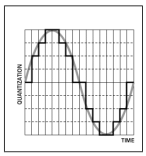
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ – ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΨΗΦΙΑΚΕΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ

Εργαστηριακή Άσκηση #1	2013-2014	Καφφέζας Γιώργος	**ΑΜ** 4465



Χαρακτηριστικά συστήματος

Η παρούσα άσκηση υλοποιήθηκε σε λειτουργικό σύστημα Windows 7 Professional SP1 (x64). Όσον αφορά το υλικό του συστήματος, ο επεξεργαστής του είναι Intel Core 2 Quad Q6600 στα 2.4GHz, με διαθέσιμη RAM DDR3 4GB. Τέλος, η έκδοση MATLAB που χρησιμοποιήθηκε είναι η R2012b (8.0.0.783), πάλι για 64-bit, και η αναφορά συντάχθηκε με LibreOffice Writer 4.1.3.2.

• • • • • •

ΕΡΩΤΗΜΑ #1

α) Σύμφωνα με τη θεωρία του βιβλίου[1] στη σελίδα 338, η παλμοκωδική διαμόρφωση (pulse-code modulation ή αλλιώς PCM) είναι το απλούστερο και παλαιότερο σχήμα κωδικοποίησης κυματομορφής. Ένας παλμοκωδικός διαμορφωτής αποτελείται από τρία μέρη: τον δειγματολήπτη, τον κβαντιστή και τον κωδικοποιητή. Ένα γενικό σχήμα ενός τέτοιου συστήματος είναι και το εξής:

Διάγραμμα βαθμίδων ενός συστήματος PCM
(σχήμα 6.15 του βιβλίου)

Στο παρόν ερώτημα μας ζητείται να υλοποιήσουμε το δεύτερο μέρος του συστήματος, τον κβαντιστή, καθώς τη δειγματοληψία την πραγματοποιούμε με την ενδογενή συνάρτηση της MATLAB, wavread (όπως υποδεικνύεται και στην εκφώνηση). Ειδικότερα, στο πρώτο υποερώτημα μας ζητείται να υλοποιήσουμε έναν ομοιόμορφο PCM κβαντιστή, ενώ στο δεύτερο έναν μη-ομοιόμορφο, με χρήση της μεθόδου Lloyd-Max.

Ο κώδικας, λοιπόν, που αφορά στον ομοιόμορφο PCM κβαντιστή βρίσκεται στο παράρτημα, με όνομα *my_quantizer.m*. Για την σχεδίασή του ακολουθήθηκαν οι υποδείξεις της άσκησης, και η λειτουργία του αναλύεται μέσω των σχολίων στο αρχείο του ίδιου του κώδικα.

β) Ξανά σύμφωνα με το βιβλίο, και ακριβέστερα στις σελίδες 340-341, η μη-ομοιόμορφη κβάντιση χρησιμοποιείται όταν το σήμα που πρέπει να κωδικοποιηθεί δεν ακολουθεί ομοιόμορφη κατανομή. Αυτό έχει σαν αποτέλεσμα να εμφανίζονται κάποια πλάτη με μεγαλύτερη πιθανότητα από άλλα, όπως συμβαίνει και στην περίπτωση της πηγής Α, της ομιλίας. Έτσι, απαιτείται ένας κβαντιστής που θα έχει περισσότερες περιοχές κβάντισης για τα πλάτη που εμφανίζονται περισσότερο, και λιγότερες για αυτά που εμφανίζονται λιγότερο. Το γενικό διάγραμμα ενός συστήματος μη-ομοιόμορφου PCM είναι το ακόλουθο:

Διάγραμμα βαθμίδων ενός συστήματος μη-ομοιόμορφου PCM
(σχήμα 6.16 του βιβλίου)

Στο παρόν υποερώτημα μας ζητείται να υλοποιήσουμε έναν μη-ομοιόμορφο κβαντιστή PCM χρησιμοποιώντας τον αλγόριθμο Lloyd-Max, όπως αναφέρθηκε και παραπάνω. Οι συνθήκες που θέτει ο αλγόριθμος Lloyd-Max αξιοποιούνται προς υλοποίηση του βέλτιστου βαθμωτού κβαντιστή (βλ. Σελίδα 333

1 Με τον όρο «βιβλίο» αναφέρεται ένα από τα διαθέσιμα συγγράμματα του μαθήματος, και συγκεκριμένα το βιβλίο των J. G. Proakis & M. Salehi, «Συστήματα Τηλεπικοινωνιών», 2002. Στη συνέχεια της αναφοράς, όπου αλλού γίνεται αναφορά σε «βιβλίο» εννοείται αυτό.

του βιβλίου). Ακολουθώντας τις οδηγίες εκτέλεσης του αλγορίθμου, όπως αυτές διευκρινίζονται, καταλήγουμε στην συγγραφή του κώδικα που βρίσκεται στο παράρτημα, με όνομα αρχείου *Lloyd_Max.m*. Η μόνη «παρέκκλιση» από τις οδηγίες ήταν η προσθήκη ενός επιπλέον ορίσματος στη συνάρτηση, ώστε να μπορεί να δοθεί ως είσοδος και η ελάχιστη αποδεκτή τιμή min_value, όπως γίνεται και στον ομοιόμορφο κβαντιστή.

ΕΡΩΤΗΜΑ #2

Για την υλοποίηση του παρόντος ερωτήματος συντάχθηκε το script που βρίσκεται στο παράρτημα της αναφοράς, με όνομα *script_1_2.m*. Η λειτουργία του είναι η ακόλουθη. Αρχικά, φορτώνεται το σήμα από την πηγή Α με τη wavread, ώστε να μπορεί να υποστεί επεξεργασία. Αμέσως μετά γίνεται η κβάντιση και με τα δύο σχήματα κβαντιστών (ομοιόμορφου και μη) για 2, 4 και 8 bits, όπως ακριβώς ζητείται.

Στη συνέχεια, υπολογίζονται οι τιμές του SQNR (signal-to-quantization-noise ratio), σύμφωνα με τον τύπο (6.5.3) της σελίδας 329 του βιβλίου, δηλαδή τον λόγο του σήματος προς τον θόρυβο κβάντισης. Καθώς ο ομοιόμορφος κβαντιστής δεν επιστρέφει την παραμόρφωση του αρχικού σήματος, την υπολογίζουμε στον παρονομαστή του κλάσματος του SQNR, ενώ για τον μη-ομοιόμορφο τη γνωρίζουμε ήδη. Για τον δεύτερο σχεδιάζουμε επίσης τη μεταβολή του SQNR σε σχέση με τον αριθμό των επαναλήψεων, μιας και χρειάζεται για το πρώτο υποερώτημα παρακάτω.

Έπειτα, χρησιμοποιούμε την συνάρτηση audioplayer[2] της MATLAB για να δημιουργήσουμε αντικείμενα (object) ήχου και να μπορούμε έτσι να ακούσουμε τα αποτελέσματα που βρήκαμε προηγουμένως με χρήση της συνάρτησης play.[3] Χρησιμοποιήθηκαν αυτές οι συναρτήσεις και όχι η wavplay[4] που προτείνεται στην εκφώνηση, καθώς η έκδοση της MATLAB που χρησιμοποιήθηκε έβγαζε προειδοποίηση (εξαιτίας μελλοντικής απομάκρυνσής της) και πρότεινε την χρήση τους αντί αυτής. Θεωρούμε ότι δεν επηρεάζει τα ζητούμενα της άσκησης, μιας και η αξιολόγηση που ζητείται στο δεύτερο υποερώτημα μπορεί να γίνει εξίσου καλά.

Τέλος, γίνεται η σχεδίαση των κυματομορφών εξόδου για κάθε περίπτωση που εξετάζουμε, ώστε να χρησιμοποιηθούν στην αξιολόγηση του τρίτου υποερωτήματος.

α) Οι τιμές του SQNR για τις περιπτώσεις του ομοιόμορφου PCM κβαντιστή είναι οι εξής:

	N=2	**N=4**	**N=8**
SQNR	1.357700	15.621504	5179.563975

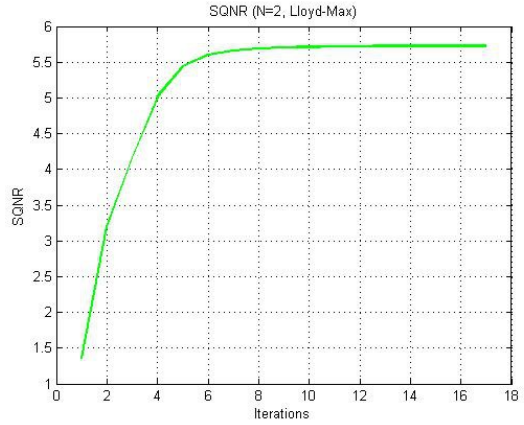
Καθώς το SQNR εκφράζει τη σχέση μεταξύ του αρχικού σήματος και της παραμόρφωσης, μπορούμε να πούμε ότι όσο μεγαλύτερο είναι, τόσο μικρότερη είναι η παραμόρφωση από το αρχικό σήμα, με αποτέλεσμα το κλάσμα να είναι μεγαλύτερο. Δηλαδή, όσο πιο μεγάλο είναι το SQNR, τόσο πιο «πιστή» είναι η κβάντιση του αρχικού σήματος.

Για τον μη-ομοιόμορφο κβαντιστή PCM έχουμε τα εξής αποτελέσματα, όσον αφορά τις επαναλήψεις που χρειάστηκε για να ικανοποιηθούν οι συνθήκες Lloyd-Max σε κάθε περίπτωση:

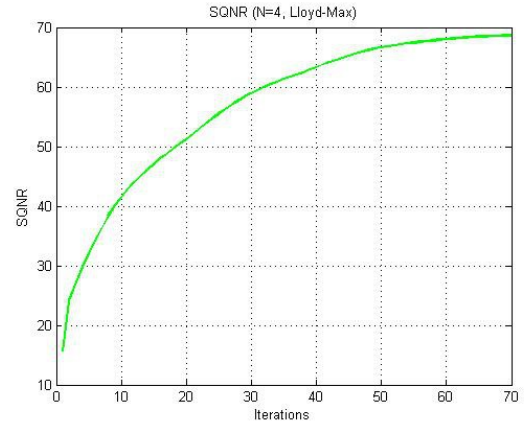
	N=2	**N=4**	**N=8**
$\mathbf{K_{max}}$	17	70	2

Η μεταβολή του SQNR σε σχέση με τις επαναλήψεις που χρειάστηκαν φαίνεται από τα ακόλουθα σχήματα (παρατίθενται με την αντίστοιχη σειρά, όπως και στον παραπάνω πίνακα):

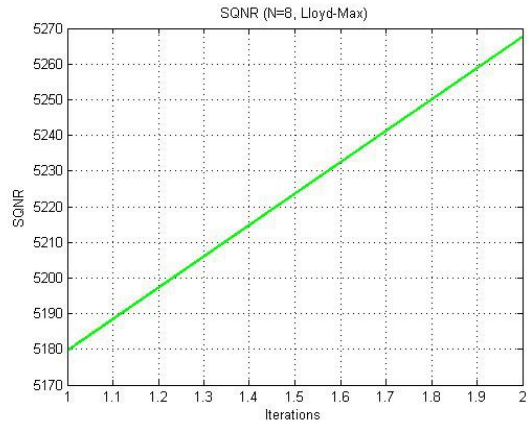
2 http://www.mathworks.com/help/matlab/ref/audioplayer.html
3 http://www.mathworks.com/help/matlab/ref/audioplayer.play.html
4 http://www.mathworks.com/help/matlab/ref/wavplay.html



Για την περίπτωση των 2 bits



Για την περίπτωση των 4 bits



Για την περίπτωση των 8 bits

Παρατηρούμε ότι και στην περίπτωση του μη-ομοιόμορφου PCM κβαντιστή, η τιμή του SQNR μεγαλώνει όσο αυξάνονται τα bits και κατ' επέκταση τα επίπεδα κβάντισης. Δηλαδή, για N=2 βλέπουμε ότι το SQNR στο τέλος της εκτέλεσης φτάνει περίπου μέχρι το 6.5 με 7, για N=4 μέχρι περίπου στο 70, ενώ για N=8 μέχρι περίπου στο 5270.

Συγκριτικά με τις τιμές του ομοιόμορφου κβαντιστή για τις αντίστοιχες περιπτώσεις, βλέπουμε ότι οι τιμές που πήραμε για τον μη-ομοιόμορφο είναι καλύτερες. Δηλαδή, για N=2 αντί για 1.4 περίπου έχουμε γύρω στο 7, και για N=4 αντί για 15.7 έχουμε γύρω στο 70. Βέβαια, στην περίπτωση των 8 bits παρατηρούμε ότι ναι μεν είναι μεγαλύτερο για την περίπτωση του μη-ομοιόμορφου, όμως η διαφορά μεταξύ τους δεν είναι ιδιαίτερα μεγάλη.

Έτσι, σαν πρώτο συμπέρασμα, θα μπορούσαμε να πούμε ότι είναι προτιμότερη η χρήση του μη-ομοιόμορφου PCM κβαντιστή που υλοποιεί και τον αλγόριθμο Lloyd-Max, κάτι που συμφωνεί και με τη θεωρία μας.

Ένα δεύτερο σχόλιο όσον αφορά τις διπλανές γραφικές παραστάσεις είναι και το ότι η τιμή του SQNR αυξάνεται από επανάληψη σε επανάληψη σε κάθε περίπτωση. Αυτό μας δείχνει ότι όντως ο αλγόριθμος Lloyd-Max βελτιώνει την όλη διαδικασία κβάντισης, με μεγάλους ή μικρότερους ρυθμούς κάθε φορά (ανάλογα και τα διαθέσιμα επίπεδα κβάντισης).

β) Αρχικά, εκτελούμε την εντολή play(initial_track) για να ακούσουμε το αρχικό μας σήμα. Έπειτα, εκτελούμε με τη σειρά τα υπόλοιπα αντικείμενα ήχου (ή κομμάτια) που έχουν δημιουργηθεί για να διαπιστώσουμε μέσω ακοής αυτά που έχουμε ήδη διαπιστώσει.

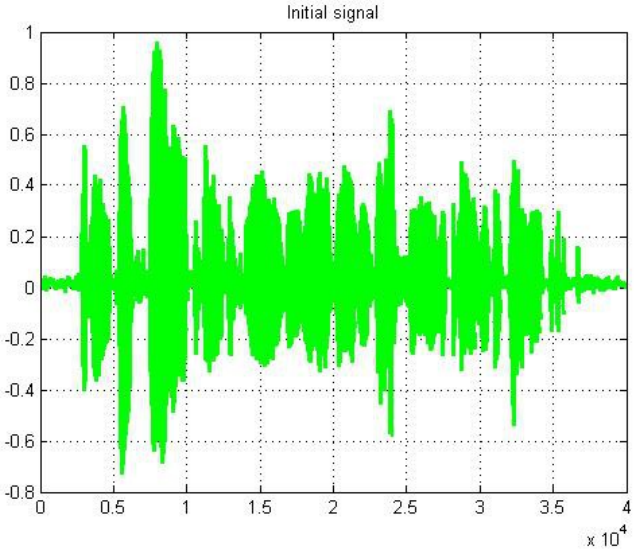
Ακούγοντας, λοιπόν, πρώτα το track2_pcm (που αντιστοιχεί, όπως υποδεικνύει το όνομά του στην περίπτωση όπου N=2 για ομοιόμορφο PCM κβαντιστή), ακούμε τη φωνή αλλά με αρκετή παραμόρφωση, που την καθιστά σχεδόν διακριτή. Βάζοντας το επόμενο, track4_pcm, ο θόρυβος έχει υποχωρήσει κάπως,

και η φωνή είναι πιο κατανοητή. Τέλος, παίζοντας το track8_pcm αυτό που ακούμε είναι παρόμοιο (αν όχι ίδιο) με το αρχικό κομμάτι. Οπότε μπορούμε να πούμε με ασφάλεια ότι θα προτιμήσουμε την κωδικοποίηση με 8 bits για μεγαλύτερη αξιοπιστία.

Ακούγοντας, τώρα, τα κομμάτια από την εκτέλεση του μη-ομοιόμορφου κβαντιστή, καταλήγουμε στο ίδιο συμπέρασμα. Το πρώτο κομμάτι, track2_lm, έχει αρκετό θόρυβο, που όμως δεν εμποδίζει τη φωνή να γίνει αντιληπτή. Το επόμενο, track4_lm, έχει σημαντικά λιγότερο θόρυβο, και η ομιλία είναι καθαρότερη. Ενώ το τελευταίο, το track8_lm, δεν έχει σχεδόν καθόλου θόρυβο και είναι ίδιο (στο αυτί τουλάχιστον) με το αρχικό. Άρα, πάλι θα προτιμήσουμε την κωδικοποίηση με περισσότερα bits.

Βέβαια, αν θέλουμε να συγκρίνουμε τα δύο διαφορετικά σχήματα, θα λέγαμε ότι στις περιπτώσεις για 2 και 4 bits σίγουρα η προτίμησή μας βρίσκεται στο μη-ομοιόμορφο κβαντιστή, μιας και ο θόρυβος είναι αισθητά λιγότερος απ' ότι στα αποτελέσματα του ομοιόμορφου. Για την περίπτωση όπου χρησιμοποιούμε 8 bits, δεν υπάρχει κάποια ουσιαστική διαφορά ως προς το άκουσμα, που να μας κάνει να επιλέξουμε τον ένα ή τον άλλο. Δηλαδή, και οι δύο έχουν παρόμοια ικανοποιητικό αποτέλεσμα.

γ) Για να συγκρίνουμε τις κυματομορφές και να μπορέσουμε να αξιολογήσουμε την κάθε μέθοδο σύμφωνα με αυτές, πρέπει πρώτα να έχουμε μια εικόνα της κυματομορφής του αρχικού σήματος. Αυτή η κυματομορφή είναι η ακόλουθη:



Κυματομορφή αρχικού σήματος ομιλίας
(από το αρχείο speech.wav)

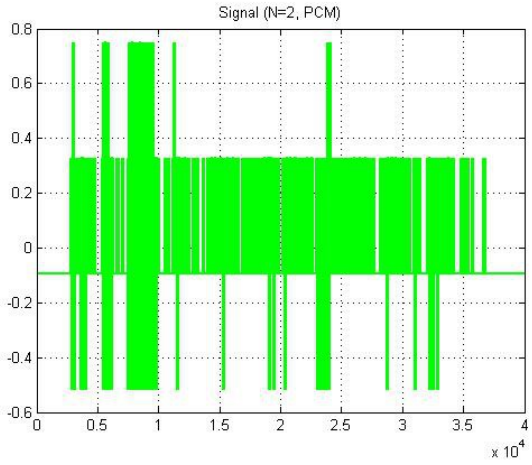
Στη συνέχεια παρατίθενται οι κυματομορφές εξόδου που αφορούν τα δύο σχήματά μας και τις διαφορετικές περιπτώσεις για το καθένα. Ο σχολιασμός τους θα γίνει πριν από αυτές, για λόγους καλύτερης παρουσίασης της αναφοράς (αποφεύγοντας μεγάλα κενά).

Μεταξύ των κυματομορφών που αφορούν τον ομοιόμορφο PCM κβαντιστή, μπορούμε να σχολιάσουμε ότι η κυματομορφή για τα 8 bits φαίνεται πιο κοντά στην αρχική κυματομορφή, με την δεύτερη των 4 bits να ακολουθεί και την πρώτη των 2 να μοιάζει λιγότερο από όλες. Αυτό στην ουσία αποτελεί οπτικοποίηση των διαφορών που εντοπίσαμε, τόσο μέσω της τιμής του SQNR, όσο και μέσω της ακοής μας.

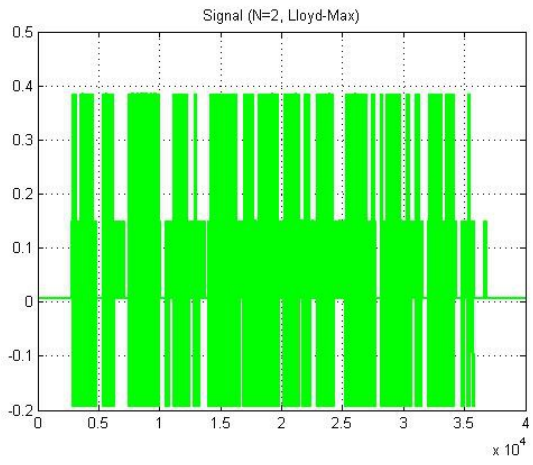
Μεταξύ των κυματομορφών που αφορούν τον μη-ομοιόμορφο PCM κβαντιστή, μπορούμε να πούμε το ίδιο, ότι δηλαδή η κυματομορφή για τα 8 bits μοιάζει περισσότερο με την αρχική, ενώ οι άλλες δύο λιγότερο. Πάλι αποτελούν οπτικοποίηση των συμπερασμάτων στα οποία καταλήξαμε προηγουμένως.

Τέλος, μεταξύ των κυματομορφών των δύο σχημάτων, μπορούμε να πούμε ότι και παραπάνω, καταλήγοντας στο ίδιο συμπέρασμα, δηλαδή στην επιλογή του μη-ομοιόμορφου κβαντιστή. Για παράδειγμα, η κυματομορφή του μη-ομοιόμορφου για 2 bits είναι πιο κοντά στην αρχική, καθώς η μέγιστη και ελάχιστη τιμή του πλάτους του σήματος είναι μικρότερες, αλλά πιο κοντά στα πλάτη της αρχικής που εμφανίζονται συχνότερα. Αντίστοιχα και για τα 4 bits, ενώ για τα 8 bits δεν υπάρχει, όπως είπαμε, ιδιαίτερη διαφορά.

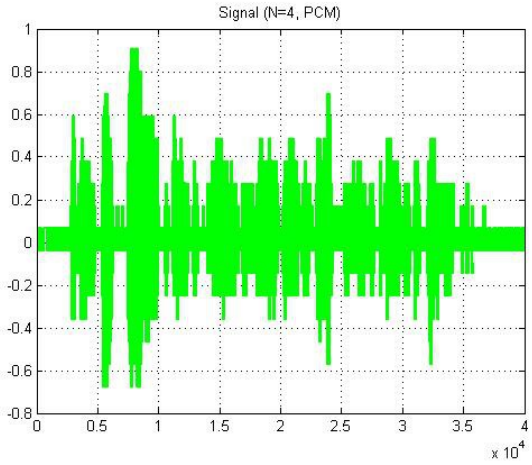
Ομοιόμορφος PCM κβαντιστής	Μη-ομοιόμορφος PCM κβαντιστής με Lloyd-Max



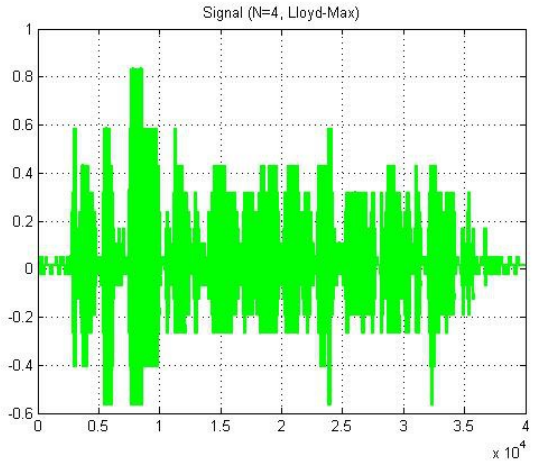
Για την περίπτωση των 2 bits



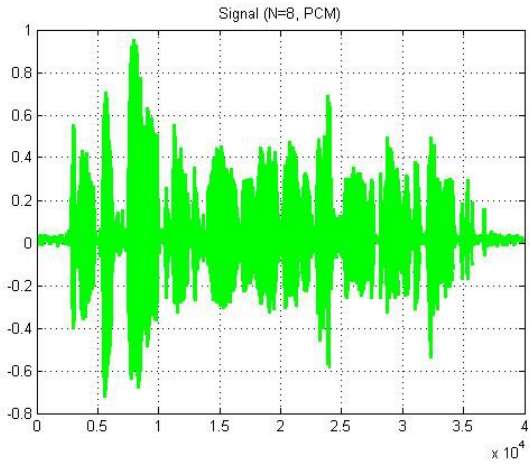
Για την περίπτωση των 2 bits



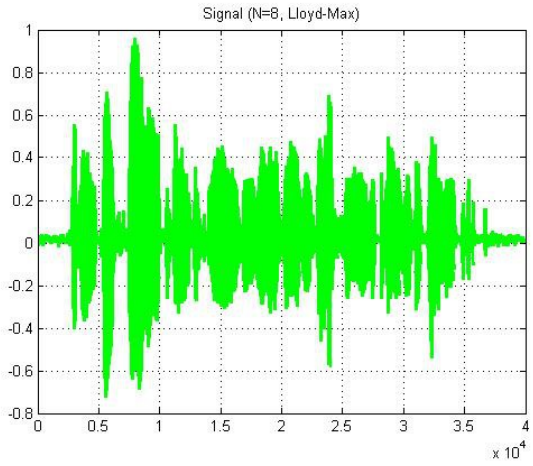
Για την περίπτωση των 4 bits



Για την περίπτωση των 4 bits



Για την περίπτωση των 8 bits



Για την περίπτωση των 8 bits

ΕΡΩΤΗΜΑ #3

Στο ερώτημα αυτό μας ζητείται να εξετάσουμε την περίπτωση του ομοιόμορφου PCM κβαντιστή για N=2, και ειδικότερα, να κάνουμε μια σύγκριση μεταξύ των πειραματικών αποτελεσμάτων και των θεωρητικών (σύμφωνα με τα δεδομένα που δίνονται).

Ο κώδικας που αφορά στο παρόν ερώτημα παρατίθεται και αυτός στο παράρτημα, με όνομα *script_1_3.m*. Επίσης, για την υλοποίηση του ερωτήματος τροποποιήθηκε λίγο ο κώδικας του πρώτου ερωτήματος για τον ομοιόμορφο PCM κβαντιστή (δηλαδή αυτός του *my_quantizer.m*), ώστε να επιστρέψει τις πιθανότητες εμφάνισης κάθε στάθμης και την παραμόρφωση του αρχικού σήματος. Ο κώδικας αυτός παρατίθεται με το όνομα *quantizer2_pcm.m*.

Η λειτουργία και των δύο κωδίκων επεξηγείται μέσω σχολίων στα ίδια τα αρχεία, παρόλ' αυτά θα περιγράψουμε συνοπτικά την λειτουργία του πρώτου κώδικα, μιας και ο δεύτερος δεν παρουσιάζει ιδιαίτερες διαφορές σε σχέση με αυτόν από τον οποίο προήλθε (και τον οποίο είχαμε παρουσιάσει στο Ερώτημα 1).

Αρχικά, λοιπόν, καθορίζεται ένα αρχείο στο οποίο θα αποθηκευτούν τα αποτελέσματα της εκτέλεσης του προγράμματος, με όνομα *results_1_3.txt*. Εν συνεχεία, αναλύεται το δοθέν σήμα από την πηγή Α και κβαντίζεται με χρήση του τροποποιημένου κβαντιστή που αναφέραμε προηγουμένως. Έτσι, εγγράφονται τα ζητούμενα πειραματικά αποτελέσματα στο αρχείο.

Έπειτα, υπολογίζονται τα όρια των περιοχών κβάντισης, ώστε να χρησιμοποιηθούν παρακάτω για τον υπολογισμό της πιθανότητας κάθε επιπέδου. Για τον υπολογισμό ακολουθούμε τη θεώρηση της εκφώνησης, ότι η κατανομή που ακολουθούν τα δείγματα ομιλίας προσεγγίζεται από την κανονική κατανομή με μέση τιμή m=-0.04 και διασπορά σ^2=0.11. Η πιθανότητα κάθε επιπέδου υπολογίζεται σύμφωνα με τη διαφορά της αθροιστικής πιθανότητας δύο διαδοχικών στάθμεων,[5] με χρήση της ενδογενούς συνάρτησης της MATLAB, normcdf.[6] Μετά υπολογίζουμε τη μέση παραμόρφωση του σήματος σύμφωνα με τον τύπο (6.5.7) του βιβλίου (δηλαδή ως άθροισμα των αντίστοιχων ολοκληρωμάτων) και με χρήση της ενδογενούς συνάρτησης normpdf[7] για τον υπολογισμό της τιμής της συνάρτησης πυκνότητας πιθανότητας. Τέλος, αποθηκεύουμε τα αποτελέσματα στο αρχείο κειμένου και αποδεσμεύουμε το handler του.

Τα αποτελέσματα που προέκυψαν έπειτα από την εκτέλεση όλων των παραπάνω είναι τα παρακάτω (τα αποτελέσματα αυτά εκτυπώνονται και στο αντίστοιχο αρχείο .txt μετά την εκτέλεση):

Πιθανότητες	**Πειραματικά αποτελέσματα**	**Θεωρητικά αποτελέσματα**
1ου επιπέδου	0.004208	0.039376
2ου επιπέδου	0.147291	0.277897
3ου επιπέδου	0.834373	0.469463
4ου επιπέδου	0.014128	0.192792
Παραμόρφωση	0.013685	0.014432

Παρατηρούμε, λοιπόν, ότι οι πιθανότητες του πρώτου και του τέταρτου επιπέδου είναι αρκετά όμοιες στο πειραματικό και θεωρητικό αποτέλεσμα, ενώ αυτές του δεύτερου και τρίτου αλλάζουν κατά πολύ, με του δεύτερου να διπλασιάζεται στο θεωρητικό και του τρίτου να υποδιπλασιάζεται.

Επίσης, παρατηρούμε ότι η παραμόρφωση είναι περίπου ίση και στα δύο αποτελέσματα, με το εξής παράδοξο να συμβαίνει. Ενώ θα περιμέναμε το αντίθετο, εντούτοις στα πειραματικά αποτελέσματα έχουμε μικρότερη παραμόρφωση απ' ότι στα θεωρητικά, γεγονός που πιθανώς οφείλεται στον τρόπο υπολογισμού της πειραματικής παραμόρφωσης.

5 Πέρα από τη θεωρία, βλ. statistics.laerd.com/statistical-guides/normal-distribution-calculations.php#betweentitle
6 www.mathworks.com/help/stats/normcdf.html
7 www.mathworks.com/help/stats/normpdf.html

ΕΡΩΤΗΜΑ #4

Ο κώδικας που υλοποιεί τον αλγόριθμο Huffman, και που μας δίνεται εξαρχής μαζί με την εκφώνηση, βρίσκεται στο παράρτημα, ως κώδικας του αρχείου *huffman.m*.

ΕΡΩΤΗΜΑ #5

Για την υλοποίηση του παρόντος ερωτήματος δημιουργήθηκε το αρχείο *script_1_5.m*, που βρίσκεται στο παράρτημα με τους κώδικες της άσκησης. Επίσης, χρησιμοποιήθηκε η παραλλαγή του *my_quantizer.m* όπως και στο τρίτο ερώτημα, δηλαδή το *quantizer2_pcm.m*, ώστε να επιστρέφει τις πιθανότητες εμφάνισης κάθε στάθμης κβάντισης, και αντίστοιχα δημιουργήθηκε το αρχείο *quantizer2_lm.m*, ως παραλλαγή του κώδικα *Lloyd_Max.m* που να επιστρέφει τις ανάλογες πιθανότητες.

Ας περιγράψουμε τη λειτουργία του script_1_5.m. Αρχικά φορτώνεται το σήμα προς κωδικοποίηση, και έπειτα γίνεται η κβάντιση για N=4 και N=8 με τους τρεις διαφορετικούς τρόπους που υποδεικνύονται: με τα δύο σχήματα και τη ρουτίνα Huffman. Τα δύο σχήματα επιστρέφουν τα διανύσματα πιθανοτήτων που απαιτεί η Huffman για να λειτουργήσει.

Στη συνέχεια, υπολογίζουμε την εντροπία για κάθε περίπτωση με χρήση των πιθανοτήτων που βρήκαμε λίγο πιο πριν, σύμφωνα με τον τύπο (6.1.1) που βρίσκεται στη σελίδα 302 του βιβλίου και με βάση λογαρίθμου το δύο. Υπολογίζουμε επίσης το μέσο μήκος του κώδικα σύμφωνα με τον τύπο (6.3.1) του βιβλίου, ώστε να μπορέσουμε να υπολογίσουμε την αποδοτικότητα.

Η αποδοτικότητα προκύπτει μέσω του λόγου της εντροπίας (της ποσότητας της πληροφορίας, δηλαδή) προς το μέσο μήκος του κώδικα επί τον λογάριθμο βάσης δύο του δύο, δηλαδή επί ένα (καθώς η κωδικοποίηση γίνεται με δύο σύμβολα, το μηδέν και το ένα).[8] Έτσι, η αποδοτικότητα του κώδικα Huffman για κάθε ένα από τα εξεταζόμενα σχήματα κωδικοποίησης είναι:

	N=4	N=8
Ομοιόμορφο PCM	0.977366	0.995519
Μη-ομοιόμορφο PCM με Lloyd-Max	0.981963	0.994981

Παρατηρούμε ότι για N=8 η αποδοτικότητα του Huffman είναι λίγο μεγαλύτερη, συγκριτικά με την περίπτωση όπου N=4. Από την άλλη, παρατηρούμε ότι, ενώ για N=4 ο Huffman με ομοιόμορφο κβαντιστή έχει μικρότερη αποδοτικότητα σε σχέση με τον μη-ομοιόμορφο, για N=8 συμβαίνει το αντίθετο. Σε κάθε περίπτωση, βέβαια, η αποδοτικότητα είναι πολύ υψηλή, γεγονός αναμενόμενο, μιας και ξέρουμε από τη θεωρία ότι ο αλγόριθμος Huffman έχει επίδοση κοντά στο φράγμα της εντροπίας.[9]

ΕΡΩΤΗΜΑ #6

Στο ερώτημα αυτό ως πηγή έχουμε το κείμενο της πηγής Β, που περιέχεται στο αρχείο *keimeno.txt* και το οποίο είναι το εξής:

> *"In computer science and electrical engineering, Lloyd's algorithm, also known as Voronoi iteration or relaxation, is an algorithm named after Stuart P. Lloyd for finding evenly-spaced sets of points in subsets of Euclidean spaces, and partitions of these subsets into well-shaped and uniformly sized convex cells. Like the closely related k-means clustering algorithm, it repeatedly finds the centroid of each set in the partition, and then re-partitions the input according to which of these centroids is closest. However, Lloyd's algorithm differs from k-means clustering in that*

8 Εκτός των ήδη γνωστών, βλ. www.cs.unm.edu/~storm/cs530/Coding.html

9 J. G. Proakis & M. Salehi, «Συστήματα Τηλεπικοινωνιών», σ.308.

its input is a continuous geometric region rather than a discrete set of points. Thus, when re-partitioning the input, Lloyd's algorithm uses Voronoi diagrams rather simply determining the nearest center to each of a finite set of points as the k-means algorithm does."

Ζητείται να υλοποιήσουμε μια συνάρτηση που να επιστρέφει το αλφάβητό του, την πιθανότητα εμφάνισης κάθε συμβόλου του αλφαβήτου και την εντροπία της πηγής. Επίσης, ζητείται να κωδικοποιηθεί το κείμενο με χρήση της συνάρτησης Huffman και να μετρηθεί η αποδοτικότητα του κώδικα. Όλα τα παραπάνω γίνονται στον κώδικα που περιλαμβάνεται στο αρχείο *function_1_6.m*, που βρίσκεται στο παράρτημα της αναφοράς.

Η λειτουργία του κώδικα επεξηγείται λεπτομερώς μέσω σχολίων στον ίδιο τον κώδικα, όμως θα την περιγράψουμε εν συντομία και εδώ. Αρχικά, διαβάζονται οι χαρακτήρες του κειμένου, μετράται η συχνότητα εμφάνισής τους και στη συνέχεια υπολογίζεται η πιθανότητα καθενός από αυτούς. Παράλληλα, αποθηκεύεται και το αλφάβητο που ζητείται. Έπειτα, γίνεται ο υπολογισμός της εντροπίας, όπως και στο προηγούμενο ερώτημα.

Τέλος, κωδικοποιούμε το κείμενο κάνοντας χρήση της Huffman, και με είσοδο τις πιθανότητες των χαρακτήρων, και στη συνέχεια, υπολογίζουμε την αποδοτικότητα του κώδικα αυτού, όπως και πριν. Τα αποτελέσματα που παίρνουμε έπειτα από την κλήση της συνάρτησης[10] είναι τα ακόλουθα:

Αλφάβητο
',-.EHILPSTVabcdefghiklmnoprstuvwxyz

Πιθανότητα εμφάνισης συμβόλου[11]							
0.1506	0.0035	0.0106	0.0082	0.0059	0.0012	0.0012	0.0012
0.0059	0.0012	0.0012	0.0012	0.0024	0.0553	0.0024	0.0294
0.0294	0.0929	0.0200	0.0200	0.0318	0.0741	0.0059	0.0341
0.0200	0.0682	0.0647	0.0188	0.0529	0.0647	0.0800	0.0176
0.0035	0.0059	0.0024	0.0106	0.0012			

Εντροπία	4.2730	**Αποδοτικότητα Huffman**	0.993440

10 Με την εντολή: `>> [a,p,e] = function_1_6;`

11 Παρατίθενται με τη σειρά που εμφανίζονται στο αλφάβητο, π.χ. το κενό έχει πιθανότητα 0.1506, το ' έχει 0.0035, το κόμμα έχει 0.0106, κοκ.

ΠΑΡΑΡΤΗΜΑ ΚΩΔΙΚΩΝ

Στο σημείο αυτό παρατίθενται όλοι οι κώδικες MATLAB που υλοποιήθηκαν και χρησιμοποιήθηκαν για την εκπόνηση της άσκησης. Παρατίθενται περίπου με τη σειρά που προσδιορίζεται από τα ερωτήματα. Παρουσιάζεται αρχικά το όνομα του αρχείου (με κατάληξη .m) και στη συνέχεια ο ίδιος ο κώδικας, με αριθμημένες γραμμές και μορφοποίηση παρόμοια με τον επεξεργαστή της MATLAB.

• • • • • •

Όνομα αρχείου: *my_quantizer.m*

```
1.  function [xq,centers] = my_quantizer(x,N,min_value,max_value) % ---------
2.  % INPUT ------------------------------------------------------------------
3.  %   x: vector of input signal
4.  %   N: number of bits to be used
5.  %   min_value: minimum acceptable value of input signal
6.  %   max_value: maximum acceptable value of input signal
7.  % OUTPUT -----------------------------------------------------------------
8.  %   xq: vector of output signal
9.  %   centers: centers of quantization segments
10. % ------------------------------------------------------------------------
11.
12. % Number of bits used.
13. v = N;
14. % Levels of quantization.
15. quant_levels = 2^v;
16. % Initialization of vector xq.
17. xq = zeros(length(x),1);
18. % Quantization step Δ.
19. quant_step = (abs(min_value)+max_value)/quant_levels;
20. % Calculation of centers.
21. centers = zeros(quant_levels,1); % Initialization.
22. for i=1:quant_levels
23.     centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
24. end
25. % Calculation of output signal. (quantization)
26. q_max_value = 1; q_min_value = quant_levels;
27. for i=1:length(x)
28.     % If x(i) is greater or equal to max_value.
29.     if (x(i) >= max_value), xq(i) = q_max_value;
30.     % If x(i) is smaller than min_value.
31.     elseif (x(i) < min_value), xq(i) = q_min_value;
32.     else % For every other case inbetween.
33.         for n=1:quant_levels
34.             if (x(i) >= (centers(n)-(quant_step/2)) && ...
35.                 x(i) < (centers(n)+(quant_step/2)))
36.                 xq(i) = n;
37.             end
38.         end
39.     end
40. end
```

Όνομα αρχείου: *Lloyd_Max.m*

```
1.  function [xq,centers,D] = Lloyd_Max(x,N,min_value,max_value) % ----------
2.  % INPUT ------------------------------------------------------------------
3.  %   x: vector of input signal
```

```
4.    %   N: number of bits to be used
5.    %   min_value: minimum acceptable value of input signal
6.    %   max_value: maximum acceptable value of input signal
7.    % OUTPUT ----------------------------------------------------------------
8.    %   xq: encoded vector of output signal
9.    %   centers: centers of quantization segments
10.   %   D: vector of signal's distortion at every repetition
11.   % ------------------------------------------------------------------------
12.
13.   % Number of bits used.
14.   v = N;
15.   % Levels of quantization.
16.   quant_levels = 2^v;
17.   % Initialization of vector xq.
18.   xq = zeros(length(x),1);
19.   % Quantization step Δ.
20.   quant_step = (abs(min_value)+max_value)/quant_levels;
21.   % Calculation of centers.
22.   centers = zeros(quant_levels,1); % Initialization.
23.   for i=1:quant_levels
24.       centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
25.   end
26.   % Loop for Lloyd-Max algorithm.
27.   T = zeros((quant_levels+1),1); % Initialization of segments' limits.
28.   counter = 1; % Initialization of iterations' counter.
29.   previous_distortion = 0; % Initialization of previous distortion.
30.   while 1
31.       % [1] Calculation of new segments' limits.
32.       T(1) = max_value; % Higher level limit.
33.       for i=2:quant_levels % Middle levels.
34.           T(i) = (centers(i)+centers(i-1))/2;
35.       end
36.       T(quant_levels+1) = min_value; % Lower level limit.
37.
38.       % [2] Calculation of output signal & distortion.
39.       q_max_value = 1; q_min_value = quant_levels;
40.       for i=1:length(x)
41.           % If x(i) is greater or equal to max_value.
42.           if (x(i) >= max_value)
43.               xq(i) = q_max_value;
44.           % If x(i) is smaller than min_value.
45.           elseif (x(i) <= min_value)
46.               xq(i) = q_min_value;
47.           else % For every other case inbetween.
48.               for n=1:quant_levels
49.                   if ((x(i) <= T(n)) && (x(i) > T(n+1)))
50.                       xq(i) = n;
51.                   end
52.               end
53.           end
54.       end
55.       % Check if there are any zeros.
56.       if (all(xq)) % If not, calculate the distortion.
57.           D(counter) = mean((x-centers(xq)).^2); %#ok<AGROW>
58.       else % If yes, print out an error message.
59.           fprintf('Error: there are zeros as index.\n');
60.       end
61.
62.       % [4] Criterion check. If fulfilled, stop the loop.
63.       difference = abs(D(counter)-previous_distortion);
64.       if (difference < eps('single'))
65.           break;
66.       else % Else, store current distortion for the next comparison.
67.           previous_distortion = D(counter);
```

```
        end

        % [3] New levels of quantization.
        temp_sum = zeros(quant_levels,1); % Temporary summary.
        temp_counter = zeros(quant_levels,1); % Number of values in each level.
        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

        % Increment of iteration counter.
        counter = counter + 1;
    end
    % Printing out the final number of iterations.
    fprintf('(%d bits) Kmax = %d\n',N,counter);
```

Όνομα αρχείου: *script_1_2.m*

```
% Loading initial signal.
[y,fs,N] = wavread('speech.wav');

% Quantization using 2,4 and 8 bits for both quantizers.
[xq2_pcm,centers2_pcm] = my_quantizer(y,2,min(y),max(y));
[xq4_pcm,centers4_pcm] = my_quantizer(y,4,min(y),max(y));
[xq8_pcm,centers8_pcm] = my_quantizer(y,8,min(y),max(y));
[xq2_lm,centers2_lm,D2_lm] = Lloyd_Max(y,2,min(y),max(y));
[xq4_lm,centers4_lm,D4_lm] = Lloyd_Max(y,4,min(y),max(y));
[xq8_lm,centers8_lm,D8_lm] = Lloyd_Max(y,8,min(y),max(y));

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

```
28.
29.   % Saving audio objects for each quantization case.
30.   initial_track = audioplayer(y,fs);
31.   track2_pcm = audioplayer(centers2_pcm(xq2_pcm),fs);
32.   track4_pcm = audioplayer(centers4_pcm(xq4_pcm),fs);
33.   track8_pcm = audioplayer(centers8_pcm(xq8_pcm),fs);
34.   track2_lm = audioplayer(centers2_lm(xq2_lm),fs);
35.   track4_lm = audioplayer(centers4_lm(xq4_lm),fs);
36.   track8_lm = audioplayer(centers8_lm(xq8_lm),fs);
37.
38.   % Plotting all signals.
39.   figure; plot(y,'g','LineWidth',2); title('Initial signal'); grid on;
40.   figure; plot(centers2_pcm(xq2_pcm),'g','LineWidth',2);
41.   title('Signal (N=2, PCM)'); grid on;
42.   figure; plot(centers4_pcm(xq4_pcm),'g','LineWidth',2);
43.   title('Signal (N=4, PCM)'); grid on;
44.   figure; plot(centers8_pcm(xq8_pcm),'g','LineWidth',2);
45.   title('Signal (N=8, PCM)'); grid on;
46.   figure; plot(centers2_lm(xq2_lm),'g','LineWidth',2);
47.   title('Signal (N=2, Lloyd-Max)'); grid on;
48.   figure; plot(centers4_lm(xq4_lm),'g','LineWidth',2);
49.   title('Signal (N=4, Lloyd-Max)'); grid on;
50.   figure; plot(centers8_lm(xq8_lm),'g','LineWidth',2);
51.   title('Signal (N=8, Lloyd-Max)'); grid on;
```

Όνομα αρχείου: *quantizer2_pcm.m*

```
1.    function [xq,centers,p,D] = quantizer2_pcm(x,N,min_value,max_value) % ---
2.    % INPUT ------------------------------------------------------------------
3.    %   x: vector of input signal
4.    %   N: number of bits to be used
5.    %   min_value: minimum acceptable value of input signal
6.    %   max_value: maximum acceptable value of input signal
7.    % OUTPUT -----------------------------------------------------------------
8.    %   xq: vector of output signal
9.    %   centers: centers of quantization segments
10.   %   p: probability of each symbol
11.   %   D: distortion after encoding
12.   % -------------------------------------------------------------------------
13.
14.   % Number of bits used.
15.   v = N;
16.   % Levels of quantization.
17.   quant_levels = 2^v;
18.   % Initialization of vector xq.
19.   xq = zeros(length(x),1);
20.   % Quantization step Δ.
21.   quant_step = (abs(min_value)+max_value)/quant_levels;
22.   % Calculation of centers.
23.   centers = zeros(quant_levels,1); % Initialization.
24.   for i=1:quant_levels
25.       centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
26.   end
27.   % Initialization for frequency counter.
28.   frequency = zeros(quant_levels,1);
29.   % Calculation of output signal. (quantization)
30.   q_max_value = 1; q_min_value = quant_levels;
31.   for i=1:length(x)
32.       % If x(i) is greater or equal to max_value.
33.       if (x(i) >= max_value)
34.           xq(i) = q_max_value;
35.           frequency(1) = frequency(1) + 1;
```

```
        % If x(i) is smaller than min_value.
        elseif (x(i) < min_value)
            xq(i) = q_min_value;
            frequency(quant_levels) = frequency(quant_levels) + 1;
        else % For every other case in-between.
            for n=1:quant_levels
                if (x(i) >= (centers(n)-(quant_step/2)) && ...
                    x(i) < (centers(n)+(quant_step/2)))
                    xq(i) = n;
                    frequency(n) = frequency(n) + 1;
                end
            end
        end
    end
    % Calculation of probability for each level.
    p = frequency./length(x);
    % Calculation of distortion.
    D = mean((x-centers(xq)).^2);
```

Όνομα αρχείου: *script_1_3.m*

```
    % Creating a file to save results.
    file_id = fopen('results_1_3.txt','w+');

    % Loading initial signal.
    [y,fs,N] = wavread('speech.wav');
    % Quantization using PCM quantizer and 2 bits.
    [xq,centers,p_e,D_e] = quantizer2_pcm(y,2,min(y),max(y));

    % Printing out experimental results.
    fprintf(file_id,'---------------------------------\n');
    fprintf(file_id,'      EXPERIMENTAL RESULTS       \n');
    fprintf(file_id,'---------------------------------\n');
    fprintf(file_id,'Probabilities ------------------\n');
    for i=1:4
        fprintf(file_id,'\t#%d level: %f\n',i,p_e(i));
    end
    fprintf(file_id,'Distortion: %f\n',D_e);

    % Calculation of segment limits.
    T = zeros(((2^2)+1),1); % Initialization.
    T(1) = max(y); % Higher level limit.
    for i=2:(2^2) % Middle levels.
        T(i) = (centers(i)+centers(i-1))/2;
    end
    T(5) = min(y); % Lower level limit.
    % Specifying mu and sigma.
    m = -0.04; s = sqrt(0.11);
    % Initilization.
    p_t = zeros((2^2),1);
    for i=1:(2^2) % Calculating theoritical probabilities.
        p_t(i) = normcdf(T(i),m,s) - normcdf(T(i+1),m,s);
    end
    % Calculation of mean distortion.
    syms y; D_t = 0; % Initialization.
    for i=4:-1:1 % Calculation of integrals.
        f = ((y-centers(i))^2)*normpdf(y,m,s);
        D_t = D_t + double(int(f,T(i+1),T(i)));
    end

    % Printing out theoritical results.
    fprintf(file_id,'---------------------------------\n');
```

```
fprintf(file_id,'       THEORITICAL RESULTS      \n');
fprintf(file_id,'--------------------------------\n');
fprintf(file_id,'Probabilities ------------------\n');
for i=1:4
    fprintf(file_id,'\t#%d level: %f\n',i,p_t(i));
end
fprintf(file_id,'Distortion: %f\n',D_t);

% Closing text file.
fclose(file_id);
```

Όνομα αρχείου: *quantizer2_lm.m*

```
function [xq,centers,p,D] = quantizer2_lm(x,N,min_value,max_value) % ----
% INPUT ------------------------------------------------------------------
%   x: vector of input signal
%   N: number of bits to be used
%   min_value: minimum acceptable value of input signal
%   max_value: maximum acceptable value of input signal
% OUTPUT -----------------------------------------------------------------
%   xq: encoded vector of output signal
%   centers: centers of quantization segments
%   p: probability of each symbol
%   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 Δ.
quant_step = (abs(min_value)+max_value)/quant_levels;
% Calculation of centers.
centers = zeros(quant_levels,1); % Initialization.
for i=1:quant_levels
    centers(i) = max_value-(2*(i-1)+1)*(quant_step/2);
end
% Loop for Lloyd-Max algorithm.
T = zeros((quant_levels+1),1); % Initialization of segments' limits.
counter = 1; % Initialization of iterations' counter.
previous_distortion = 0; % Initialization of previous distortion.
while 1
    % [1] Calculation of new segments' limits.
    T(1) = max_value; % Higher level limit.
    for i=2:quant_levels % Middle levels.
        T(i) = (centers(i)+centers(i-1))/2;
    end
    T(quant_levels+1) = min_value; % Lower level limit.

    % [2] Calculation of output signal & distortion.
    q_max_value = 1; q_min_value = quant_levels;
    for i=1:length(x)
        % If x(i) is greater or equal to max_value.
        if (x(i) >= max_value)
            xq(i) = q_max_value;
        % If x(i) is smaller than min_value.
        elseif (x(i) <= min_value)
            xq(i) = q_min_value;
        else % For every other case inbetween.
            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); %#ok<AGROW>
        else % If yes, print out an error message.
            fprintf('Error: there are zeros as index.\n');
        end

        % [4] Criterion check. If fulfilled, stop the loop.
        difference = abs(D(counter)-previous_distortion);
        if (difference < eps('single'))
            break;
        else % Else, store current distortion for the next comparison.
            previous_distortion = D(counter);
        end

        % [3] New levels of quantization.
        temp_sum = zeros(quant_levels,1); % Temporary summary.
        temp_counter = zeros(quant_levels,1); % Number of values in each level.
        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

        % Increment of iteration counter.
        counter = counter + 1;
    end
    % Calculating probabilities.
    frequency = zeros(quant_levels,1);
    for i=1:quant_levels
        for k=1:length(xq)
            if (i == xq(k))
                frequency(i) = frequency(i) + 1;
            end
        end
    end
    p = frequency./length(xq);
    % Printing out the final number of iterations.
    fprintf('(%d bits) Kmax = %d\n',N,counter);
```

Όνομα αρχείου: *huffman.m*

```
function [code,len]=huffman(p);
% Huffman Coding.
%       [code,len]=huffman(p),
%       INPUTS
%       p(vector): contains the probabilities of each symbol
%       OUTPUTS
%       code(vector): the code for each symbol (in ascii format)
%       len(vector): the number of bits needed for each code

p = p(:)';

if length(find(p<0))~=0,
    error('Not a probability vector, negative component(s)')
end;

if abs(sum(p)-1)>10e-10,
    error('Not a probability vector, components do not add up to 1')
end;
n=length(p);
q=p;
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;

for i=1:n-1,
    c(i,:)=blanks(n*n);
end;

c(n-1,n)='0';
c(n-1,2*n)='1';

for i=2:n-1,
    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,
    code(i,1:n)=c(1,n*(find(m(1,:)==i)-1)+1:find(m(1,:)==i)*n);
    len(i)=length(find(abs(code(i,:))~=32));
end;
```

Όνομα αρχείου: *script_1_5.m*

```
% Loading initial signal.
[y,fs,N] = wavread('speech.wav');

% Quantization using PCM quantizer.
[xq4_pcm,centers4_pcm,p4_pcm,D4_pcm] = quantizer2_pcm(y,4,min(y),max(y));
[xq8_pcm,centers8_pcm,p8_pcm,D8_pcm] = quantizer2_pcm(y,8,min(y),max(y));
```

```
7.    % Quantization using PCM with Lloyd-Max quantizer.
8.    [xq4_lm,centers4_lm,p4_lm,D4_lm] = quantizer2_lm(y,4,min(y),max(y));
9.    [xq8_lm,centers8_lm,p8_lm,D8_lm] = quantizer2_lm(y,8,min(y),max(y));
10.   % Quantization using Huffman.
11.   [~,len4_pcm] = huffman(p4_pcm);
12.   [~,len8_pcm] = huffman(p8_pcm);
13.   [~,len4_lm] = huffman(p4_lm);
14.   [~,len8_lm] = huffman(p8_lm);
15.
16.   % Calculation of entropy.
17.   temp_entr4_pcm = zeros(length(p4_pcm),1);
18.   temp_entr8_pcm = zeros(length(p8_pcm),1);
19.   temp_entr4_lm = zeros(length(p4_lm),1);
20.   temp_entr8_lm = zeros(length(p8_lm),1);
21.   for i=1:length(p4_pcm)
22.       if (p4_pcm(i) ~= 0)
23.           temp_entr4_pcm(i) = p4_pcm(i)*log2(1/p4_pcm(i));
24.       end
25.   end
26.   for i=1:length(p8_pcm)
27.       if (p8_pcm(i) ~= 0)
28.           temp_entr8_pcm(i) = p8_pcm(i)*log2(1/p8_pcm(i));
29.       end
30.   end
31.   for i=1:length(p4_lm)
32.       if (p4_lm(i) ~= 0)
33.           temp_entr4_lm(i) = p4_lm(i)*log2(1/p4_lm(i));
34.       end
35.   end
36.   for i=1:length(p8_lm)
37.       if (p8_lm(i) ~= 0)
38.           temp_entr8_lm(i) = p8_lm(i)*log2(1/p8_lm(i));
39.       end
40.   end
41.   entropy4_pcm = sum(temp_entr4_pcm);
42.   entropy8_pcm = sum(temp_entr8_pcm);
43.   entropy4_lm = sum(temp_entr4_lm);
44.   entropy8_lm = sum(temp_entr8_lm);
45.
46.   % Calculation of mean code-word length.
47.   len4_pcm = len4_pcm'; len8_pcm = len8_pcm';
48.   len4_lm = len4_lm'; len8_lm = len8_lm';
49.   L4_pcm = sum(len4_pcm.*p4_pcm);
50.   L8_pcm = sum(len8_pcm.*p8_pcm);
51.   L4_lm = sum(len4_lm.*p4_lm);
52.   L8_lm = sum(len8_lm.*p8_lm);
53.
54.   % Calculation of efficiency.
55.   efficiency4_pcm = entropy4_pcm/L4_pcm;
56.   efficiency8_pcm = entropy8_pcm/L8_pcm;
57.   efficiency4_lm = entropy4_lm/L4_lm;
58.   efficiency8_lm = entropy8_pcm/L8_lm;
59.
60.   % Printing out results.
61.   fprintf('-----------------------------------\n');
62.   fprintf('       HUFFMAN EFFICIENCIES        \n');
63.   fprintf('-----------------------------------\n');
64.   fprintf('\tPCM (4 bits): %f\n',efficiency4_pcm);
65.   fprintf('\tPCM (8 bits): %f\n',efficiency8_pcm);
66.   fprintf('\tLM (4 bits): %f\n',efficiency4_lm);
67.   fprintf('\tLM (8 bits): %f\n',efficiency8_lm);
```

Όνομα αρχείου: *function_1_6.m*

```
function [alphabet,probability,entropy] = function_1_6 % ----------------
% INPUT ------------------------------------------------------------------
%   (none)
% OUTPUT -----------------------------------------------------------------
%   alphabet: set of letters in source text
%   probability: probability of each alphabet symbol
%   entropy: measure of information in source text
% ------------------------------------------------------------------------

% Opening given text file.
file_id = fopen('keimeno.txt');
% Reading every character.
characters = fscanf(file_id,'%c');
% Closing text file handler.
fclose(file_id);
% Counting read characters.
number_of_chars = length(characters);
% Converting characters to ASCII.
ascii_chars = abs(characters);
% Initializing frequency and probability.
frequency = zeros(128,1);
% Calculation of every character's frequency.
for i=1:number_of_chars
    for n=1:128
        if (ascii_chars(i) == (n-1))
            frequency(n) = frequency(n) + 1;
        end
    end
end
% Calculation of probability.
temp_prob = frequency./number_of_chars;
% Finding nonzero values of probability.
pos_ind = find(temp_prob);
% Returning text's alphabet. (ASCII = pos-1)
alphabet = char(pos_ind-1);
% Returning probability for each letter of alphabet.
probability = temp_prob(pos_ind);
% Calculation of entropy.
temp_entr = zeros(length(pos_ind),1);
for i=1:length(pos_ind)
    temp_entr(i) = probability(i)*log2(1/probability(i));
end
% Returning text's entropy.
entropy = sum(temp_entr);
% Encoding text with Huffman.
[~,len] = huffman(probability);
% Reversing len for next calculation.
len = len';
% Calculating mean code-word length.
L = sum(len.*probability);
% Calculation of Huffman's efficiency.
efficiency = entropy/L;
% Printing out the result.
fprintf('Huffman efficiency: %f\n',efficiency);
```