ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Εργαστηριακές Ασκήσεις 2020-2021 : 1ο σετ

Ελευθεριάδης Πέτρος

Παλαιός ΑΜ: 6044

Νέος ΑΜ: 1041741

ΕΡΩΤΗΜΑ 1

Κώδικας Huffman

Η βασική ιδέα του κώδικα huffman είναι πως σε μια πηγή, τα σύμβολα που εμφανίζονται πιο συχνά θα πρέπει να κωδικοποιηθούν με μικρότερου μήκους ψηφιά, ενώ τα σύμβολα που εμφανίζονται σπάνια θα κωδικοποιηθούν με περισσότερα ψηφιά.

Ο αλγόριθμος του λειτουργεί ως εξής

1. Ταξινομεί τις πιθανότητες των συμβόλων σε φθίνουσα σειρά
2. Συγχωνεύει τα 2 σύμβολα με τις μικρότερες πιθανότητες σε ένα σύμβολο με πιθανότητα το άθροισμα των δυο που συγχωνεύτηκαν
3. Αντιστοιχούμε τους αριθμούς 0 & 1 στις εξόδους των δυο συμβόλων
4. Πηγαίνει στο βήμα 1. μέχρι να μείνει ένα τελικό σύμβολο και η πιθανότητα να είναι == 1

Έπειτα για να πάρουμε το κώδικα κάθε σύμβολα, ξεκινάμε από τη ρίζα του δέντρου και πηγαίνουμε ως το κάθε φύλλο. Κάθε 0 ή 1 που συναντάμε προθέτεται στη λέξη.

Το αποτέλεσμα είναι μια προθεματική κωδικοποίηση η οποία προσφέρει μονοσήμαντη αποκωδικοποίηση

Ζητούμενο 1

Υλοποίηση huffmandict

Αρχικά η συνάρτηση μου παίρνει ως όρισμα ένα cell array από χαρακτήρες/αριθμούς ή ένα array από αριθμούς και ένα array από πιθανότητες.

Ταξινομώ και τα δυο με βάση τις πιθανότητες και ενώνω τα δυο τελευταία σύμβολα και μπαίνουν στη θέση του προτελευταίου (δλδ αν είχα 5 σύμβολα, τώρα έχω 4). Το ίδιο και για τις πιθανότητες, έχω πλέον 4 πιθανότητες και η 4η είναι η πρόσθεση του 4ου και του 5ου.

Φτιάχνω ένα matrix A το οποίο θα περιέχει τις τιμές 0&1. Έχει τόσες γραμμές όσες τα σύμβολα και τόσες στήλες όσες οι ενώσεις που θα γίνουν. Για το παράδειγμα που έδωσα πριν το matrix A τώρα

θα είχε τις τιμές [5;5;5;0;1]. Οι τιμές 5 είναι ένα άσχετο νούμερο απλά για να φαίνεται η διαφορά από τα 0.

Επίσης φτιάχνω ένα cell aray C το οποίο σε κάθε γραμμή περιέχει τα σύμβολα που ενωθήκαν μαζί ώστε αν σε ένα από αυτά ανατεθεί πχ η τιμή 0, τότε θα αναθέσω και σε όλα τα άλλα της ίδιας γραμμής τη τιμή 0. Για το ίδιο παράδειγμα, μετά από τη πρώτη πρόσθεση η πρώτη γραμμή του C θα είναι {[4],[5]}. Ύστερα αν ενωθεί το σύμβολο 1&2 θα γίνει έτσι: 

. Προφανώς στο τέλος το C θα έχει μόνο μια γραμμή από όλα τα σύμβολα αφού θα έχουν συγχωνευτεί όλα.

Εφόσον έχει τελειώσει ο αλγόριθμος huffman και έχει φτιαχτεί πλήρως το matrix A, παίρνω κάθε γραμμή του A, τη βάζω σε ένα array, διαγράφω τις τιμές “5”, τη κάνω fliplr και τη τοποθετώ στη κατάλληλη θέση του dict.

Έτσι φτιάχνεται το dict που στη πρώτη στήλη έχει το σύμβολο και στη δεύτερη το κώδικα huffman.

Υλοποίηση huffmanenco

Η συνάρτηση παίρνει ως όρισμα ένα cell array ή ένα array για το σήμα και το dict που φτιάξαμε πάνω.

Λειτουργεί ως εξής.

Παίρνει το πρώτο σύμβολο του σήματος, ψάχνει να βρει τη θέση του στο dict και παίρνει το κώδικα του. Έχοντας ορίσει ένα array ‘code’ από πριν, κάνω horzcat το code και το κώδικα κάθε συμβόλου του σήματος.

Στο τέλος, η ‘code’ έχει όλο το κώδικα huffman του σήματος.

Υλοποίηση huffmandeco

Παίρνει ως όρισμα το code και το dict που φτιάξαμε με τις δυο προηγούμενες συναρτήσεις

Λειτουργεί ως εξής.

Παίρνει κάθε σύμβολο του code και το κάνει horzcat() σε ένα προσωρινό array. Κάθε φορά που το array αυτό παίρνει ένα καινούριο σύμβολο ελέγχουμε αν υπάρχει στο dict η ακολουθία που έχει φτιαχτεί ως τώρα. Όταν φτιαχτεί μια ακολουθία που υπάρχει στο dict, τότε παίρνουμε το αντίστοιχο σύμβολο και το τοποθετούμε στη πρώτη θέση ενός cell array κ.ο.κ.

Στο τέλος το αποκωδικοποιημενο σήμα θα είναι στη πρώτη γραμμή του cell array.

Ζητούμενο 2

Πηγή Α

Αρχικά έφτιαξα ένα dict με τα σύμβολα και τις πιθανότητες από το wiki(ελάχιστα πειραγμένες ώστε να αθροίζουν στο 1).

Δημιούργησα ένα τυχαίο σήμα από 10000 σύμβολα και το έδωσα ως όρισμα στο myhuffmanenco μαζί με το dict

Το αποτέλεσμα είναι ένας κώδικας από 41180 ψηφιά

Η αποκωδικοποίηση του με το myhuffmandeco επέστρεψε το αρχικό σήμα των 10000 χαρακτήρων.

Πηγή Β

Για τη πηγή Β έβαλα όλα τα σύμβολα που από a έως z σε ένα cell array. Το μέγεθος ήταν 29092 χαρακτήρες.

Το έδωσα ως input στη myhuffmanenco και το αποτέλεσμα ήταν ένας κώδικας μήκους 136764 ψηφίων.

Η αποκωδικοποίηση επέστρεψε σωστά το αρχικό σήμα

Σχόλια

Παρατηρούμε ότι παρόλο που η πηγή Β είναι 2.9 φορές μεγαλύτερη από την Α, ο κώδικας που παράγεται είναι 3.3 φορές μεγαλύτερος από του Α. Αυτό οφείλεται στο γεγονός πως οι πιθανότητες του Α εκφράζουν καλυτέρα το σήμα Α. Παράλληλα οι πιθανότητες του Β (που είναι οι ιδίες με του Α) δεν εκφράζουν καλά το σήμα Β επειδή αυτό περιέχει μόνο λέξεις που αρχίζουν από 'k'. Γεγονός που κάνει το 'k' να έχει πολύ περισσότερες εμφανίσεις από ότι θα είχε σε μια τυχαία πηγή κειμένου.

Ζητούμενο 3

Μέτρησα τις εμφανίσεις κάθε συμβόλου στο cell array που περιέχει τα σύμβολα της πηγής Β και τα διαίρεσα δια το πλήθος των συμβολών για να βρω τις καινούριες πιθανότητες

Δημιούργησα ένα νέο dict, το dict_B το οποίο έχει τις νέες κωδικοποιήσεις.

Με αυτό έτρεξα το huffmanenco και έπειτα το huffmandeco για επιβεβαίωση

Σχόλια

Αυτό που παρατηρούμε εδώ είναι πως ο νέος κώδικας είναι μεγέθους 118865 ψηφίων, αρκετά μικρότερο από το ερώτημα 2. Αυτό συμβαίνει επειδή οι νέες πιθανότητες εκφράζουν πολύ καλύτερα τη πηγή, μιας και είναι βγαλμένες από την ίδια.

Ζητούμενο 4

Δημιούργησα τη 2ης τάξης επέκταση της πηγής Α, ενώνοντας όλα τα σύμβολα με όλα τα σύμβολα και πολλαπλασιάζοντας τις πιθανότητες τους. Τα αποτελέσματα τους τα έβαλα σε δυο arrays μεγέθους (26*26) ή 676.

Με αυτά τα δυο arrays έκανα από την αρχή το myhuffmandict και με ένα τυχαίο σήμα 5000 συμβόλων έτρεξα το myhuffmanenco και myhuffmandeco.

Τα αποτελέσματα ήταν τα εξής. Ο κώδικας που βγήκε έχει μέγεθος 41087, δηλαδή ελάχιστα μικρότερος από τη πηγή Α. Όσο για το μέσο μήκος, η πηγή Α είχε **μέσο μήκος**: **4.1170** ενώ η 2ης

τάξης επέκταση έχει **μέσο μήκος**: 8.2017/2= **4.10085** το οποίο είναι και αυτό ελάχιστα μικρότερο από της Α.

Για την **εντροπία** της Α γνωρίζουμε πως είναι **4.0846.**

Οπότε παρατηρούμε ότι η επέκταση της Α είναι πιο αποδοτική από την Α επειδή πλησιάζει πιο πολύ την εντροπία.

Ζητούμενο 5

Η 2ης τάξης επέκταση του Β με τις πιθανότητες του Α έχει το ίδιο **μέσο μήκος** με το Α δηλαδή **4.10085**. Ο κώδικας όμως από το myhuffmanenco έχει μήκος **136758**.

Όσο για την επέκταση με τη χρήση των πιθανοτήτων από το αρχείο kwords, έχουμε τα εξής. Δημιούργησα νέο dict το οποίο τώρα έχει διαφορετικές κωδικοποιήσεις. Στο myhuffmanenco τώρα έχουμε μήκος **136619** το οποίο είναι λίγο μικρότερο από πριν.

Όσο για το **μέσο μήκος**, αυτό είναι **4.0609.** (8.1217/2)

Η **εντροπία** της πηγής Β είναι **4.0470**.

Παρατηρούμε ότι η 2η περίπτωση είναι πολύ πιο κοντά στην εντροπία άρα και είναι πιο αποδοτική κωδικοποίηση

Γενικά όπως παρατηρούμε εδώ αλλά και όπως γνωρίζουμε από τη θεωρία, με μεγαλύτερης τάξης επεκτάσεις μπορούμε να πλησιάσουμε όσο θέλουμε την εντροπία. Το μειονέκτημα βέβαια σε αυτό είναι οι περισσότεροι υπολογισμοί που θα πρέπει να γίνουν και η ανάγκη για μεγαλύτερο χώρο αποθήκευσης.

ΕΡΩΤΗΜΑ 2

Ομοιόμορφο PCM

Αρχικά κατασκεύασα το σήμα με το τρόπο που ζητείται στην εκφώνηση για να το χρησιμοποιήσω ως είσοδο

Στη συνάρτηση my_quantizer.m πρώτα υπολογίζουμε τον αριθμό των περιοχών που είναι 2^N και την απόσταση δ των περιοχών.

Όσο για τη κατασκευή του σήματος xq δηλαδή των αντιπροσώπων τιμών, αυτή γίνεται ως εξής. Για κάθε τιμή στο σήμα x, εξετάζουμε σε ποια περιοχή ανήκει ή αν βρίσκεται εκτός ορίων και δίνουμε στο xq (με το τρόπο που αναφέρει η εκφώνηση) την αντίστοιχη τιμή .

Έπειτα κατασκευάζω το διάνυσμα που περιέχει τα κέντρα των περιοχών, όπου στη πρώτη θέση δίνουμε τη μεγαλύτερη τιμή και στη τελευταία δίνουμε τη μικρότερη ώστε να μπορούμε να χρησιμοποιούμε ως δείκτη το διάνυσμα xq.

Ζητούμενο 1a

Στη συνάρτηση my_quantizer.m υπολογίζω τα Px και παραμόρφωση για να βγάλω το sqnr. Για N=**4** αυτό είναι **16.3869** ενώ για N=**6** είναι **17.2915.**

Η μέση παραμόρφωση που υπολογίζω στο πρόγραμμα είναι **0.0444** και **0.0360** για N = **4** και **6** αντίστοιχα.

Όσο για το θεωρητικό υπολογισμό της παραμόρφωσης, ο οποίος έγινε στη συνάρτηση calc_distortion.m χρησιμοποίησα τον τύπο του βιβλίου με το άθροισμα ολοκληρωμάτων. Συμπεριέλαβα επίσης και το ολοκλήρωμα από 4 έως άπειρο. Το αποτέλεσμα είναι **0.0466** και **0.0381** για N = **4** & **6** αντίστοιχα.

Παρατηρούμε ότι οι θεωρητικές πλησιάζουν σε μεγάλο βαθμό τις τιμές του προγράμματος. Αυτό συμβαίνει επειδή το σήμα ακολουθεί την εκθετική κατανομή και το χρησιμοποίησα αυτό στο θεωρητικό υπολογισμό, οπότε είναι λογικό να είναι κοντά οι δυο απαντήσεις.

Ζητούμενο 1b

Υπολογίζοντας το πλήθος των τιμών του x που είτε είναι μικρότερες του 0 ή μεγαλύτερες του 4, δίνει αποτέλεσμα 167. Αυτό σημαίνει πως η πιθανότητα υπερφόρτωσης είναι 1,67%.

Μη Ομοιόμορφο PCM / Lloyd_Max

Αρχικά για να μετατρέψω το σήμα ήχου σε διάνυσμα τιμών χρησιμοποίησα την audioread() της Matlab.

Στο πρόγραμμα αρχικά κατασκεύασα το διάνυσμα centers όπως και στο ομοιόμορφο PCM. Υστέρα έφτιαξα μια while loop που σταματάει όταν η διαφορά διαδοχικών μέσων παραμορφώσεων είναι μικρότερη του ‘ε’.

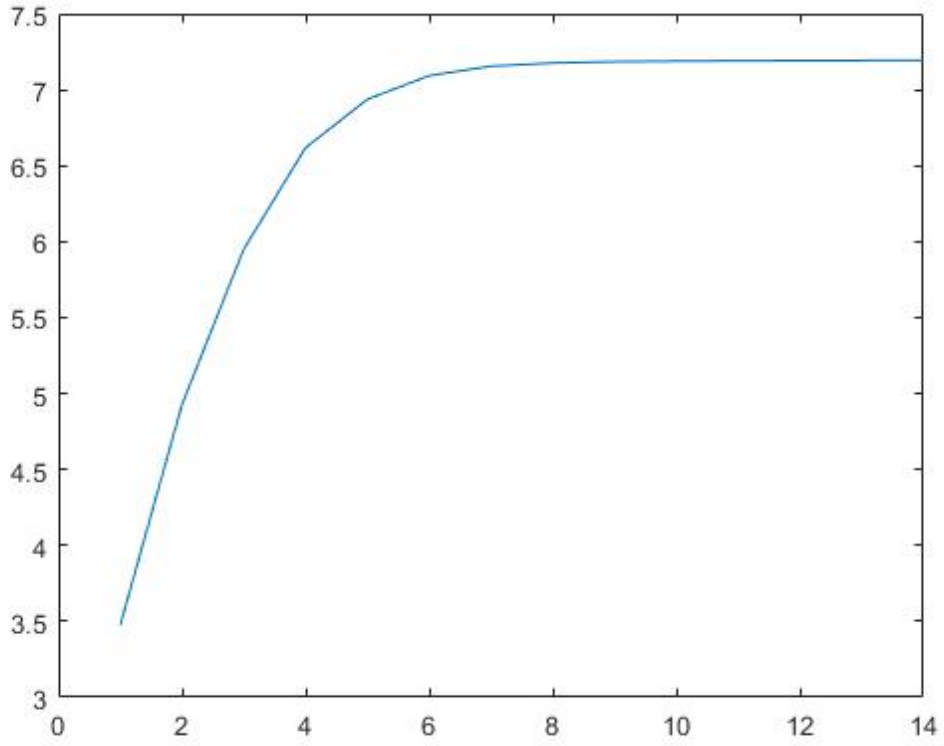
Υστέρα κατασκεύασα το διάνυσμα T δίνοντας του ως τιμές τα μέσα δυο διαδοχικών τιμών του centers εκτός των ακραίων τιμών που πήραν τις τιμές των min & max value.

Ακολουθεί η κατασκευή του xq που παίρνει την αντιπρόσωπο τιμή ανάλογα αναμεσά ποιων T βρίσκεται η τιμή x.

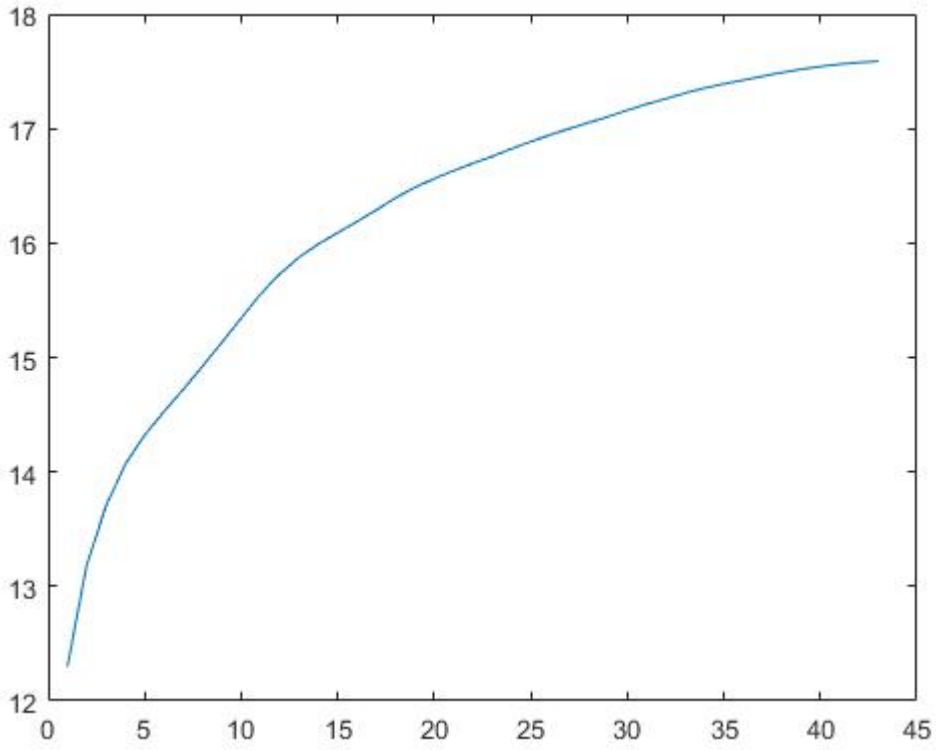
Τέλος ξανακατασκευάζουμε το centers σύμφωνα με τον αλγόριθμο. Οπότε παίρνουμε για κάθε περιοχή και αθροίζουμε τις τιμές που πέφτουν μέσα και μετράμε το πλήθος τους ώστε να βρούμε τον σταθμισμένο μέσο όρο. Στις περιπτώσεις των άκρων, μετράμε και τις τιμές που πέφτουν εκτός ορίων. Όταν σταματήσει ο αλγόριθμος, το τελευταίο k είναι το kmax.

Ζητούμενο 2a

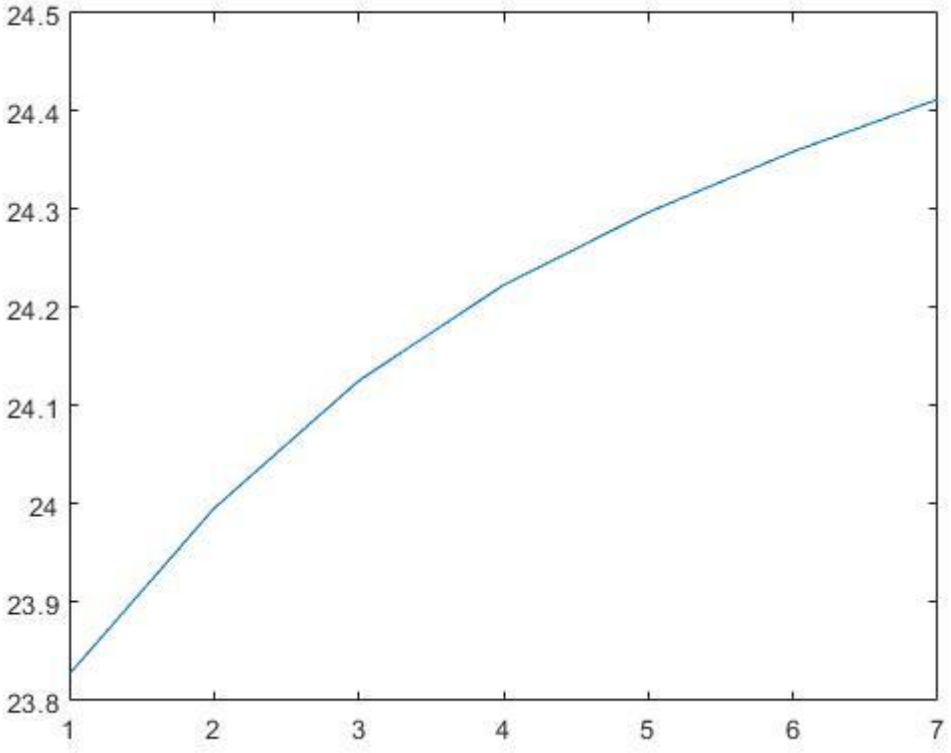
Μεταβολή SQNR για N=2



Μεταβολή SQNR για N=4



Μεταβολή SQNR για N=6



Το αξιοσημείωτο εδώ είναι πως για N=4 το αλγόριθμος κάνει 43 επαναλήψεις για να φτάσει στο kmax, ενώ για N=2 θέλει 14. Παρ' όλ' αυτά όμως για N=4 το sqnr καταφέρνει και φτάνει μεγαλύτερη τιμή από τη περίπτωση του N=2.

Ζητούμενο 2b

Στις συναρτήσεις Lloyd_Max.m και my_quantizer.m υπολογίζω τα distortions και Px ώστε να πάρω το sqnr το οποίο δίνεται από το τύπο 10log10 (Px/distortion). Τα αποτελέσματα φαίνονται στο πίνακα.

SQNR (db)

	Ομοιόμορφο PCM	Μη Ομοιομορφο PCM
N=2	-2.9	7.193
N=4	10.755	17.587
N=6	23.705	24.412

Παρατηρούμε ότι για μικρό N το ομοιόμορφο PCM αποδίδει πολύ άσχημα καθώς το sqnr βγαίνει αρνητικό, ενώ όσο το N αυξάνεται τότε το ομοιόμορφο πλησιάζει το μη ομοιόμορφο σε απόδοση. Όπως βλέπουμε για N=6 οι τιμές των sqnr είναι πολύ κοντά μεταξύ τους.

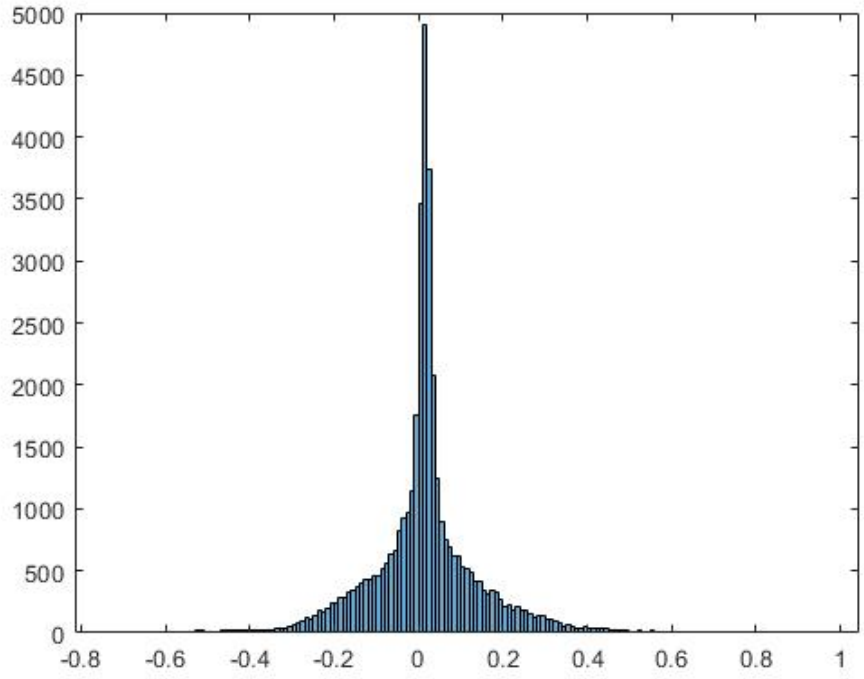
Για 'ε' πήρα τη τιμή 10^-6.

Ζητούμενο 2c

Για N=2

Θεωρητικές Πιθανότητες	Πειραματικές πιθανότητες
0.2723	0.1152
0.0723	0.6394
0.0627	0.1910
0.1903	0.0544

Για να πάρω τις θεωρητικές τιμές έκανα το εξής. Αρχικά από το ιστόγραμμα του y



παρατηρούμε ότι το σήμα μπορεί να προσεγγιστεί από τη κανονική κατανομή. Ο σκοπός μας είναι να πάρουμε όλες τις πιθανότητες μεταξύ δυο τιμών. Αυτό επιτυγχάνεται με την αθροιστική συνάρτηση κατανομής και πιο συγκεκριμένα πρέπει να αφαιρέσουμε την α.σ.κ της μικρότερης τιμής από την α.σ.κ της μεγαλύτερης. Αυτό έκανα στο calc_p.m για κάθε T και T_{+1}. Στο τέλος αυτής της συνάρτησης υπολόγισα και την εντροπία με το γνωστό τύπο.

Για τις πειραματικές τιμές των πιθανοτήτων, στο Lloyd_Max.m μέτρησα πόσες τιμές αντιπροσωπεύονται από μια συγκεκριμένη στάθμη. Κάνοντας αυτό για όλες τις στάθμες, βρήκα τις πιθανότητες.

Παρατηρούμε το εξής. Για τις 3 από τις 4 στάθμες η διαφορά μεταξύ των πιθανοτήτων είναι περίπου στο 0.15. Για τη 2η στάθμη όμως η διαφορά είναι 0,45.

Όσο για την **εντροπία** των σταθμών, αυτή είναι **1.033** από τις θεωρητικές τιμές και **1.0095** αν χρησιμοποιήσω τις πειραματικές.

Ζητούμενο 2d

Αποδοτικότητα σε MSE

	Ομοιομορφο PCM	Μη Ομοιομορφο PCM
N=2	0.03622	0.00354
N=4	0.00156	0.00156
N=6	7.9168e-05	7.9168e-05

Στο calc_efficiency.m αφαίρεσα τις τιμές που έβγαλε το σύστημα από τις αρχικές του σήματος και υστέρα τα τετραγώνισα όπως καθορίζει το Mean Squared Error (MSE). Και εδώ φτανουμε στο ιδιο

συμπέρασμα. Για μικρό Ν, το μη ομοιόμορφο είναι πολύ καλύτερο, ενώ για μεγαλύτερα Ν το ομοιόμορφό καλύπτει τη διαφορά.

ΚΩΔΙΚΕΣ ΕΡΩΤΗΜΑΤΟΣ 1

myhuffmandict.m

```
5.  function [mydict, integer_symbols]= myhuffmandict(symbols, prob)
6.    %get number of symbols
7.    [~, num_of_symbols] = size(symbols);
8.    [~, num_of_probs] = size(prob);
9.    if num_of_symbols ~= num_of_probs
10.     error("Number of symbols must equal to number of probabilities");
11.   end
12.   if sum(prob)>1.0001 || sum(prob)< 0.9999
13.     error("probabilities must sum to 1");
14.   end
15.   k=1;
16.   integer_symbols(1,num_of_symbols)=0;
17.   char_positions(1,num_of_symbols)=0;
18.   if iscell(symbols)
19.     for i=1:num_of_symbols
20.       if ischar(symbols{1,i})
21.         integer_symbols(1,i) = double(symbols{1,i});
22.         char_positions(1,k) = i;
23.         k=k+1;
24.       else
25.         integer_symbols(1,i)= symbols{1,i};
26.       end
27.     end
28.   else
29.     for i=1:num_of_symbols
30.       integer_symbols(1,i)= symbols(1,i);
31.     end
32.   end
33.   symbols = integer_symbols;
34.   %Initialize matrix A with an irrelevant number eg 5
35.   %The matrix will have the final huffman code for each symbol
36.   %Row 1 of A represents the first symbol
37.   %Second row represents second symbol etc...
38.   %Columns of A represent the time of each addition of the probabilities
39.
```

```
% H U F F M A N
    A(1:num_of_symbols,1:num_of_symbols) = 5;
    C{1,1}=0;
    for i = 1:num_of_symbols-1

      if prob >= 1
         break;
      end
      %sort probabilities and symbols
      [prob, sortIdx] = sort(prob, 'descend');
      symbols = symbols(sortIdx);
      [~, last_symbol] = size(symbols);
      %add the last two probabilities and assign to second last symbol
      disp(last_symbol-1);
      prob(last_symbol-1) = prob(last_symbol-1) + prob(last_symbol);
      %keep track of symbols position
      symid1 = symbols(last_symbol-1);
      idx1 = find(integer_symbols == symid1);
      symid2 = symbols(last_symbol);
      idx2 = find(integer_symbols == symid2);
      %0 goes to second last, 1 goes to last probability
      A(idx1,i) = 0;%0
      A(idx2,i) = 1;%1
      flag1=0;
      flag2=0;
     [rowsize, ~] = size(C);
     %Symbols that were added together are concatanated in arrays in the
     %cell. Each row of C cell is an array of symbols that have
     %been added together in the past. So whatever happens to one of
     %them, must happen to the whole array
     for k=1:rowsize
        if ~isempty(find(C{k,1} == idx1, 1))
           x = k;
           flag1=1;
        end
        if ~isempty(find(C{k,1} == idx2, 1))
           y = k;
           flag2=1;
        end
     end
      %If a symbol gets a 0 or 1, make sure that all others in
      %the same row as that symbol get the same value too
      if flag1
        temp = C{x,1};
```

```
84.          [~, column_size] = size(temp);
85.           for j = 1:column_size
86.              var=temp(1,j);
87.              A(var, i) = A(idx1,i);
88.           end
89.           C{x,1} = horzcat(temp, idx2);
90.       end
91.       if flag2
92.          temp = C{y,1};
93.          [~, column_size] = size(temp);
94.           for j = 1:column_size
95.              var=temp(1,j);
96.              A(var, i) = A(idx2,i);
97.           end
98.           C{y,1} = horzcat(temp, idx1);
99.       end
100.              %Starting condition. First addition
101.              if i==1
102.                 C{i,1} =[idx1 idx2];
103.              end
104.              %if symbols are not already in the cell. Create new row in the
105.              %cell to place them
106.              if i>1 && ~flag1 && ~flag2
107.                 C{rowsize+1,1} = [idx1 idx2];
108.              end
109.              %if both symbols are already in the cell, merge them
110.              if flag1 && flag2
111.                 temp = horzcat(C{x,1}, C{y,1});
112.                 C{x,1} = unique(temp);
113.                 C{y,1}=[];
114.              end
115.              %delete last symbol and its prob
116.              symbols(last_symbol) = [];
117.              prob(last_symbol) = [];
118.           end
119.           disp(A);
120.           disp(C);
121.           %Transform rows of A to 1d-arrays. Get rid of the 5s, and reverse the
    vectors
122.           %Create a cell array with symbols and vectors
123.
124.           %initialize cell array
125.           mydict{num_of_symbols,2}=0;
126.           [rows, ~]= size(A);
```

```
        for i = 1:rows
            temp = A(i,:);
            temp(temp==5)=[];
            temp = fliplr(temp);
            if ~isempty(find(i == char_positions(1,:), 1))
                mydict{i,1} = char(integer_symbols(1,i));
            else
                mydict{i,1} = integer_symbols(1,i);
            end
            mydict{i,2} = temp;
        end
    end
```

myhuffmanenco.m

```
    function code = myhuffmanenco(sig, dict)
        [~,sig_cols] = size(sig);
        [dict_rows,~] = size(dict);

        code = [];
        if iscell(sig)
            for i = 1:sig_cols
                for j = 1:dict_rows
                    if sig{1,i}==dict{j,1}
                        code = horzcat(code, dict{j,2});
                    end
                end
            end
        else
            for i = 1:sig_cols
                for j = 1:dict_rows
                    if sig(1,i)==dict{j,1}
                        code = horzcat(code, dict{j,2});
                    end
                end
            end
        end
    end
```

myhuffmandeco.m

```
function signal = myhuffmandeco(code, dict)
  [~, code_cols] = size(code);
  [dict_rows,~] = size(dict);
  temp=[];
  signal={};
  k=1;
  for i=1:code_cols
    temp = horzcat(temp, code(1,i));
    for j=1:dict_rows
      [~,temp_cols]=size(temp);
      [~,dict_cols]=size(dict{j,2});
      if temp_cols == dict_cols
        if temp==dict{j,2}
          signal{1,k} = dict{j,1};
          k=k+1;
          temp=[];
        end
      end
    end
  end

end
```

calc_avg_len.m

```
function avg = calc_avg_len(sym,prob, dict)
[charrows,~]= size(dict);
avg=0;
temp=[];
for i=1:charrows
    for j=1:charrows
        if isequal(sym(i), dict{j,1})%sym(i) becomes
  sym{i} in case of cells
            temp(1,j) = prob(1,j) * length(dict{j,2});

        end
    end
end
avg = sum(temp);
```

combine_symbols.m

```
combined_symbols{26,26}=0;
int_symbols(1:26,1:26)=0;
[prob_B, sortIdx] = sort(prob_B, 'descend');
```

```
200.    symbols = symbols(sortIdx);
201.    for i=1:26
202.      for j=1:26
203.        combined_symbols{i,j} = horzcat(symbols{i}, symbols{j});
204.        combined_prob(i,j)= prob_B(i)*prob_B(j);
205.
206.      end
207.    end
208.    int_symbols=(1:26*26);
209.    clear i;
210.    clear sortIdx;
```

get_prob.m

```
211.    function p = get_prob(signal, dict)
212.      total_symbols = length(signal);
213.      [char_rows, ~] = size(dict);
214.      p(1:26) = 0;
215.      for i=1:total_symbols
216.        for j=1:char_rows
217.          if signal{1,i} == dict{j,1}
218.            p(1,j) = p(1,j)+1;
219.
220.          end
221.        end
222.      end
223.      p = p/total_symbols;
224.    end
```

read_file.m

```
225.    fileID = fopen('kwords.txt','r');
226.    formatSpec = '%c';
227.    C={};
228.    int_symbols=[];
229.    B = fscanf(fileID,formatSpec);
230.    [~,cols]=size(B);
231.    k=1;
232.
233.    for i=1:cols
234.      if ~isspace(B(1,i)) && double(B(1,i))>96 && double(B(1,i))<123
235.        C{1,k} = B(1,i);
236.        int_symbols(1,k) = double(C{1,k});
```

```
            k=k+1;
        end

    end
```

ΚΩΔΙΚΕΣ ΕΡΩΤΗΜΑΤΟΣ 2

my_quantizer.m

```
function [xq, centers] = my_quantizer(x, N, min_value, max_value)
    %number of regions
    num_of_regions = 2^N;
    %distance between two centers
    range = (max_value - min_value) / (num_of_regions);
    %frequency for distortion overload
    frequency=0;
    %represtative values of x
    xq = zeros(length(x),1);
    for i=1:length(x)
        %if x is less than min value
        if x(i)<=min_value
            xq(i) = num_of_regions;
            frequency = frequency + 1;
        %if x is bigger than max value
        elseif x(i) >= max_value
            xq(i) = 1;
            frequency = frequency + 1;
        else
            for j=2:num_of_regions+1
                %find the region of x and give xq the appropriate number
                if x(i)>= (min_value + (j-2) * range) && x(i)<= (min_value + (j-1)*
range)
                    xq(i) = num_of_regions + 1 - (j-1);
                end
            end
        end
    end


    %last element has lowest value
    centers(num_of_regions) = min_value + range/2;
```

```
272.        %assign values
273.        for i=2:num_of_regions-1
274.            centers(num_of_regions-i+1) = min_value + range/2 + (i-1)*range;
275.        end
276.        %first element has highest value
277.        centers(1) = max_value - range/2;
278.        %Px
279.        px=0;
280.        %distortion
281.        dist=0;
282.
283.        for i=1:length(xq)
284.            %calculate distortion & px
285.            dist = dist + ((x(i)-centers(xq(i))) ^2);
286.            px = px + x(i)^2;
287.        end
288.        disp('SQNR')
289.        disp(10*(log10(px/dist)));
290.        disp('Distortion:');
291.        disp(dist);
292.        disp('Px:');
293.        disp(px);
294.        disp('Frequencies');
295.        frequency = frequency/length(x);
296.        disp(frequency);
297.
298.
299.    end
300.
301.
```

Lloyd_Max.m

```
302.    function [xq, centers, D, sqnr] = Lloyd_Max(x, N, min_value, max_value)
303.        %number of regions
304.        num_of_regions = 2^N;
305.        %distance of region like my_quantizer
306.        range = (max_value - min_value) / (num_of_regions);
307.        %last element has lowest value
308.        centers(num_of_regions) = min_value + range/2;
309.        %assign values
310.        for i=2:num_of_regions-1
```

```
311.            centers(num_of_regions-i+1) = min_value + range/2 + (i-1)*range;
312.        end
313.        %first element has highest value
314.        centers(1) = max_value - range/2;
315.        %k counts iterations of while loop
316.        k=1;
317.        %initialize T
318.        T = zeros(1, num_of_regions+1);
319.        %initialize xq
320.        xq = zeros(1, length(x));
321.        while true
322.            %first T element has the maxvalue
323.            T(1) = max_value;
324.            %assign according to LloydMax algorithm
325.            for i=2:num_of_regions
326.                T(i) = (centers(i)+centers(i-1))/2;
327.
328.            end
329.            %last T element has the minvalue
330.            T(num_of_regions+1) = min_value;
331.
332.            for i=1:length(x)
333.                %assign values to xq according to which region x falls in
334.                if x(i)>= max_value
335.                    xq(i) = 1;
336.                elseif x(i) <= min_value
337.                    xq(i) = num_of_regions;
338.                else
339.                    for j=1:num_of_regions
340.                        if x(i)<=T(j) && x(i)>T(j+1)
341.                            xq(i) = j;
342.                        end
343.                    end
344.                end
345.
346.            end
347.            %all xq values must be assigned
348.            if find(xq==0)
349.                error('0 found');
350.            end
351.            %initialize sum of x values
352.            sum = zeros(num_of_regions,1);
353.            %initialize number of x values
354.            counter = zeros(num_of_regions,1);
```

```
            for n=1:num_of_regions
                %add and count values that are in the same region
                for i=1:length(x)
                    if (x(i) <= T(n) && x(i) > T(n+1))
                        sum(n) = sum(n) + x(i);
                        counter(n) = counter(n) + 1;

                    elseif ((x(i) > T(n)) && (n == 1))
                        sum(n) = sum(n) + T(n);
                        counter(n) = counter(n) + 1;

                    elseif ((x(i) < T(n+1)) && (n == num_of_regions))
                        sum(n) = sum(n) + T(n+1);
                        counter(n) = counter(n) + 1;
                    end
                end
                %divide sum by the number of elements to get weighted average
                if (counter(n) > 0)
                    centers(n) = sum(n)/counter(n);
                end
            end
            %distortion
            dist=0;
            %Px
            px=0;
            %calculate distortion and Px
            for i=1:length(xq)
                dist = dist + (x(i)-centers(xq(i))) ^2;
                px = px + x(i)^2;
            end
            %get kth sqnr
            sqnr(k)= 10*log10(px/dist);
            %get kth average distortion
            D(k) = dist/length(xq);

            %break condition
            if k>1
                if(abs(D(k)-D(k-1)) < 10^(-6))
                    break;
                end
            end
            k=k+1;
        end
        %frequency of every region
```

```
399.        frequency = zeros(num_of_regions,1);
400.        %count every value in the same region
401.        for i=1:num_of_regions
402.          for k=1:length(xq)
403.            if (i == xq(k))
404.              frequency(i) = frequency(i) + 1;
405.            end
406.          end
407.        end
408.        %calculate probability
409.        p = frequency./length(xq);
410.        disp(p);
411.      end
```

calc_distortion.m

```
412.      clear distortion;
413.      %first calculate every x bigger than max value
414.      f = @(x)((x - centers(1)).^2) .*exp(-x);
415.      distortion = integral(f,4,Inf);
416.      %calculate the integral of every region and add them together to get
417.      %the value of distortion
418.      for i=1:num_of_regions
419.      f = @(x)((x - centers(num_of_regions+1-i)).^2) .*exp(-x);
420.      distortion = distortion + integral(f,(i*range-range),i*range);
421.      end
```

calc_px.m

```
422.      %calculate Px
423.      fun = @(x) (x.^2).*exp(-x);
424.      px = integral(fun,0,Inf);
```

calc_p.m

```
425.      function [p, entropy] = calc_p(y)
426.        [~,centers] = Lloyd_Max(y,2,-1,1);
427.        num_of_regions = 2^2;
428.        T = zeros((num_of_regions+1),1);
429.        T(1) = max(y);
430.        for i=2:(num_of_regions)
```

```
431.            T(i) = (centers(i)+centers(i-1))/2;
432.        end
433.        T(5) = min(y);
434.        p = zeros((num_of_regions),1);
435.        %calculate probability that x is inside a region
436.        for i=1:(num_of_regions)
437.            p(i) = normcdf(T(i)) - normcdf(T(i+1));
438.        end
439.        %entropy formula
440.        entropy = -sum(p.*log(p));
441.    end
```

calc_efficiency.m

```
442.    function [efficiency] = calc_efficiency(x, centers, xq)
443.        efficiency=0;
444.        %MSE
445.        for i=1:length(x)
446.            efficiency = efficiency + (x(i) - centers(xq(i)))^2;
447.        end
448.        efficiency = efficiency / length(x);
449.    end
```