulse
  g = sqrt(2/Tsymbol);
  samples = Tsymbol / Tsample;
  rt = zeros(length(received)/samples, 1);
  t = (1:samples) * Tsample;
  for i = 1:length(rt)
     start_index = (i-1) * samples + 1;
     end_index = i * samples;
     temp_y = received(start_index:end_index).*g.*cos(2*pi*fc*t)';
     rt(i) = sum(temp_y) * Tsample;
  end
end
```

20. demapper

```
function signal = demapper(symbols_order, symbols, M)
  size = log2(M);
  % specifies the type of coding applied to the binary words
  val_decimal = pamdemod(symbols, M, 0, symbols_order);
  signal = zeros(1, length(symbols)/size);
  for i = 1:length(val_decimal)
     start_index = (i-1) * size + 1;
     end_index = i * size;
     % converts the integers to binary
```

```
        value = de2bi(val_decimal(i), size, 'left-msb');
        signal(start_index:end_index) = value;
    end
end
```

21. Ερωτήματα 2 & 4

```
Lb = 20232;
source_stream = source(Lb);
SNRs = 0:2:20;
% 4-PAM and Bin coding
M = 4;
symbols_order = 'bin';
symbols = mapper(symbols_order, source_stream, M);
sm = modulator(symbols);
BERs1 = zeros(1, length(SNRs));
for j = 1:length(SNRs)
    SNR1 = SNRs(j);
    noise = AWGN(sm, SNR1);
    rt = demodulator(noise);
    source_output = demapper(symbols_order, rt, M);
    BERs1(j) = sum(source_stream - source_output~=0)/Lb;
end
% ------------------------------------------------------------
% 8-PAM and Gray coding
M = 8;
symbols_order = 'gray';
symbols = mapper(symbols_order, source_stream, M);
sm = modulator(symbols);
BERs2 = zeros(1, length(SNRs));
for j = 1:length(SNRs)
    SNR2 = SNRs(j);
    noise = AWGN(sm, SNR2);
    rt = demodulator(noise);
    source_output = demapper(symbols_order, rt, M);
    BERs2(j) = sum(source_stream - source_output~=0)/Lb;
end
figure()
sml1 = semilogy(SNRs, BERs1, '-ob');
hold on;
sml2 = semilogy(SNRs, BERs2, '-m*');
hold off;
title('BER Measurements')
xlabel('SNR')
ylabel('BER')
```

```
legend([sml1(1) sml2(1)], {'4-PAM Binary ', '8-PAM Gray'}, 'Location', 'southwest',
'NumColumns', 1);
```

22. Ερώτημα 5

```
Lb = 20232;
source_stream = source(Lb);
SNRs = 0:2:20;
symbols_order = 'bin';
% 4-PAM and Bin coding
M = 4;
symbols = mapper(symbols_order, source_stream, M);
sm = modulator(symbols);
SERs1 = zeros(1, length(SNRs));
for j = 1:length(SNRs)
  SNR1 = SNRs(j);
  noise = AWGN(sm, SNR1);
  rt = demodulator(noise);
  source_output = demapper(symbols_order, rt, M);
  symbols_found = mapper(symbols_order, source_output, M);
  SERs1(j) = sum(symbols - symbols_found~=0)/length(symbols_found);
end
% ------------------------------------------------------------
% 8-PAM and Bin coding
M = 8;
symbols = mapper(symbols_order, source_stream, M);
sm = modulator(symbols);
SERs2 = zeros(1, length(SNRs));
for j = 1:length(SNRs)
  SNR2 = SNRs(j);
  noise = AWGN(sm, SNR2);
  rt = demodulator(noise);
  source_output = demapper(symbols_order, rt, M);
  symbols_found = mapper(symbols_order, source_output, M);
  SERs2(j) = sum(symbols - symbols_found~=0)/length(symbols_found);
end
figure()
sml1 = semilogy(SNRs, SERs1, '-ob');
hold on;
sml2 = semilogy(SNRs, SERs2, '-m*');
hold off;
title('SER Measurements')
xlabel('SNR')
ylabel('SER')
legend([sml1(1) sml2(1)], {'4-PAM Binary ', '8-PAM Binary'}, 'Location', 'southwest',
'NumColumns', 1);
```