Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Συστήματα Πολυμέσων

Απλοποιημένος codec MP3

Κούτση Χριστίνα

ΑΕΜ: 9872

cvkoutsi@ece.auth.gr

Κερεμίδης Αντώνιος

ΑΕΜ: 9717 keremidis@ece.auth.gr

Φεβρουάριος, 2023

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η παρούσα εργασία στα Συστήματα Πολυμέσων υλοποιεί έναν απλοποιημένο codec MP3. Η εργασία υλοποιήθηκε σε Python και τα αρχεία κώδικα που απαιτούνται σε κάθε ενότητα αναφέρονται στην αρχή της ενότητας

Κεφάλαιο 2

Sub-band Filtering

Στην πρώτη ενότητα της εργασίας μας ζητείται η δημιουργία φίλτρων H και G και την ανάλυση και την ανακατασκευή του σήματος σε μπάντες. Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιούνται τα αρχεία:

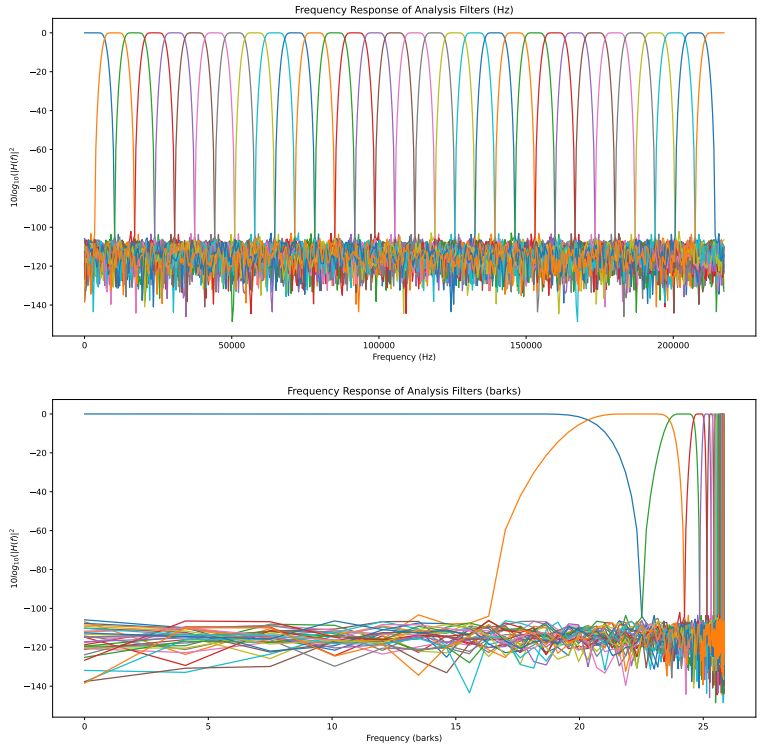
- Section1-Test.py
- codec0.py
- mp3.py
- frame.py

2.1 Υπολογισμός φίλτρων H και G

Για τον υπολογισμό των φίλτρων H και Γ χρησιμοποιήθηκαν οι συναρτήσεις frame_sub_analysis και frame_sub_synthesis του αρχείου frame.py

2.2 Απόκριση Συχνότητας φίλτρου (Hz,barks)

Αφού υπολογίσαμε το φίλτρο H, σχεδιάζουμε το μέτρο των συναρτήσεων μεταφοράς ως προς τη συχνότητα σε Hz και σε Barks:



2.3 Κατασκευή coder

Σκοπός μας με την κατασκευή του coder είναι η δημιουργία του Y_{tot}. Ο Y_{tot} είναι ένας πίνακας διάστασης (#*frames*xN)xM, όπου:

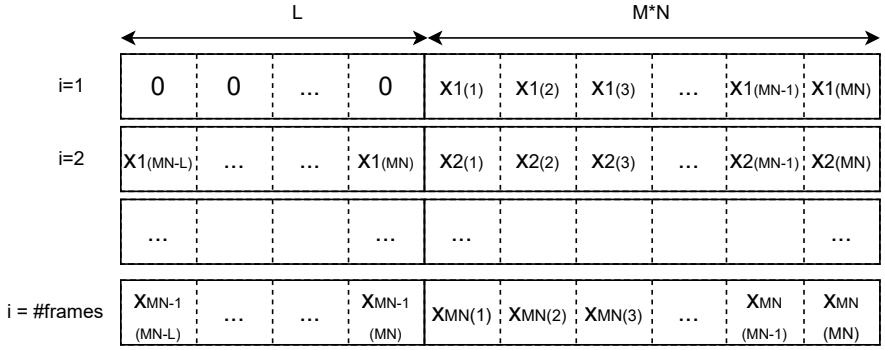
- #*frames* = length(wavin) / M*N
- N=36 : Αριθμός δειγμάτων σε κάθε frame
- M=32 : Αριθμός ψηφιακών φίλτρων

ο οποίος υπολογίζει την απόκριση των ψηφιακών φίλτρων ανάλυσης H για κάποια δείγματα του wavin που καθορίζονται από έναν buffer. Η απόκριση των φίλτρων υπολογίζεται χρησιμοποιώντας την συνάρτηση frame_sub_analysis.

Για να πετύχουμε επομένως το παραπάνω, θα πρέπει αρχικά να δημιουργήσουμε έναν buffer, ο οποίος θα επιλέγει τα δείγματα από το σήμα τα οποία θα χρησιμοποιηθούν για τον υπολογισμό του Y_{tot} σε κάθε επανάληψη. Η βασική αρχή λειτουργίας του buffer είναι ότι θέλουμε να λαμβάνει M*N καινούρια δείγματα από το σήμα σε κάθε επανάληψη με overhead L. Επομένως, χρησιμοποιούμε buffer διάστασης (M*N + L), ο

οποίος λειτουργεί επαναληπτικά για τόσες επαναλήψεις, όσα και ο αριθμός των frames που περιέχει το σήμα.

Η βασική λειτουργία του buffer απεικονίζεται παρακάτω:

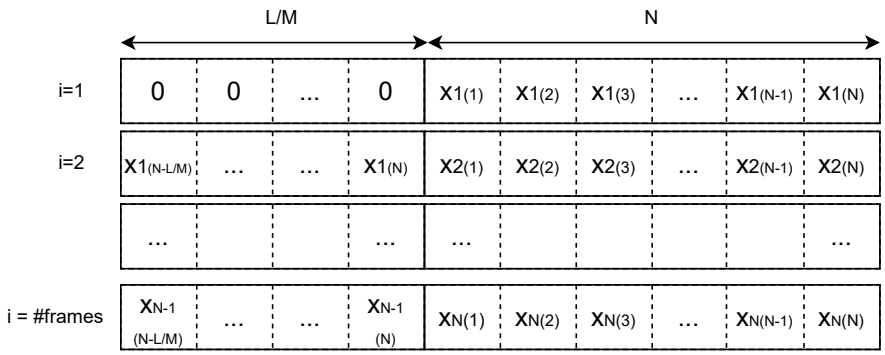


2.4 Κατασκευή decoder

Ο decoder, πραγματοποιεί την αντίστροφη διαδικασία από τον coder, δηλαδή λαμβάνει το Y_{tot} και παράγει έναν πίνακα $\hat{x}$, ο οποίος έχει ίδια διάσταση με το αρχικό σήμα εισόδου. Ο decoder υπολογίζει την απόκριση των φίλτρων σύνθεσης G για κάποια δείγματα του Y_{tot} που καθορίζονται και πάλι από έναν buffer. Η απόκριση των φίλτρων υπολογίζεται χρησιμοποιώντας την συνάρτηση frame_sub_sunthesis.

Η λειτουργία του βυφφερ είναι παρόμοια με αυτόν που χρησιμοποιείται στον coder, με διαφορά στο μέγεθος του buffer και στο overhead με το οποίο λειτουργεί. Συγκεκριμένα, ο buffer λαμβάνει (N x M) καινούρια δείγματα από το Y_{tot} (2 διαστάσεις) με overhead ($\frac{L}{M}$ x M). Επομένως έχει διάσταση (N + $\frac{L}{M}$) x M.

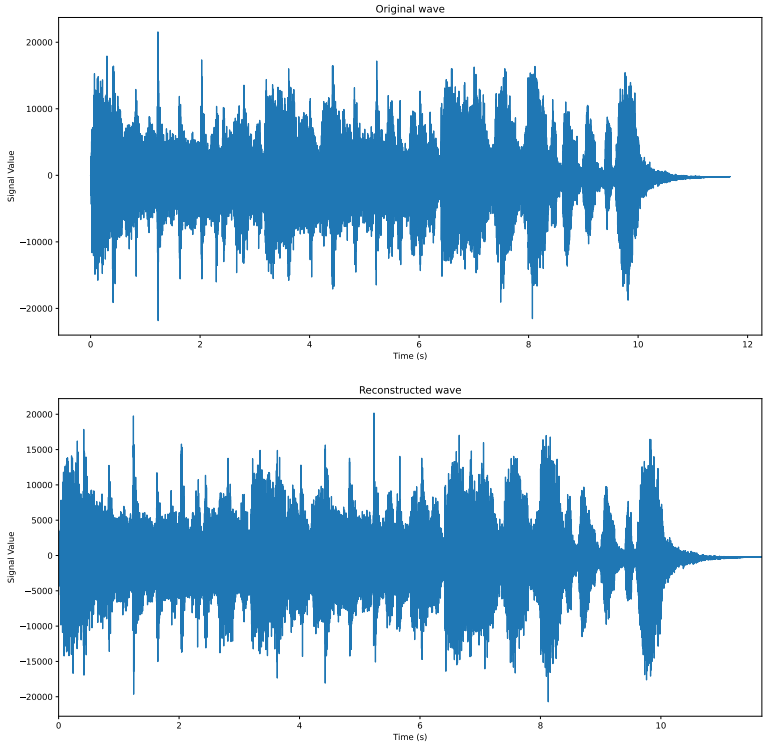
Η βασική λειτουργία του buffer κατά την οριζόντια διάσταση απεικονίζεται παρακάτω:



Όπου #frames = (οριζόντια διάσταση του Y_{tot}) // N

2.5 Κωδικοποίηση και αποκωδικοποίηση σήματος εισόδου

Αφού έχουμε υλοποιήσει την συνάρτηση codec0 η οποία περιέχει τις συναρτήσεις coder0 και decoder0, υπολογίζουμε το Y_{tot} και στη συνέχεια το $\hat{x}$ για σήμα εισόδου wavin. Το ανακατασκευασμένο αρχείο ήχου μπορεί να βρεθεί στον φάκελο της εργασίας με το όνομα x_hat.wav, ενώ παρακάτω παρουσιάζονται τα διαγράμματα του αρχικού και του ανακατασκευασμένου σήματος ήχου σε συνάρτηση με τον χρόνο:



2.5.1 Ακουστική Σύγκριση του wavin και $\hat{x}$

Ακούγοντας το αρχείο myfile.wav και το x_hat.wav, δεν παρατηρούμε διαφορές στον ήχο.

2.5.2 SNR

Για να υπολογίσουμε το SNR, αρχικά μετατοπίζουμε το x_hat κατα L-M προς τα δεξιά και το wavin κατα L-M προς τα αριστερά. Το SNR που πετυχαίνουμε ισούται με 0.98. Σε αυτό το σενάριο, το αποτέλεσμα υποδηλώνει ότι ο θόρυβος είναι εξίσου δυνατός με το αρχικό σήμα, καθιστώντας δύσκολη τη διάκριση του αρχικού σήματος από τον θόρυβο. Όπως αναφέρθηκε παραπάνω, τα δύο αρχεία ακούγονται πανομοιότυπα. Εξετάζοντας το παραπάνω φαινόμενο, είναι πιθανό ο θόρυβος που έχει φιλτραριστεί να μην είναι αντιληπτικά σημαντικός. Με άλλα λόγια, ο θόρυβος στο σήμα μπορεί να είναι μιας

συγκεκριμένης συχνότητας ή περιοχής συχνοτήτων που δεν είναι ιδιαίτερα αισθητή στο ανθρώπινο αυτί, και επομένως η αφαίρεσή του δεν επηρεάζει σημαντικά τη συνολική ποιότητα του ήχου. Σε αυτήν την περίπτωση, παρόλο που το SNR είναι χαμηλό, τα δύο αρχεία ήχου μπορεί να εξακολουθούν να ακούγονται πανομοιότυπα.

Κεφάλαιο 3

DCT

Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιείται το αρχείο DCT.py

3.1 Μετασχηματισμός DCT

Για να κατασκευάσουμε τους συντελεστές DCT χρησιμοποιούμε την συνάρτηση dct της βιβλιοθήκης fftpack της scipy. Για ένα frame υπολογίζουμε τους συντελεστές ανά σειρά. Στην έξοδο λαμβάνουμε ένα διάνυσμα μήκους M*N.

3.2 Αντίστροφος Μετασχηματισμός DCT

Για να αντιστρέψουμε διαδικασία, χρησιμοποιούμε την συνάρτηση idct της βιβλιοθήκης fftpack της scipy. Και πάλι υπολογίζουμε τις τιμές ανά σειρά και στην έξοδο λαμβάνουμε έναν πίνακα MxN που αντιστοιχεί στο αρχικό φραμε.

Κεφάλαιο 4

Υπολογισμός του κατωφλίου ακουστότητας

Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιούνται τα αρχεία:

- Section3-Test.py
- tonal_masking.py

4.1 Κατασκευή πίνακα Δκ

Αρχικοποιούμε τον πίνακα Δκ ως εξής:

Για συχνότητες κ ∈ [0,K_{max}]

1. Αν κ ∈ [1,281): Δκ[1] = 1
2. Αν κ ∈ [281,569): Δκ[1:12] = 1
3. Αν κ ∈ [569,1151): Δκ[1:26] = 1
4. Αλλιώς Δκ = 0

4.2 Εύρεση υποψήφιων maskers

Αρχικοποιούμε τον πίνακα με τους υποψήφιους maskers ως εξής:

Για κάθε διακριτή συχνότητα κ ∈ [0,K_{max}] πραγματοποιούμε δύο ελέγχους:

1. Αν η ισχύς στη συχνότητα κ είναι μεγαλύτερη από την ισχύ στην αμέσως επόμενη και προηγούμενη διακριτή συχνότητα
2. Αν η ισχύς στη συχνότητα κ είναι μεγαλύτερη κατά 7dB από την ισχύ σε μια γειτονιά συχνοτήτων η οποία καθορίζεται από τον πίνακα Δκ.

Αν και οι δύο έλεγχοι είναι αληθείς,τότε η συχνότητα κ εντάσσεται στον πίνακα με τους υποψήφιους maskers.

4.3 Μείωση υποψήφιων maskers

Αφού αρχικοποιήσουμε τον πίνακα με τους υποψήφιους maskers, στην συνέχεια τους μειώνουμε χρησιμοποιώντας δύο συνθήκες:

Για κάθε κ που ανήκει στον πίνακα με τους υποψήφιους maskers:

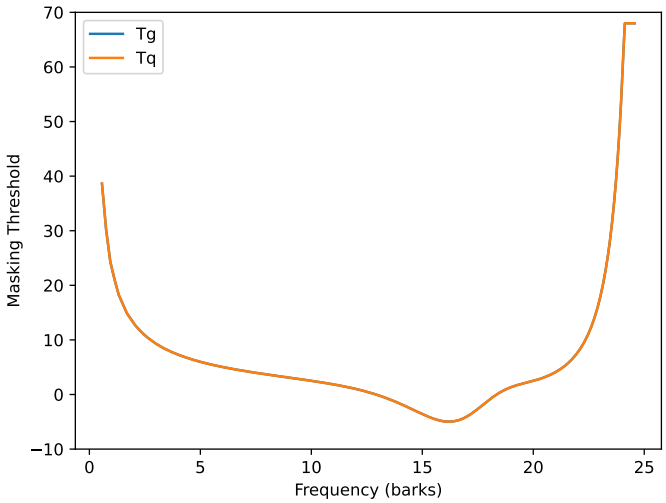
1. Ελέγχουμε αν η ισχύς στη συχνότητα κ είναι μεγαλύτερη από την ισχύ στη συχνότητα κ του κατωφλίου ακουστότητας στην ησυχία. Αν είναι, η συχνότητα κ παραμένει στον πίνακα.
2. Ελέγχουμε αν η απόσταση σε της συχνότητας κ σε barks από την αμέσως επόμενη συχνότητα που ανήκει στον πίνακα με τους maskers είναι μεγαλύτερη απο 0.5. Αν είναι, παραμένει στον πίνακα με τους maskers.

4.4 Εύρεση συνολικού κατωφλίου ακουστότητας

Έχοντας βρει τους υποψήφιους maskers, στη συνέχεια βρίσκουμε την συνεισφορά του κάθε masker στο κατώφλι ακουστότητας. Επιπλέον, υπολογίζουμε την spreading function για κάθε συχνότητα κ που ανήκει στους maskers. Τέλος, υπολογίζουμε το συνολικό κατώφλι ακουστότητας το οποίο προκύπτει ως υπέρθεση του κατωφλίου ακουστότητας στην ησυχία και της συνεισφοράς του κάθε μασκερ.

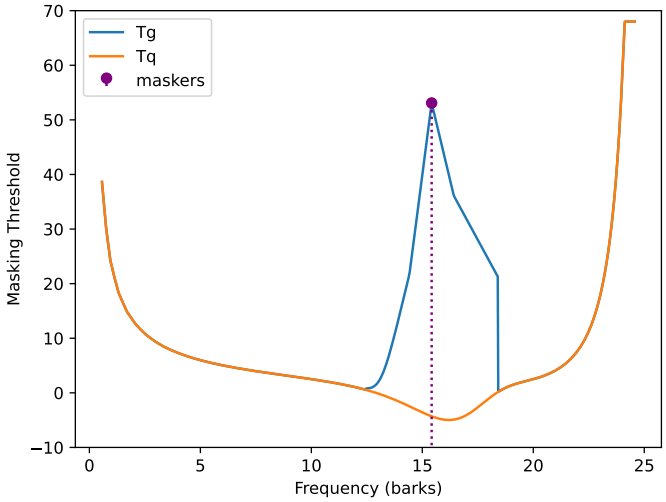
Παρακάτω παρουσιάζονται γραφήματα με το κατώφλι ακουστότητας πριν και μετά την συνεισφορά των maskers για κάποια frames:

- 1ο frame



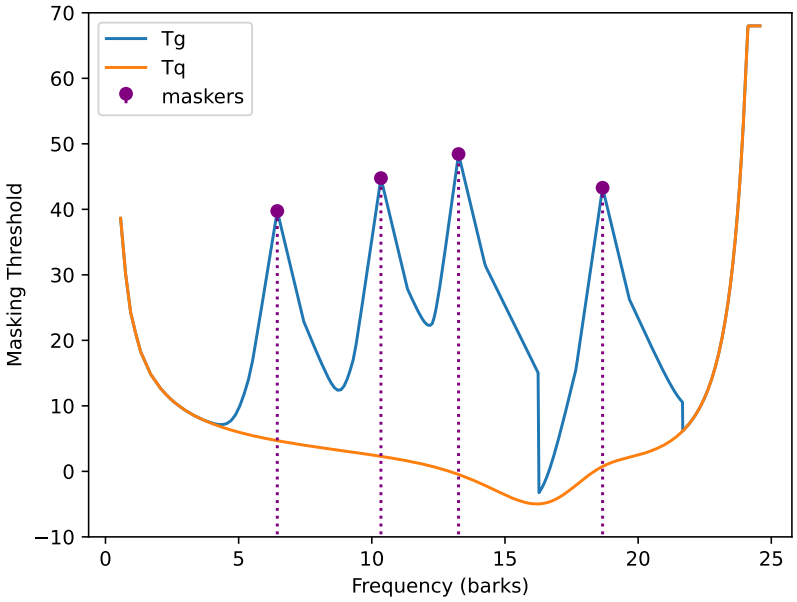
Παρατήρηση: Στο 1ο frame δεν έχουμε maskers, επομένως το κατώφλι ακουστότητας ταυτίζεται με το κατώφλι ακουστότητας στην ησυχία.

- 3ο frame

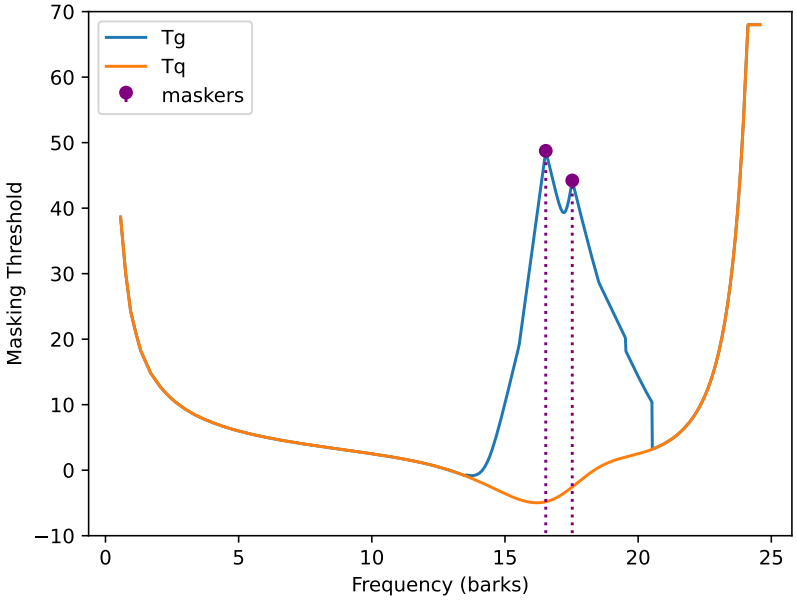


Σχήμα 4.1: 3ο frame

- 7ο frame

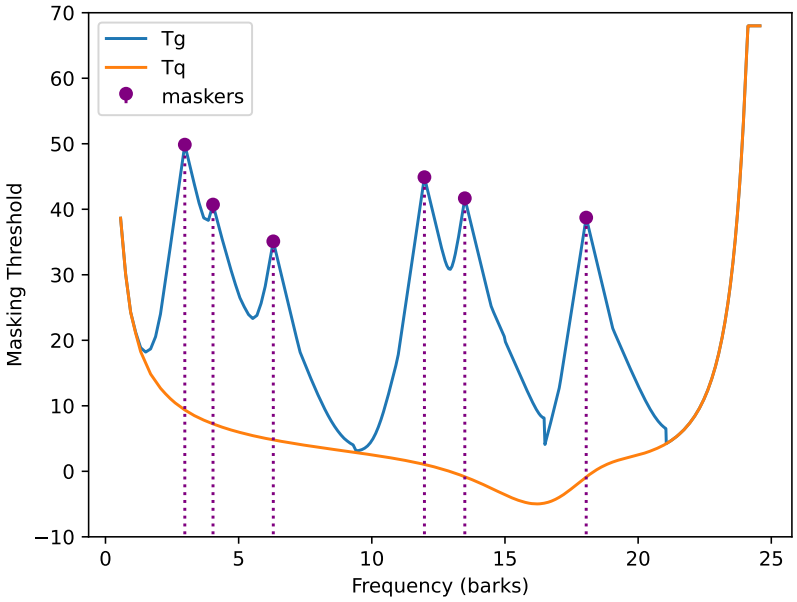


- 11ο frame



Σχήμα 4.2: 11ο frame

- 15ο frame



Σχήμα 4.3: 15ο frame

Κεφάλαιο 5

Κβαντισμός/Αποκβαντισμός Συντελεστών DCT

Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιούνται τα αρχεία:

- Section4-Test.py
- Quantization.py

5.1 Δημιουργία Συνάρτησης κβαντιστή

Η συνάρτηση κβάντισης δέχεται στην είσοδο ένα διάνυσμα x στο διάστημα [-1,1] και τον αριθμό των bit με τον οποίο θα γίνει ο κβαντισμός. Επιστρέφει στην έξοδο μία λίστα με τον αύξοντα αριθμό του συμβόλου στο οποίο αντιστοιχίζεται η κάθε τιμή του x.

5.2 Δημιουργία Συνάρτησης αποκβαντιστή

Η συνάρτηση αποκβαντισμού δέχεται στην είσοδο ένα διάνυσμα s με τους αριθμούς συμβόλων στο οποίο αντιστοιχίζεται το κάθε στοιχείο και τον αριθμό των bit με τον οποίο έχει γίνει ο κβαντισμός. Επιστρέφει στην έξοδο μία λίστα με τις τιμές στο διάστημα [-1,1] στις οποίες αντιστοιχίζεται το κάθε σύμβολο.

5.3 Εύρεση αριθμού bits για κάθε μπάντα

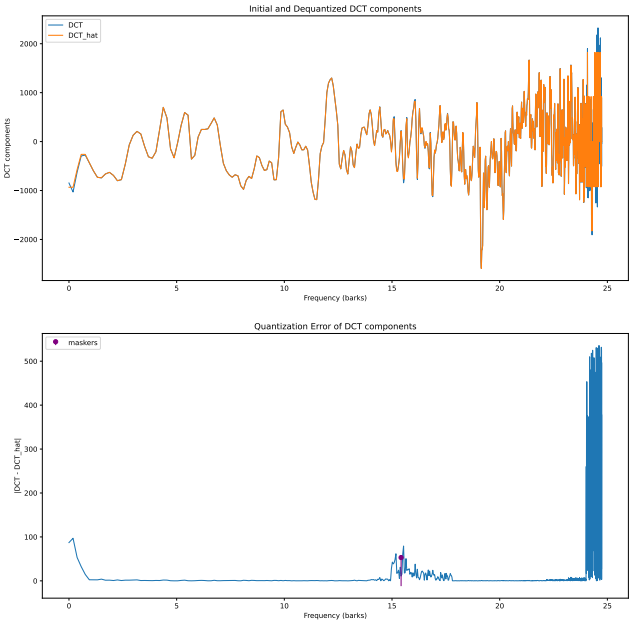
Η συνάρτηση αυτή εκμεταλλεύεται το ψυχοακουστικό φαινόμενο κατά τον κβαντισμό. Συγκεκριμένα, για είσοδο των συντελεστών DCT ενός frame:

1. Βρίσκει την μπάντα στην οποία ανήκει ο κάθε συντελεστής DCT
2. Για κάθε μπάντα:
   (α') Ξεκινώντας με #bits = 1 βρίσκει την αποκβαντισμένη τιμή των DCT συντελεστών της μπαντας
   (β') Βρίσκει την τιμή του κατωφλίου ακουστότητας για τις συχνότητες της μπάντας
   (γ') Υπολογίζει την τιμή DCT_hat
   (δ') Υπολογίζει το σφάλμα και την ισχύ κβαντισμού
   (ε') Ελέγχει αν η ισχύς του σφάλματος είναι μικρότερη από το κατώφλι ακουστότητα στις συχνότητες της μπάντας: - Αν είναι μικρότερη, ο κβαντισμός θα πραγματοποιηθεί με #bits = 1 - Αλλιώς αυξάνει το #bits κατά ένα και επαναλαμβάνει την διαδικασία
3. Επιστρέφει τον αριθμό των bits με τον οποίο θα κβαντιστεί η κάθε μπάντα, τα σύμβολα κβαντισμού και το scale factor κάθε μπάντας.

5.4 Κβαντισμός/ Αποκβαντισμός δοκιμαστικών frames

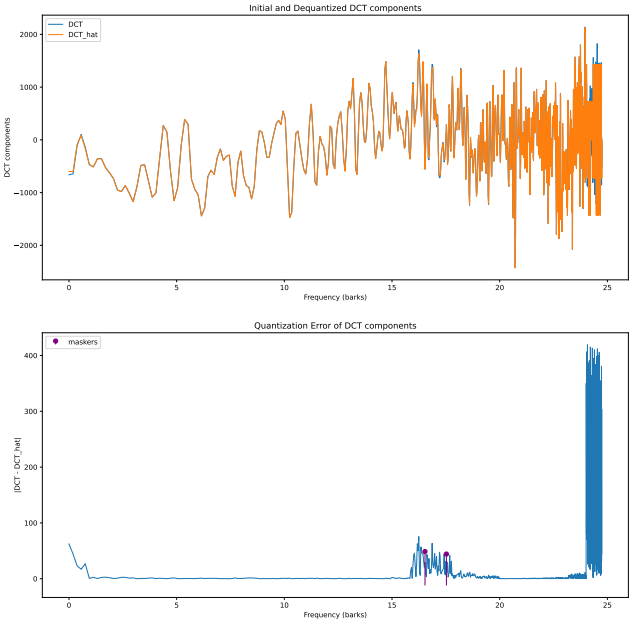
Δοκιμάσουμε να κβαντίσουμε και να αποκβαντίσουμε 3 δοκιμαστικά frames. Στη συνέχεια σχεδιάζουμε τους αρχικούς συντελεστές DCT και τους αποκβαντισμένους συντελεστές DCT και το σφάλμα κβαντισμού.

- 3ο frame



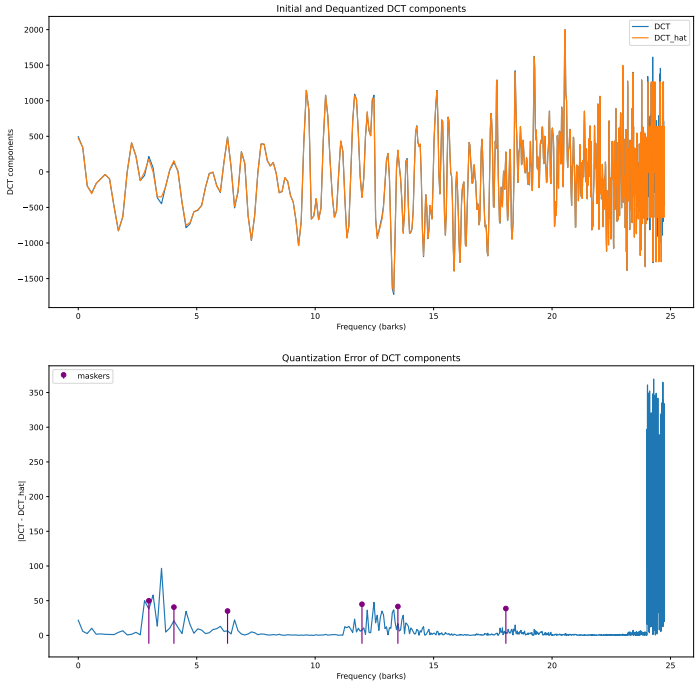
Παρατήρηση:Παρατηρώ ότι έχω έναν masker, επομένως στη συχνότητα που βρίσκεται ο masker το σφάλμα κβαντισμού αυξάνεται. Αυτό συμβαίνει διότι στο σημείο αυτό το κατώφλι ακουστότητας αυξάνεται όπως φαίνεται και στην εικόνα 4.1

- 11ο frame



Παρατήρηση:Παρατηρώ ότι έχω δύο maskers, επομένως στη συχνότητα που βρίσκονται οι maskers το σφάλμα κβαντισμού αυξάνεται. Αυτό συμβαίνει διότι στο σημείο αυτό το κατώφλι ακουστότητας αυξάνεται όπως φαίνεται και στην εικόνα 4.2

- 15ο frame



Παρατήρηση:Παρατηρώ ότι έχω δύο maskers, επομένως στη συχνότητα που βρίσκονται οι maskers το σφάλμα κβαντισμού αυξάνεται. Αυτό συμβαίνει διότι στο σημείο αυτό το κατώφλι ακουστότητας αυξάνεται όπως φαίνεται και στην εικόνα 4.3

Κεφάλαιο 6

Κωδικοποίηση μήκους διαδρομής

Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιείται το αρχείο:

- Section 5,6-Test
- rle.py

Η κωδικοποίηση μήκους διαδρομής αποτελείται από δύο συναρτήσεις, τον κωδικοποιητή και τον αποκωδικοποιητή. Είναι δύο απλές συναρτήσεις και συντάσσονται ως εξής:

6.1 Κωδικοποιητής

Η συμβολοσειρά που έρχεται από την έξοδο του κβαντιστή εισέρχεται σε ένα for-loop όπου ¨σκανάρουμε¨ προς τα μπροστά για να δούμε πόσες φορές επαναλαμβάνεται το παρόν σύμβολο. Έπειτα, αποθηκεύουμε σε ένα δισδιάστατο πίνακα με όνομα run_symbols το κάθε σύμβολο μαζί με τον αριθμό επαναλήψεών του.

6.2 Αποκωδικοποιητής

Ο αποκωδικοποιητής δέχεται ως είσοδο τον παραπάνω run_symbols πίνακα και στην έξοδό του επαναλαμβάνει τις τιμές της πρώτης στήλης για τον αριθμό των φορών της δεύτερης στήλης.

Κεφάλαιο 7

Κωδικοποίηση Huffman

Για την υλοποίηση της ενότητας αυτής σε κώδικα χρησιμοποιείται το αρχείο:

- Section 5,6-Test
- huffman.py

Η συνάρτηση κωδικοποίησης δέχεται τον πίνακα run_symbols ως όρισμα και κωδικοποιεί τα tuples (a,b) όπου a είναι το σύμβολο της εξόδου του αποκβαντιστή και b ο αριθμός επαναλήψεών του, ως συμβολοσειρές.

Χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης numpy ορίζουμε τον πίνακα symbol_occurrences διαστάσεων (k, 3), του οποίου οι πρώτες δύο στήλες συμβολίζουν τα a και b, με την τρίτη στήλη να συμβολίζει τον αριθμό εμφανίσεων του tuple (a,b) στην είσοδο.

Χρησιμοποιώντας τον πίνακα symbol_occurrences η διαδικασία υπολογισμού των δυαδικών συμβολοσειρών είναι κοινή και απαραίτητη για τις δύο συναρτήσεις κωδικοποίησης και αποκωδικοποίησης. Χρησιμοποιώντας τις συναρτήσεις των αλγορίθμων ουράς σωρών (heapq) της python δημιουργούμε ένα δυαδικό δέντρο όπου κάθε φύλλο του αναπαριστά ένα tuple (a,b). Στο παραπάνω δυαδικό δέντρο για κάθε μετάβαση σε αριστερό κλαδί προσθέτουμε ’0’ στην συμβολοσειρά, και για κάθε δεξί κλαδί προσθέτουμε

΄1΄. Με έναν αναδρομικό αλγόριθμο εξάγουμε το λεξικό codes που περιέχει την αντιστοίχηση tuple $\longleftrightarrow$ δυαδική συμβολοσειρά.

Σε αυτό το σημείο η διαδικασία κωδικοποίησης προσθέτει στο τέλος του bitstream frame_stream τη δυαδική συμβολοσειρά που ταιριάζει στο κάθε στοιχείο του πίνακα εισόδου. Αντιθέτως, η διαδικασία αποκωδικοποίησης διαβάζει έναν προς έναν τους χαρακτήρες της εισερχόμενης συμβολοσειράς frame_stream και όταν αντιστοιχίσει τους χαρακτήρες εισόδου σε έναν από τις δυαδικές συμβολοσειρές, προσθέτει το αντίστοιχο tuple (a,b) στην έξοδο.