Ψηφιακές Τηλεπικοινωνίες

Ρουχωτά Ειρήνη
ΑΜ**: 1059654**

2020-2021

Α ΜΕΡΟΣ ΕΡΓΑΣΙΑΣ – Huffman Code

Τι είναι ο κώδικας Huffman;

- Ο κώδικας Huffman είναι μια μέθοδος συμπίεσης τόσο για εικόνες όσο και για αρχεία κειμένου. Η επιτυχία του έγκειται στον ***προθεματικό*** κώδικα, ο οποίος παράγεται. Πιο συγκεκριμένα, κάθε σύμβολο του αλφαβήτου μιας διακριτής πηγής αντιστοιχίζεται σε μια ακολουθία από δυαδικά ψηφία. Έτσι προκύπτουν οι κωδικές λέξεις. Για μια πηγή με N σύμβολα, η διαδικασία του αλγορίθμου θα επαναληφθεί N-1 φορές. Η ιδέα είναι ότι σε κάθε βήμα γίνεται αναδιάταξη των συμβόλων με φθίνουσα πιθανότητα εμφάνισης τους. Στα λιγότερο δύο πιθανά σύμβολα αντιστοιχίζεται ένα δυαδικό σύμβολο στο καθένα και στο επόμενο βήμα του ενοποιούνται σαν ένα σύμβολο. Η πιθανότητα εμφάνισης του ενοποιημένου συμβόλου στο επόμενο βήμα ισούται με το άθροισμα των πιθανοτήτων των δύο αυτών συμβόλων. Η κωδική λέξη προκύπτει στο τέλος παίρνοντας με αντίστροφη σειρά τις αναθέσεις δυαδικών συμβόλων που προέκυψαν σε κάθε βήμα. Με αυτό λοιπόν τον τρόπο, πετυχαίνουμε χαμηλό μέσο μήκος.

ΑΠΑΝΤΗΣΕΣ

1. α. Σκοπός είναι να κατασκευαστεί η αντίστοιχη συνάρτηση της MATLAB huffmandict().

   ```
   [dict,avglen] = huffmandict(symbols,prob)
   ```

   Είσοδος: το διάνυσμα του αλφαβήτου εισόδου, πιθανότητα εμφάνισης κάθε συμβόλου. Τα μήκη των δύο διανυσμάτων πρέπει να είναι ίδια

   Έξοδος: λεξικό δυαδικού κώδικας Huffman, μέσο μήκος κώδικα του λεξικού σύμφωνα με τη πιθανότητα εμφάνισης του κάθε συμβόλου εισόδου

   Η υλοποίηση της συνάρτησης αυτής βρίσκεται στο αρχείο ***myhuffmandict*** ***εδώ*** με αντίστοιχα σχόλια στον κώδικα για περαιτέρω ανάλυση.

β. Η huffmanenco() κωδικοποιεί μία ακολουθία από σύμβολα σε δυαδικά ψηφία.

```
code = huffmanenco(sig,dict)
```

Είσοδος: ακολουθία λέξεων, κωδικές λέξεις που έχουν παραραχθεί από την huffmandict και βρίσκονται αποθηκευμένες στο dict

Έξοδος: ο κωδικοποιημένος κώδικας, όπου κάθε σύμβολο έχει αντιστοιχηθεί σε μία κωδική λέξη

Η υλοποίηση της συνάρτησης αυτής βρίσκεται στο αρχείο ***myhuffmanenco*** **εδώ** με αντίστοιχα σχόλια στον κώδικα για περαιτέρω ανάλυση.

γ. Η huffmandeco() κάνει την αντίστροφη διαδικασία της huffmanenco(), δηλαδή από μία δοσμένη δυαδική ακολουθία αντιστοιχίζεται σε σύμβολα.

Είσοδος: ακολουθία από bit παραγόμενη από τη συνάρτηση huffmanenco(), το λεξικό dict που είχε παραχθεί από τη huffmandict()

Έξοδος: ο αποκωδικοποιημένος κώδικας, όπου ακολουθία από bits έχει αντιστοιχηθεί σε ένα από τα σύμβολα του αλφαβήτου

Η υλοποίηση της συνάρτησης αυτής βρίσκεται στο αρχείο ***myhuffmandeco*** **εδώ** με αντίστοιχα σχόλια στον κώδικα για περαιτέρω ανάλυση.

Έχει ακόμα υλοποιηθεί το info_table εδώ που περιέχει πληροφορίες όπως η εντροπία και μέσο μήκος κώδικα για τις ανάγκες των υπόλοιπων ερωτημάτων.

2. Ο κώδικας σχετικά με την υλοποίηση του ερωτήματος αυτού βρίσκεται στο αρχείο **erotima2.m. εδώ**

Για τη πηγή Α

- Αλφάβητο εισόδου: οι χαρακτήρες a-z
- Οι αντίστοιχες πιθανότητες ήταν στο αντίστοιχο link της εκφώνησης.
- Στη συνέχεια φτιάχνουμε τη τυχαία πηγή Α με βάση τη randsrc
- Καλούμε την myhuffmandict για να προκύπτει η κωδικοποίηση Huffman, για κάθε σύμβολο του αλφαβήτου μας

Παρατήρηση: Η κωδικοποίηση θεωρείται σωστή, καθώς γράμματα με μεγαλύτερη πιθανότητα εμφάνισης κωδικοποιούνται με λιγότερα bit έναντι αυτών που εμφανίζονται με μικρότερη

- Με την myhuffmanenco, κωδικοποιούμε τη πηγή μας με βάση το λεξικό που παράχθηκε στο παραπάνω βήμα
- Καλούμε την myhuffmandeco, για να αποκωδικοποιήσουμε την παραπάνω κωδικοποίηση
- Αν η πηγή μας η αρχική, ταυτίζεται με την παραπάνω αποκωδικοποίηση, τότε λειτουργεί ορθά η κωδικοποίηση Huffman

Τα αποτελέσματα για την πηγή Α είναι:

```
Huffman code works for sourceA

ans =

  1×4 table

    Entropy    Average Length    Huffman bits    efficiency
    _______    ______________    ____________    __________

    4.1758         4.2051           42050         0.99304
```

Για τη πηγή Β ακολουθήσαμε τα ίδια βήματα με παραπάνω, με τη μόνη διαφορά ότι η πηγή μας τώρα προέρχεται από το kword.txt .

3 Huffman bits	4 efficiency
135482	0.9930

Ένα βασικό συμπέρασμα είναι ότι η εντροπία της κάθε πηγής έχει πολύ κοντινή τιμή με το μέσο μήκος κώδικα. Η αποδοτικότητα του αλγορίθμου και στις δύο περιπτώσεις είναι παραπάνω από 0,99! Τέλος, τα huffman bits που χρειάστηκαν για την κωδικοποίηση της πηγής Β (135482) είναι κατά πολύ περισσότερα από αυτά που χρειάστηκαν για την κωδικοποίηση της πηγής Α (42050). Αυτό είναι λογικό άμα σκεφτεί κανείς ότι οι χαρακτήρες προς κωδικοποίηση στο αρχείο είναι 29384. Οι συναρτήσεις λειτουργούν ορθά και στις δύο περιπτώσεις, αφού η αρχή πηγή ταυτίζεται με την αποκωδικοποιημένη.

3. Στο ερώτημα **εδώ** πρέπει να λάβουμε υπόψιν μας την πιθανότητα εμφάνισης ενώ συμβόλου στηριζόμενοι στο kword.txt, σε αντίθεση με πριν που χρησιμοποιούσαμε την πιθανότητα εμφάνισης κάθε συμβόλου από το link της εκφώνησης. Η κωδικοποίηση είναι αναμενόμενη, αφού σύμβολα με μεγαλύτερη πιθανότητα εμφάνισης κωδικοποιούνται με λιγότερα bits έναντι αυτών με λιγότερη πιθανότητα εμφάνισης που κωδικοποιούνται με περισσότερα bits.

```
Huffman code works for source B

ans =

  1×4 table

    Entropy    Average Length    Huffman bits    efficiency
    _______    ______________    ____________    __________

    4.0482         4.0869         1.1897e+05      0.99053
```

Είναι φανερό ότι τα huffman bits σε αυτή τη περίπτωση είναι 118971 έναντι πριν που ήταν 135482. Αυτό είναι λογικό, αφού τώρα λαμβάνεται υπόψιν το αρχείο κειμένου και έτσι σύμβολα με συχνότερη εμφάνιση αντιστοιχούν σε λιγότερα bit. Επίσης, η εντροπία και σε αυτή τη περίπτωση ισούται με το μέσο μήκος κώδικα.

4. Στο ερώτημα εδώ ζητείται να φτιάξουμε δεύτερη τάξης επέκταση της πηγής Α. Αυτό σημαίνει ότι πρέπει να πάρουμε όλους τους πιθανούς συνδυασμών ζευγών του αγγλικού αλφαβήτου. Άρα τώρα το αλφάβητό μας θα αποτελείται από 676 σύμβολα (26^2). Επίσης οι πιθανότητες αυτών, θα ισούται με την πιθανότητα να εμφανιστεί το ένα σύμβολο επί την πιθανότητα να βρεθεί το δεύτερο σύμβολο του ζευγαριού. Αυτό σημαίνει ότι οι πιθανότητες για κάθε ζεύγος-σύμβολο μειώνονται, σε αντίθεση με το να είχαμε μονά σύμβολα. Έτσι, αφού οι πιθανότητες είναι πολύ μικρές, η κωδικοποίηση για κάθε σύμβολο του καινούργιου αυτού αλφαβήτου αυξάνονται ακόμα και μέχρι 20 bits χρησιμοποιούνται για την κωδικοποίηση ενός ζεύγους.

```
Huffman code works for doubled SourceA

ans =

  1×4 table

    Entropy    Average Length    Huffman bits    efficiency
    _______    ______________    ____________    __________

    8.3517         8.382            41995         0.99638
```

Τα συμπεράσματα είναι:
α. Η εντροπία και το μέσο μήκος κώδικα έχουν κοντινές τιμές
β. Η εντροπία και το μέσο μήκος κώδικα έχουν διπλασιαστεί σε σχέση με τα προηγούμενα ερωτήματα.
γ. Η απόδοση έχει αυξηθεί

δ. Τα bits που χρειάστηκαν για την κωδικοποίηση της πηγής είναι 41995, ενώ στο προηγούμενο ερώτημα ήταν 42050

Το (β) μπορεί να αιτιολογηθεί και από τη θεωρία

$$H(\Phi^n) = n \cdot H(\Phi)$$

Στην περίπτωση μας το n=2 οπότε για αυτό προκύπτει ότι η δευτέρας τάξης εντροπία ισούται με το διπλασιασμό της εντροπίας που είχαμε στο δεύτερο ερώτημα.

Η καλύτερη απόδοση μπορεί να αποδειχθεί και θεωρητικά ξεκινώντας από το γνωστό τύπο: (όπου L το μήκος του προθεματικού κώδικα)

$$H(\Phi) \leq \overline{L} < H(\Phi) + 1$$

Εφαρμόζοντας τον παραπάνω τύπο για n-οστής επέκτασης πηγής:

$$\begin{aligned} H(\Phi^n) \leq \overline{L}_n < H(\Phi^n) + 1 \quad &\Leftrightarrow \\ n \cdot H(\Phi) \leq \overline{L}_n < n \cdot H(\Phi) + 1 \quad &\Leftrightarrow \\ H(\Phi) \leq \frac{\overline{L}_n}{n} < H(\Phi) + \frac{1}{n} & \end{aligned}$$

Έτσι, καθώς το *n* αυξάνει τόσο περισσότερο το μήκος του προθεματικού κώδικα πλησιάζει την εντροπία της πηγής (=βέλτιστο μήκος κώδικα) και η απόδοση της κωδικοποίησης τείνει στο 1 καθώς το n τείνει στο άπειρο.

5. i. Στο ερώτημα **εδώ** ,κωδικοποιώντας τη πηγή B με τις πιθανότητες ζευγών χαρακτήρων του προηγούμενου ερωτήματος παίρνουμε τα εξής αποτελέσματα:

Entropy	Average Length	Huffman bits	efficiency
8.0964	8.124	1.1825e+05	0.9966

Ii. Ενώ κωδικοποιώντας με βάση τις πιθανότητες για ζεύγη χαρακτήρων από το αρχείο κειμένου έχουμε:

Entropy	Average Length	Huffman bits	efficiency
8.1541	8.1839	1.473e+05	0.99636

Είναι φανερό ότι στη (ii) τα huffman bits που χρησιμοποιήθηκαν είναι περισσότερα 147302>118249. Αυτό συνέβη γιατί αυξήθηκε το μήκος των κωδικών λέξεων του dict σε σχέση με το προηγούμενο ερώτημα. Αυτό σημαίνει ότι και μειώθηκαν οι πιθανότητες των ζευγών. Στη (ι) περίπτωση, έχουμε λιγότερα στοιχεία προς κωδικοποίηση. Επίσης, παρατηρούμε ότι στη (ι) περίπτωση έχει αυξηθεί η απόδοση του huffman code, αφού πλησιάζουμε σημαντικά την εντροπία. (η διαφορά μεταξύ εντροπίας είναι 0.0276.) Τέλος, γνωρίζουμε ότι και στις δύο

περιπτώσεις έχει εφαρμοστεί σωστά ο κώδικας Huffman, αφού οι αρχικές πηγές ταυτίζονται με τις αποκωδικοποιημένες. Εμφανίζεται και σχετικό μήνυμα στον κώδικα.

Γ ΜΕΡΟΣ - Μελέτη Απόδοσης Ομόδυνου Ζωνοπερατού Συστήματος M-PAM

Βασικά Σημεία

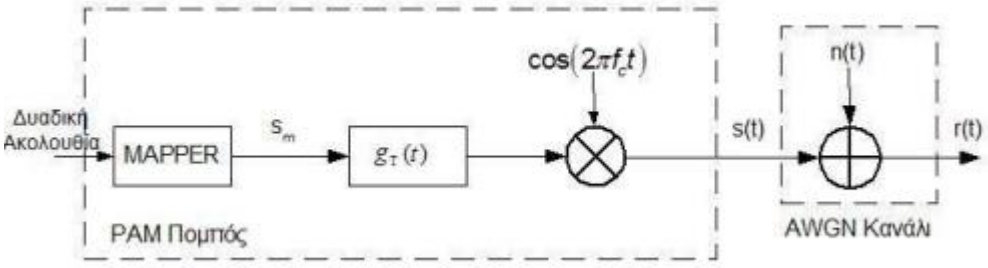
Κατά τη μετάδοση ψηφιακής πληροφορίας, δηλαδή μιας ακολουθίας από bits, μέσα από **AWGN** κανάλι, πρέπει η πληροφορία αυτή να μετατραπεί σε αναλογικές κυματομορφές, μιας και το κανάλι, που είναι πάντα αναλογικό, μπορούν να μεταδοθούν μόνο αναλογικά σήματα. Η αντιστοίχιση των bits σε αναλογικές κυματομορφές ονομάζεται **διαμόρφωση** και γίνεται στον Μ-διαμορφωτή.

Με τον αριθμό **M** δηλώνεται το **πλήθος των συμβόλων** του διαμορφωτή, δηλαδή το πλήθος όλων των διαφορετικών κυματομορφών στις οποίες θα αντιστοιχισθεί η ακολουθία bit , που είναι ως είσοδος σε αυτόν.

Ένα από τα είδη διαμόρφωσης είναι η **διαμόρφωση κατά πλάτος**. (PAM)

Για την M-PAM διαμόρφωση χρησιμοποιείται μια βασική κυματομορφή, η οποία τροποποιείται κατά πλάτος ώστε να δημιουργηθούν οι Μ-κυματομορφές ή αλλιώς τα Μ σύμβολα. Εφαρμόζεται τόσο σε κανάλια βασικής ζώνης όσο και σε ζωνοπερατά κανάλια.

Στο **pam_coding.m** **εδώ** έχει μοντελοποιηθεί το παρακάτω



Είσοδος: Τυχαία ακολουθία από bit π.χ. bit_stream= randsrc(1,10000,[0 1]); , πλήθος συμβόλων του διαμορφωτή (πρέπει πολλαπλάσιο του 2), τιμή μηδέν για μη κωδικοποίηση Gray και 1 για κωδικοποίηση Gray, καθορισμένες τιμές SNR

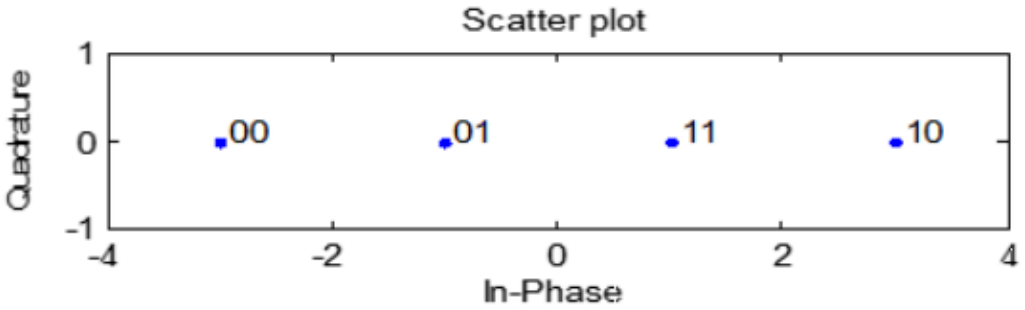
Έξοδος: από το σχήμα την r(t)=s(t)+n(t), τα σύμβολα που έχουν ταιριαστεί με την αρχική ακολουθία bits

Ορίζω χρήσιμες μεταβλητές όπως:

- Eb=1/log2(M) %Ενέργεια ανά bit
- Tsymbol=4; %περίοδος συμβόλου

- Tc=0.4; %φέρουσα περίοδος σε μsec
- fc=1/Tc; %φέρουσα συχνότητα σε μsec
- Tsample=Tc/4; %περίοδος δειγματοληψίας σε μsec
- No=Eb/10^(SNR/10) % No/2 είναι φασματική πυκνότητα ισχύος θορύβου
- g_t=sqrt(2/Tsymbol) %ορθογώνιος παλμός
- Eg=(g_t^2)*Tsymbol; %ενέργεια ορθογώνιου παλμού
- t=0:Tsample:Tsymbol; %άξονας του χρόνου
- A=sqrt(3/(Eg*(M^2-1))) %ενέργεια συμβόλου
- m=(1:M); %διάνυσμα συμβόλων
- sm=(2*m-1-M)*A; %παράγοντας για την μορφή των κυματομορφών

Στη συνέχεια κατασκευάζεται ο mapper με ή χωρίς κωδικοποίηση **Gray.**
Στην κωδικοποίηση Gray, τα **διαδοχικά σύμβολα διαφέρουν** μεταξύ τους κατά **ένα bit**, όπως φαίνεται και παρακάτω



Σχήμα: Διάγραμμα αστερισμού για διαμόρφωση 4-PAM

Η κωδικοποίηση Gray **ελαχιστοποιεί τον αριθμό των λάθος bits**, αφού συνήθως ένα λάθος σύμβολο ανιχνεύεται ως το γειτονικό του. Σε αυτή τη περίπτωση, **κάθε λάθος σύμβολο θα δίνει μόνο ένα λάθος bit.**

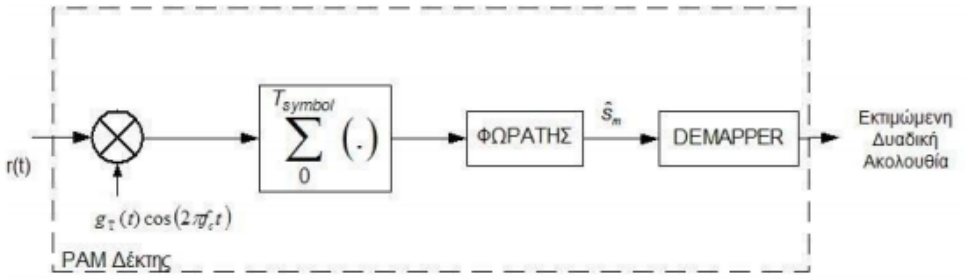
Απαραίτητο επίσης είναι το διάνυσμα **matched_symbols**, το οποίο περιέχει τα σύμβολα που έχουν “ταιριαστεί” με την αρχική ακολουθία εισόδου.

$$s_m(t) = s_m g_T(t) cos(2\pi f_c t),\ \ 0 \le t \le T_{symbol}$$

Τέλος, κατασκευάζεται το σήμα
Προστίθεται σε αυτόν ο **λευκός θόρυβος** μηδενικής μέσης τιμής και διασποράς **No/2.**

noise=random('Normal',0,sqrt(No/2),size(signal));

Στο **pam_decoding.m** **εδώ** έχει μοντελοποιηθεί η αντίστροφη διαδικασία



Είσοδος: Μ πλήθος συμβόλων, 0 ή 1 χωρίς ή με κωδικοποίηση gray , το r(t) από τη προηγούμενη διαδικασία

Έξοδος: εκτιμώμενη δυαδική ακολουθία, εκτιμώμενα σύμβολα

Ακολουθώντας πιστά το σχήμα, κατασκευάζεται το

r(t).*(g_t.*cos(2*pi*fc*t))

Καθώς και ο αθροιστής

sum(r(t).*(g_t.*cos(2*pi*fc*t)) ,2) (1)

Στη συνέχεια υλοποιείται ο φωρατής, ο οποίος αποφασίζει έχοντας ως δεδομένο το (1) σε ποιο σύμβολο βρίσκεται εγγύτερα. Για να το κάνουμε αυτό βρίσκουμε την ελάχιστη τιμή της απόλυτης διαφοράς του sm-(1).
min|sm-r_add(i)|

Έτσι βρίσκουμε το s_hat.

Για τον demapper σημαντικό ρόλο παίζει το **λογικό διάνυσμα** vector, η οποία παίρνει σε κάθε θέση την τιμή 1 αν το εκτιμώμενο σύμβολο υπάρχει στον mapper που είχε κατασκευαστεί νωρίτερα, αλλιώς λαμβάνει την τιμή μηδέν.

Τέλος, έχοντας τώρα τα “ταιριασμένα” σύμβολα, μπορούμε να τα μετατρέψουμε σε διάνυσμα γραμμής, προκειμένου να μπορέσουμε να το συγκρίνουμε με το αρχικό bit_stream. Με αυτό τον τρόπο προκύπτει το bin_est.

Στο **calculate.m** **εδώ** υπολογίζεται

- **Bit Error Rate**

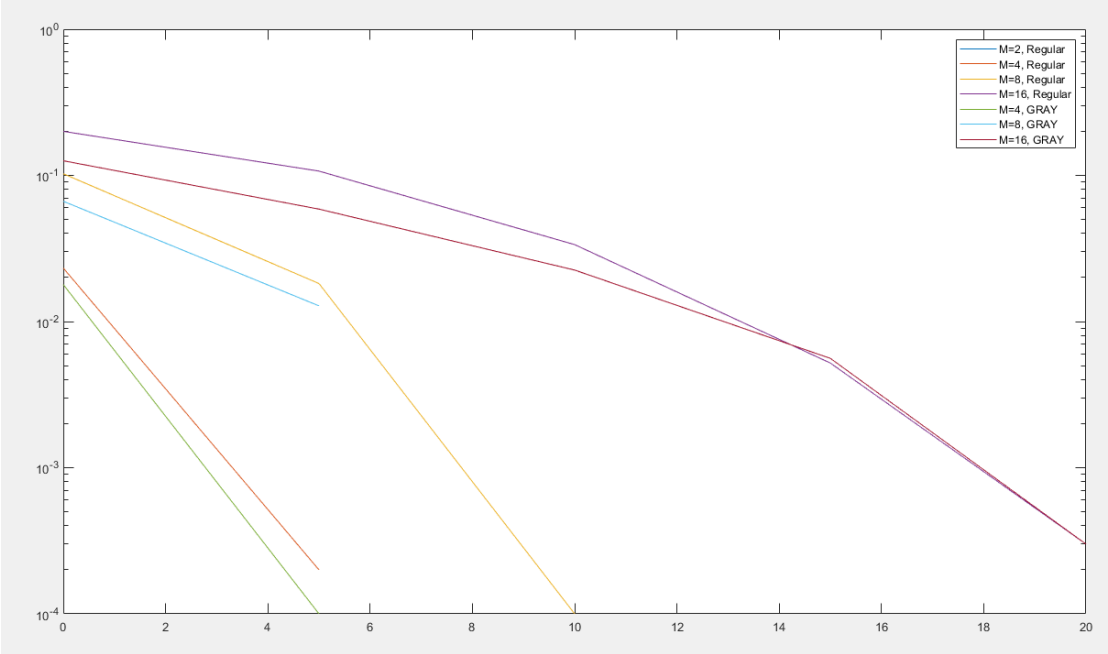
Έχοντας ως δεδομένα την ακολουθία bit εισόδου και την εκτιμώμενη δυαδική ακολουθία , παίρνουμε το άθροισμα του πλήθους των διαφορετικών τιμών ανάμεσα τους και το διαιρούμε με το μήκος της αρχικής ακολουθίας bit.

- **Symbol Error Rate**

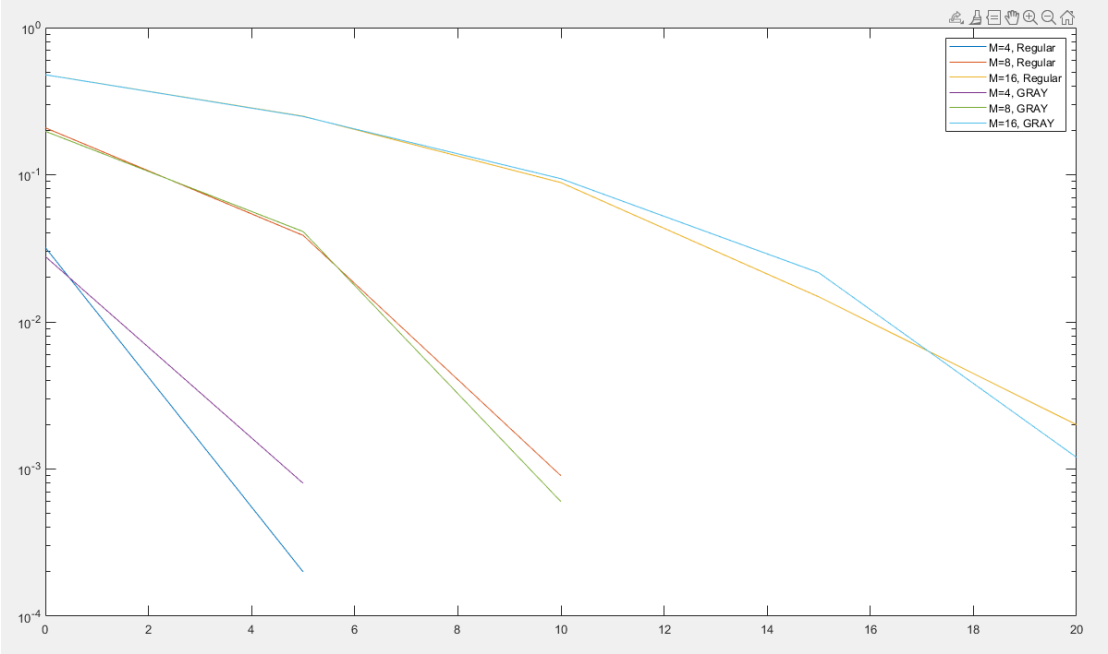
Ακολουθείται η ίδια λογική με τη μόνη διαφορά ότι η σύγκριση γίνεται ανάμεσα στα matched_symbol που έχει προκύπτει από το pam_coding και στο s_hat, την εκτίμηση για κάθε διάνυσμα συμβόλου που έχει προκύψει από το pam_decoding.

Στο **ber.m** **εδώ** και **ser.m** **εδώ** γίνονται τα plot για το **Bit Error Rate** και **Symbol Error Rate** αντίστοιχα.

Οι γραφικές παραστάσεις που προέκυψαν είναι:



Σχήμα: BER/SNR



Σχήμα: SER/SNR

ΚΩΔΙΚΕΣ ΜΕ ΑΝΑΛΥΣΗ

----------Α ΜΕΡΟΣ----------

myhuffmandict (επιστροφή)

```
function [ dict, avglen ] = myhuffmandict(s, p ) %symbols my alphabet
and p equals to probabilities of each symbol
if length(s) ~= length(p),
    error('The vector s and p must have the same dimensions')
end;

if (sum(p)-1)>10e-10,
    error('The sum of propabilities must equal to one')
end;

p = p(:)';
n=length(p);
q=p; %copy the propabilities to vector q
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;

%initialize the matrix c, which keeps the prefix code
for i=1:n-1,
    c(i,:)=blanks(n*n);
end;

c(n-1,n)='0'; %put the first zero starting from the bottom
c(n-1,2*n)='1'; %put the first one starting from the bottom

for i=2:n-1, %make the code to be prefix with zeros and ones
    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, %calculate the length of each codeword
    code(i,1:n)=c(1,n*(find(m(1,:)==i)-1)+1:find(m(1,:)==i)*n);%the
code for each symbol (in ascii format)
    len(i)=length(find(abs(code(i,:))~=32)); %the number of bits
needed for each code

end
```

```
   dict = cell(numel(p)+1,5); %create a dict type cell with
rows=number of possibilities and 5 columns
   dict{1,1} = {'Symbols'};
    dict{1,2}= {'Codeword'};
    dict{1,3} = {'length of codeword'};
    dict{1,4} = {'possibility of each symbols'};
    dict{1,5}=  {'len(i)*p(i)'};

    for i=2:numel(p)+1
        dict{i,1} = s(i-1);
        tempC = double(code(i-1,:))-48;
        dict{i,2} = tempC(tempC>=0);
        dict{i,3} = len(i-1);
        dict{i,4} = p(i-1);
        dict{i,5} = dict{i,3}*dict{i,4};

    end

avglen = sum(cellfun(@double,dict(2:numel(p)+1,5))); %calculate the
sum of p(i)*length(i)

end
```

myffmanenco (επιστροφή)

```
function bit_stream = myhuffmanenco(signal,dict)
 symbols = [dict{2:size(dict,1),1}]; %get the symbols from dict
 bit_stream = cell(numel(signal),1); %initialize the bit stream
 for i = 1:numel(bit_stream)
 index = strfind(symbols,signal(i));%search the signal(i) to find in
the symbols and keeps the index
 bit_stream{i} = dict{index+1,2}; %get the codeword of that symbol
 end
end
```

myffmandeco (επιστροφή)

```
function [ sig ] = myhuffmandeco( bit_stream,dict )
N=size(dict,1)-1
sig = [];
for i=1:size(bit_stream,1)

 for j=1:N
 if isequal(dict{j+1,2},bit_stream{i})
 sig = [sig dict{j+1,1}];
 break;
 end
 end
end
end
```

info_table (επιστροφή)

```
function [info] = info_table(dict,code)
p = [dict{2:size(dict,1),4}];
H=p*log2(1./p)';
avglen = sum(cellfun(@double,dict(2:numel(p)+1,5)));
efficiency = H/avglen;
for i=1:size(code,1)
       len(i)=length(code{i,1});
end;
huffmanbits=sum(len);
info=table(H,avglen,huffmanbits,efficiency,'VariableNames',{'Entropy'
'Average Length' 'Huffman bits' 'efficiency'});
end
```

ΕΡΩΤΗΜΑ 2 (επιστροφή)

```
%%%%%SOURCE A%%%%%%%%%%%%
alphabet =char([97:122]); %ascii table letters a-z (97-122)
prob
=[0.0816700000000000;0.0149200000000000;0.0278200000000000;0.04253000
00000000;0.127020000000000;0.0222800000000000;0.0201500000000000;0.06
09400000000000;0.0696600000000000;0.0015300000000000;0.0077200000000
0000;0.0402500000000000;0.0240600000000000;0.0674900000000000;0.07507
00000000000;0.0192900000000000;0.0009500000000000;0.059870000000000
0;0.0632700000000000;0.0905600000000000;0.0275800000000000;0.00978000
000000000;0.0236000000000000;0.0015000000000000;0.0197400000000000;0
.0007500000000000177]';

sourceA = alphabet(randsrc(1,10000,[1:26; prob])); %generate a rand
source of 10000 symbols of english letters
dictA = myhuffmandict(alphabet,prob);
codeA = myhuffmanenco(sourceA,dictA);
decodedA = myhuffmandeco(codeA,dictA);
if strcmp(sourceA,decodedA) %check if decoded data equals to the
original data
   disp('Huffman code works for sourceA')
end
info_table(dictA,codeA)
%%%%%%%%%%SOURCE B%%%%%%%%%%%%%%%%%%

ascii_number = double(lower(fscanf(fopen('kwords.txt','r'),'%s')))-
96; %keep the ascii number of each letter of file kwords. Because the
numbers exceeds 96(means not letters a to z) we substract 96
ascii_number= ascii_number(ascii_number>0 &
ascii_number<=26)+96;%ascii format in order all symbolas must me
letters(97-122)
sorted_ascii=char(sort(unique(ascii_number)));%letters in order
sourceB=char(ascii_number);%generate the correct source
dictB=myhuffmandict(sorted_ascii,prob); %follow the same procedure as
before
codeB=myhuffmanenco(sourceB,dictB);
decodedB=myhuffmandeco(codeB,dictB);
if strcmp(sourceB,decodedB)
    disp('Huffman code works for source B')
end;
info_table(dictB,codeB)
```

ΕΡΩΤΗΜΑ 3 (επιστροφή)

```
ascii_number = double(lower(fscanf(fopen('kwords.txt','r'),'%s')))-
96; %keep the ascii number of each letter of file kwords. Because the
numbers exceeds 96(means not letters a to z) we substract 96
ascii_number= ascii_number(ascii_number>0 &
ascii_number<=26)+96;%ascii format in order all symbolas must me
letters(97-122)
sorted_ascii=char(sort(unique(ascii_number)));%letters in order
A=(sort(unique(ascii_number)));
[A,j] = unique(A(:)) ;
[Idx ,Idx] = ismember(ascii_number,A);%returns the index in which
element of ascii_number is found on matrix A
 count = histc(Idx(:),1:length(A)) ;%calculates the occurences of
each symbol
%%%find the possibilites of each symbol in the 'kwords' text
count=count';
proB=count/sum(count);
sourceB=char(ascii_number);%generate the correct source
dictB=myhuffmandict(sorted_ascii,proB); %follow the same procedure as
before
codeB=myhuffmanenco(sourceB,dictB);
decodedB=myhuffmandeco(codeB,dictB);
if strcmp(sourceB,decodedB)
    disp('Huffman code works for source B')
end;
info_table(dictB,codeB)
```

ΕΡΩΤΗΜΑ 4 (επιστροφή)

```
alphabet =char([97:122]); %ascii table letters a-z (97-122)
prob
=[0.0816700000000000;0.0149200000000000;0.0278200000000000;0.04253000
00000000;0.127020000000000;0.0222800000000000;0.0201500000000000;0.06
09400000000000;0.0696600000000000;0.0015300000000000;0.0077200000000
0000;0.0402500000000000;0.0240600000000000;0.0674900000000000;0.07507
00000000000;0.0192900000000000;0.0009500000000000;0.059870000000000
0;0.0632700000000000;0.0905600000000000;0.0275800000000000;0.00978000
000000000;0.0236000000000000;0.0015000000000000;0.0197400000000000;0
.000750000000000177]';
sourceA = alphabet(randsrc(1,10000,[1:26; prob]));

[X,Y] = meshgrid(1:26, 1:26); %create two matrixes with the same
elements but different structure
newp = prod(prob([X(:) Y(:)]'))'; %new propabilities of each double
news = [1:676]'; %the total doubles are 676 = 26 * 26
newsig = reshape([double(sourceA)-96],2,5000)'; %produce our new
sourceA,which includes now doubles
newsig(:,1) = newsig(:,1) - 1;
newsig = newsig(:,1)*26+newsig(:,2);%multiply by 26, our alphabet is
from 1-676 not only 26
%%%%%%Follow the same procedure as before for Huffman code
newdict = myhuffmandict(news,newp);
codeA= myhuffmanenco(newsig,newdict);
decodedA = myhuffmandeco(codeA,newdict); %Now decodedA has numbers,
while sourceA is with double chars
%%%%%%%%%%%%%
```

```
newresp = ones(2,5000); %response matrix
newresp(2,:) = mod(decodedA,26);
temp = (newresp(2,:)==0);
newresp(newresp==0) = 26;
newresp(1,:) = floor(decodedA/26)+1;
newresp(1,temp) = newresp(1,temp)-1;
%%%%Check if Huffman works
newresp = char(newresp+96); %from double to char a to z
newresp = newresp(:)';
if strcmp(newresp,sourceA)
disp 'Huffman code works for doubled SourceA'
end
info_table(newdict,codeA)
```

ΕΡΩΤΗΜΑ 5 (επιστροφή)

```
%%%Kvdikopoisi tis pigis B me dictA
%%%%%get source B
ascii_number = double(lower(fscanf(fopen('kwords.txt','r'),'%s')))-
96; %keep the ascii number of each letter of file kwords. Because the
numbers exceeds 96(means not letters a to z) we substract 96
ascii_number= ascii_number(ascii_number>0 &
ascii_number<=26)+96;%ascii format in order all symbolas must me
letters(97-122)
sorted_ascii=char(sort(unique(ascii_number)));%letters in order
A=(sort(unique(ascii_number)));
[A,j] = unique(A(:)) ;
[Idx ,Idx] = ismember(ascii_number,A);%returns the index in which
element of ascii_number is found on matrix A
 count = histc(Idx(:),1:length(A)) ;%calculates the occurences of
each symbol
%%%find the possibilites of each symbol in the 'kwords' text
count=count';
proB=count/sum(count);
sourceB=char(ascii_number);

%%%%%%%%%%
%alphabet =char([97:122]); %ascii table letters a-z (97-122)
pro = [11.602 4.702 3.511 2.670 2.007 3.779 1.950 7.232 6.286 0.597
0.590 2.705 4.383 2.365 6.264 2.545 0.173 1.653 7.755 16.671 1.487
0.649 6.753 0.017 1.620 0.034]/100;

[X,Y] = meshgrid(1:26, 1:26); %create two matrixes with the same
elements but different structure
newpB = prod(pro([X(:) Y(:)]'))'; %new propabilities of each double
for dict from link
newpA = prod(proB([X(:) Y(:)]'))'; %new propabilities of each double
for dict from kword
news = [1:676]'; %the total doubles are 676 = 26 * 26
newsigB = reshape([double(sourceB)-96],2,numel(sourceB)/2)';
newsigB(:,1) = newsigB(:,1) - 1;
newsigB = newsigB(:,1)*26+newsigB(:,2);%multiply by 26, our alphabet
is from 1-676 not only 26
%%%%%%Follow the same procedure as before for Huffman code
newdictB = myhuffmandict(news,newpB);
newbB = myhuffmanenco(newsigB,newdictB);
```

```
numRespB = myhuffmandeco(newbB,newdictB); %Now decodedA has numbers,
while sourceA is with double chars
newresp = ones(2,numel(sourceB)/2);%response matrix
newresp(2,:) = mod(numRespB,26);
temp = (newresp(2,:)==0);
newresp(newresp==0) = 26;
newresp(1,:) = floor(numRespB/26)+1;
newresp(1,temp) = newresp(1,temp)-1;
newresp = char(newresp+96); %from double to char a to z
newresp = newresp(:)';
%%%%Check if Huffman works
if strcmp(newresp,sourceB)
 disp 'huffman code works with possibilites from link wiki'
end
a = info_table(newdictB,newbB)

%%%%%%%% copy paste from above but different possibilities
newdictA = myhuffmandict(news,newpA);
newbB = myhuffmanenco(newsigB,newdictA);
numRespB = myhuffmandeco(newbB,newdictA);
newresp = ones(2,numel(sourceB)/2);
newresp(2,:) = mod(numRespB,26);
temp = (newresp(2,:)==0);
newresp(newresp==0) = 26;
newresp(1,:) = floor(numRespB/26)+1;
newresp(1,temp) = newresp(1,temp)-1;
newresp = char(newresp+96);
newresp = newresp(:)';
if strcmp(newresp,sourceB)
 disp 'huffman code works for text'
end
info_table(newdictA,newbB)
```

---------- **Γ ΜΕΡΟΣ** ----------

pam_coding (επιστροφή)

```
function [r,matched_symbols,signal] =
pam_coding(bit_stream,M,gray,SNR)
if mod(M,2) ~= 0
    error('M must me power of 2')
end
%%%data given from the exercise
Eb=1/log2(M); %bit energy, since Es=1
Tsymbol=4; %Tsymbol=4 μsec
Tc=0.4; %period of carrier in μsec
fc=1/Tc; %carrier frequency in μsec
Tsample=Tc/4; %sampling period in μsec
No=Eb/10^(SNR/10); %since SNR=10log(Eb/No)
g_t=sqrt(2/Tsymbol); %orthogonal pulse gT(t)
Eg=(g_t^2)*Tsymbol;
t=0:Tsample:Tsymbol; %x axis is time
A=sqrt(3/(Eg*(M^2-1))); %energy of symbol formula from book

%%%%Create mapper with or no Gray
m=(1:M); %symbol vector
```

```
sm=(2*m-1-M)*A; %symbols formula from exercise

if gray == 0
    bin=de2bi(0:M-1,'left-msb');
    mapper=[sm',bin];
else
    gray=bitxor(0:M-1, floor((0:M-1)/2))';
    bin_gray=de2bi(gray,'left-msb');
    mapper=[sm',bin_gray];
end

%Matrix that contains rows that have log2(M) elements. The
%%number of them must be length(bit_stream)/log2(M) in order
%%%to be all represented
col=log2(M);
row=ceil(length(bit_stream)/log2(M)); %if the division is not
interger, round to the nearest integer
mat=zeros(row,col);

for i=1:row
    for j=1:col
        if (i-1)*col+j>length(bit_stream) %if the index goes beyond
the length of x, then stop
            break
        end
        mat(i,j)=bit_stream(1,(i-1)*log2(M)+j);

    end
end

%%%%%%Create a vector that it is initialized with zeros. It is used
to find
%%%%%%if each column (2:end) of mapper equals to the bit_stream.If
yes then
%%%%%%vector becomes 1

vector=logical(zeros(M,1));
matched_symbols=zeros(row,1); %symbols that are matched with the
initial bit_stream
for i=1:row
    for j=1:M
        vector(j)=isequal(mapper(j,2:end),mat(i,:)); %this is the
function that makes the logical 1 in vector
    end
    matched_symbols(i)=mapper(vector,1);
end

%%%Implement the signals
signal=matched_symbols*g_t.*cos(2*pi*fc*t);

%Create White Noise
%noise=(No/2)*random(size(signal)); %No/2 is the variance
noise=random('Normal',0,sqrt(No/2),size(signal));
%Add noise to the signal
r=signal+noise;
% figure;
 %hold on
%for i=1:size(r,1)
 %    plot(t,r(i,:));
%end
```

```
%hold off
end
```

pam_decoding (επιστροφή)

```
function [bin_est,s_hat] = pam_decoding(M,gray,r)
if mod(M,2) ~= 0
    error('M must me power of 2')
end
%%%data given from the exercise
Eb=1/log2(M); %bit energy, since Es=1
Tsymbol=4; %Tsymbol=4 μsec
Tc=0.4; %period of carrier in μsec
fc=1/Tc; %carrier frequency in μsec
Tsample=Tc/4; %sampling period in μsec

g_t=sqrt(2/Tsymbol); %orthogonal pulse gT(t)
Eg=(g_t^2)*Tsymbol;
t=0:Tsample:Tsymbol; %x axis is time
A=sqrt(3/(Eg*(M^2-1))); %energy of symbol formula from book
%%%%Create mapper with or no Gray
 %symbols formula from exercise
m=(1:M).'; %symbol vector
sm=(2*m-1-M)*A; %symbols formula from exercise
if gray == 0
    bin=de2bi(0:M-1,'left-msb');
    mapper=[sm,bin];
else
    gray=bitxor(0:M-1, floor((0:M-1)/2))';
    bin_gray=de2bi(gray,'left-msb');
    mapper=[sm,bin_gray];
end
%%Create the multiply r*g_t*cos(2*pi*fc*t)
for i=1:size(r,1)
    r_mul(i,:)=r(i,:).*(g_t.*cos(2*pi*fc*t));
end
%Adder
r_add=sum(r_mul,2)/10; %sum the elements of r_mult by row(dimension
2) and divide by 10 for not having values grater than 1

%Create the detector, which decides which symbol is closed to the
existing
%symbol %the distance is determined as min|sm-r_add(i)|
s_hat=zeros(size(r_add,1),1);
for i=1:size(r_add,1)
    [~,index]=min(abs(sm-r_add(i)));
    s_hat(i)=sm(index);
end
%%Create the Demapper
vector=logical(zeros(M,1)); %initialise logical vertical vector (it
will have this type of values:  1 (if each row of the mapper matches
the input) or 0 (if not))
matched_symbols=zeros(size(s_hat,1),log2(M)); %vector of the symbols
matched to the input binary strings

for i=1:size(s_hat,1)
    for j=1:M
        vector(j)=isequal(mapper(j,1),s_hat(i));  %this is the
function that makes the logical 1 in vector
```

```
    end
    matched_symbols(i,:)=mapper(vector,2:end); %match the symbol
s_hat(i) with the binary string
end
%Having the matched_symbols transorme it to row vector, it has to be
the
%initial bit_stream, but its not. The result is the bin_est, binary
estimation
bin_est=[];
for i=1:size(matched_symbols,1)
bin_est=[bin_est matched_symbols(i,:)];
end

end
```

calculate (επιστροφή)

```
function [bin_est,ber,ser] = calculate(bit_stream,M,gray,SNR)
[r,s_hat,~] = pam_coding(bit_stream,M,gray,SNR);
[bin_est,sw] = pam_decoding(M,gray,r);
bin_est=bin_est(1:length(bit_stream));
ber=sum(bit_stream~=bin_est)/length(bit_stream)
ser=sum(s_hat~=sw)/length(s_hat)

end
```

plot_ber (επιστροφή)

```
SNR=0:5:40;%9 different values of SNR
SER=zeros(1,9);
BER=zeros(1,9);

hold on
for i=1:1:9

    [~,BER(i),SER(i)]=calculate(bit_stream,2,0,SNR(i));

end

close all;

ber_fig=figure('Name','BER');
semilogy(SNR,BER);

hold on %hold ber figure


for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,4,0,SNR(i));
end
figure(ber_fig);
semilogy(SNR,BER);

hold on
```

```
for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,8,0,SNR(i));
end

figure(ber_fig);
semilogy(SNR,BER);
hold on

for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,2,0,SNR(i));
end
figure(ber_fig);
semilogy(SNR,BER);
hold on
for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,4,1,SNR(i));
end
figure(ber_fig);
semilogy(SNR,BER);
hold on
for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,8,1,SNR(i));
end
figure(ber_fig);
semilogy(SNR,BER);
hold on
for i=1:1:9
[~,BER(i),~]=calculate(bit_stream,16,1,SNR(i));
end
figure(ber_fig);
semilogy(SNR,BER);

legend('M=0, Regular','M=4, Regular','M=8, Regular','M=2, Regular',
'M=4, GRAY','M=8, GRAY','M=16, GRAY');
hold off %close ber figure
```

plot_ser (επιστροφή)

```
SNR=0:5:40;
SER=zeros(1,9);
BER=zeros(1,9);
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,4,0,SNR(i));
end
ser_fig=figure('Name','SER');
figure(ser_fig);
semilogy(SNR,SER);
hold on
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,8,0,SNR(i));
end
figure(ser_fig);
semilogy(SNR,SER);
hold on
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,16,0,SNR(i));
```

```
end
figure(ser_fig);
semilogy(SNR,SER);
hold on
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,4,1,SNR(i));
end
figure(ser_fig);
semilogy(SNR,SER);
hold on
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,8,1,SNR(i));
end
figure(ser_fig);
semilogy(SNR,SER);
hold on
for i=1:1:9
[~,~,SER(i)]=calculate(bit_stream,16,1,SNR(i));
end
figure(ser_fig);
semilogy(SNR,SER);
legend('M=4, Regular','M=8, Regular','M=16, Regular', 'M=4,
GRAY','M=8, GRAY','M=16, GRAY');
hold off
```