

Συστήματα Πολυμέσων

Ρούσος Σταμάτης

20 Φεβρουαρίου 2026

1 Introduction

Σε αυτή την εργασία θα φτιάξουμε ένα σύστημα Encoding/Decoding ηχου κατά το πρότυπο AAC (Advanced Audio Coding). Σκοπός είναι να παρουμε ένα αρχικό σήμα ηχου, να το συμπίεσουμε - κωδικοποιήσουμε σε ένα σήμα μικροτερων "διαστάσεων" (λιγότερα bits) όσο περισσότερο γίνεται, και ταυτόχρονα να το αποκωδικοποιήσουμε - ανακατασκευάσουμε όσο πιο καλά γίνεται, δηλαδή να είναι όσο δυνατόν περισσότερο όμοιο με το αρχικό (ελάχιστη αντιληπτή διαφορά στην ακοή του σήματος). Με άλλα λόγια, αν x το αρχικό σήμα ηχου και \tilde{x} η εξοδος του αποκωδικοποιητη, θέλουμε:

$$x \approx \tilde{x}$$

Αρα επιδιώκουμε μέγιστη συμπίεση με trade off την ποιότητα.

2 SSC (Sequence Segmentation Control)

Η βαθμίδα SSC (Sequence Segmentation Control) κατηγοριοποιεί τα frames σε τύπους. Στο AAC χρησιμοποιούνται τέσσερις τύποι frame:

- **OLS** (Only Long Sequence): long frame, κατάλληλο για στασιμα/ομαλα τμήματα σήματος.
- **ESH** (Eight Short Sequence): 8 short subframes, κατάλληλο για μεταβατικά/κρουστικά τμήματα (attacks), ώστε να μειώνεται το pre-echo.

- **LSS** (Long Start Sequence): μεταβατικό frame που ομαλα μεταβαίνει από OLS σε ESH.
- **LPS** (Long Stop Sequence): μεταβατικό frame που ομαλα μεταβαίνει από ESH σε OLS.

Η βασική ιδέα είναι η ανίχνευση αποτομών αυξησεων ενεργειας (attacks) στο *επομενο* frame, ώστε το *τρεχον* frame να επιλεγεί ως κατάλληλο μεταβατικό (LSS) πριν από την εισοδο σε short sequence (ESH). Αντιστοίχα, όταν δεν υπάρχει attack στο *επομενο* frame, το σύστημα μπορεί να επιστρέψει σε long frames, χρησιμοποιώντας LPS για ομαλή μεταβαση.

Στην υλοποίηση, η ανίχνευση attacks γίνεται με high-pass φίλτρο και έλεγχο ενεργειας σε 8 διαδοχικά τμήματα των 128 samples. Υπολογίζεται ένα μέτρο *attack value* ως λόγος της ενεργειας ενός τμήματος προς τον μέσο όρο των προηγούμενων τμημάτων, και αν ξεπεραστούν συγκεκριμένα thresholds, το *επομενο* frame θεωρείται ESH. Στη συνέχεια, με βάση τον τύπο του προηγούμενου frame και την πληροφορία αν το *επομενο* θα είναι ESH, αποφασίζεται ο τύπος του *τρεχοντος* frame. Τέλος, επειδή υπάρχουν δύο κανάλια, υπολογίζεται τύπος ανά κανάλι και συνδυάζεται σε έναν τελικό τύπο με βάση έναν πίνακα συνδυασμού.

Στην υλοποίηση μου, το αρχείο SSC περιέχει τις εξής συναρτησεις/δομές:

- `_is_next_esh`: Ελέγχει αν το **επομενο** frame (ανά κανάλι) πρέπει να χαρακτηριστεί ως ESH, με βάση attack detection.
- `SSC`: Επιλέγει τον τύπο του **τρεχοντος** frame (OLS, LSS, ESH, LPS) με βάση το προηγούμενο frame type και την προβλεψη για το *επομενο*.
- `_COMBINE_TABLE`: Πίνακας που συνδυάζει τους τύπους των δύο κανάλιων σε έναν τελικό τύπο frame.

- `_HP_A`, `_HP_B`: Συντελεστές του high-pass IIR φίλτρου που χρησιμοποιείται για attack detection.

2.1 `_is_next_esh(next_frame_ch)`

Η συνάρτηση αυτή παίρνει ως ορίσμα το σήμα του επόμενου frame για ένα κανάλι:

$$next_frame_ch \in \mathbb{R}^{2048}$$

δηλαδή ένα μονοδιαστατο διάνυσμα 2048 δειγμάτων.

Λειτουργία: Σκοπός της είναι να ανιχνεύσει αν το επόμενο frame περιέχει attack, ώστε να χρειαστεί short sequence (ESH). Η διαδικασία είναι:

1) Για την ανιχνεύση αποτομών μεταβολών (attacks) στο επόμενο frame χρησιμοποιείται το υψιπερατό φίλτρο:

$$H(z) = \frac{0.7548 - 0.7548z^{-1}}{1 - 0.5095z^{-1}}$$

Στο πεδίο z ισχύει:

$$Y(z) = H(z)X(z)$$

δηλαδή:

$$Y(z)(1 - 0.5095z^{-1}) = X(z)(0.7548 - 0.7548z^{-1})$$

Περνώντας στο διακριτό πεδίο χρόνου, με την αντιστοιχισή $z^{-1}X(z) \leftrightarrow x[n-1]$ και $z^{-1}Y(z) \leftrightarrow y[n-1]$, προκύπτει η εξίσωση διαφοράς:

$$y[n] - 0.5095y[n-1] = 0.7548x[n] - 0.7548x[n-1]$$

η οποία γραφεται:

$$\boxed{y[n] = 0.5095y[n-1] + 0.7548(x[n] - x[n-1])}$$

Παρατηρούμε ότι ο όρος $x[n] - x[n - 1]$ αποτελεί διακριτή παραγωγο. Επομένως, όσο μεγαλύτερη είναι η διαφορά μεταξύ διαδοχικών δειγμάτων, τόσο μεγαλύτερο γίνεται το $y[n]$. Το φίλτρο επομένως τονίζει τις αποτομές μεταβολές του σήματος ("attacks").

Ο όρος $0.5095y[n - 1]$ εισαγει μνημη στο συστημα και δίνει ομαλότητα στην αποκρίσή του. Το φίλτρο είναι IIR πρώτης τάξης, καθώς περιλαμβάνει αναδρομικό όρο. Ο πόλος του συστήματος βρίσκεται στο:

$$z = 0.5095$$

και επειδή $|0.5095| < 1$, ο πόλος βρίσκεται εντός του μοναδιαίου κύκλου, άρα το φίλτρο είναι ευσταθές.

2) Χρήση του φίλτρου στο SSC

Το SSC λαμβάνει το επόμενο frame $i + 1$ και υπολογίζει για όλα τα δείγματα του την έξοδο του φίλτρου:

$$y[n] = 0.5095y[n - 1] + 0.7548x[n] - 0.7548x[n - 1]$$

Χωρίζει ολόκληρο το frame σε 8 διαδοχικά τμήματα μήκους 128 δειγμάτων. Για κάθε τμήμα $l = 0, 1, \dots, 7$ υπολογίζεται η ενεργεια:

$$s_l^2 = \sum_{n=128l}^{128(l+1)-1} y[n]^2$$

Για κάθε $l = 1, \dots, 7$ υπολογίζεται επιπλέον ο μέσος όρος των ενεργειών των προηγούμενων τμημάτων:

$$\bar{s}_{l-1}^2 = \frac{1}{l} \sum_{m=0}^{l-1} s_m^2$$

και ορίζεται το μέτρο attack:

$$d_{s_l}^2 = \frac{s_l^2}{\frac{1}{l} \sum_{m=0}^{l-1} s_m^2}$$

Όσο μεγαλύτερη είναι η ενέργεια s_l^2 σε κάποιο τμήμα (δηλαδή όσο περισσότερη μεταβολή υπάρχει στο σήμα ($y[n]$) την οποία εντοπίσε το φίλτραρισμα), και όσο μικρότερος είναι ο μέσος όρος των προηγούμενων ενεργειών (δηλαδή όσο μεγαλύτερη η συχία προηγείται), τόσο μεγαλύτερο γίνεται το "attack" $d_{s_l}^2$

Εάν για εστώ ένα τμήμα l ισχύει ταυτόχρονα:

$$s_l^2 > 10^{-3} \quad \text{και} \quad d_{s_l}^2 > 10$$

τότε το επόμενο frame κατηγοριοποιείται ως **EIGHT SHORT SEQUENCE**.

Για την κατηγοριοποίηση του τρεχόντος frame i μας αρκεί να γνωρίζουμε αν το επόμενο frame ($i + 1$) θα είναι EIGHT SHORT SEQUENCE ή όχι, όπως προκύπτει από τη λογική μεταβάσεων.

2.2 _COMBINE_TABLE

Επειδή ο έλεγχος attack γίνεται ανά κανάλι, μπορεί να προκύψει διαφορετικός τύπος τρεχόντος frame για το αριστερό και το δεξί κανάλι. Η AAC λογική απαιτεί τελικά **ενα** κοινό τύπο frame, ώστε να χρησιμοποιηθεί κοινή δομή παραθυρού/MDCT για τα δύο κανάλια.

Ο πίνακας _COMBINE_TABLE είναι ένα dictionary που υλοποιεί ακριβώς αυτόν τον συνδυασμό:

$$(\text{type}_{ch0}, \text{type}_{ch1}) \mapsto \text{frame_type}$$

Για παράδειγμα:

- (OLS, OLS) \rightarrow OLS
- (OLS, ESH) \rightarrow ESH

- (OLS, LSS) \rightarrow LSS
- (ESH, OLS) \rightarrow ESH
- (LPS, OLS) \rightarrow LPS
- (LSS, LPS) \rightarrow ESH

Γενικά, ο πίνακας τείνει να επιλέγει τον "πιο απαιτητικό" τύπο (π.χ. ESH) όταν τα κανάλια διαφωνούν, ώστε να μην χάθει attack σε κανένα κανάλι.

2.3 SSC(frame_T, next_frame_T, prev_frame_type)

Η συνάρτηση αυτή υλοποιεί τον Sequence Segmentation Control και επιλέγει τον τύπο του **τρέχοντος** frame.

Ορισματα:

- frame_T: το τρέχον frame στο πεδίο χρόνου, διαστάσεων (2048,2) για stereo.
- next_frame_T: το επόμενο frame στο πεδίο χρόνου, διαστάσεων (2048,2).
- prev_frame_type: string που δηλώνει τον τύπο του προηγούμενου frame (OLS, LSS, ESH, LPS).

Λειτουργία: Η συνάρτηση ακολουθεί τα βήματα:

1. Έλεγχος αν το επόμενο frame θα είναι ESH (ανά κανάλι):

$$next_esh_ch0 = _is_next_esh(next_frame_T[:, 0])$$

$$next_esh_ch1 = _is_next_esh(next_frame_T[:, 1])$$

2. Αποφαση τρέχοντος frame type ανά κανάλι: Στη συνέχεια υπάρχει μια εσωτερική βοηθητική λογική decide(prev, next_is_esh) που αποφασίζει τον τύπο του **τρέχοντος** frame για κάθε κανάλι, με βάση:

- τον τύπο του προηγούμενου frame

- το αν το **επομενο** frame προβλεπεται ESH

Η λογική μεταβασεων είναι:

- Αν **prev = OLS**:
 - αν **next_is_esh = True** \Rightarrow **current = LSS**
 - αλλιως \Rightarrow **current = OLS**
- Αν **prev = ESH**:
 - αν **next_is_esh = True** \Rightarrow **current = ESH**
 - αλλιως \Rightarrow **current = LPS (Long Stop Sequence)**
- Αν **prev = LSS**:

current = ESH

δηλαδή μετα απο LSS ακολουθει παντα ESH.

- Αν **prev = LPS**:

current = OLS

δηλαδή μετα απο LPS επιστρεφουμε σε long frames.

Με βάση τα παραπάνω, προκυπτουν:

$ch0_type = decide(prev_frame_type, next_esh_ch0)$

$ch1_type = decide(prev_frame_type, next_esh_ch1)$

3. **Συνδυασμος καναλιων**: Τελος, οι δυο τυποι συνδυαζονται με τον πινακα **_COMBINE_TABLE**:

$frame_type = _COMBINE_TABLE[(ch0_type, ch1_type)]$

Επιστρεφει: Επιστρεφει ενα string:

$frame_type \in \{OLS, LSS, ESH, LPS\}$

το οποιο είναι ο τελικος τυπος του τρεχοντος frame.

3 Filterbank

Η βαθμίδα αυτή χρησιμοποιεί το μετασχηματισμό Modified Discrete Cosine Transform (MDCT) ο οποίος μετασχηματίζει ένα σήμα από το πεδίο του χρόνου στο πεδίο της συχνότητας. Στο πεδίο της συχνότητας, οι συντελεστές έχουν μικρότερη μεταξύ τους συσχέτιση. Συγκεκριμένα, ο μετασχηματισμός εφαρμόζεται σε διαδοχικά frames-subframes και εκτελεί:

$$X_{i,k} = 2 \sum_{n=0}^{N-1} x_{i,n} \cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right), \quad 0 \leq k < \frac{N}{2}$$

όπου:

- $x_{i,n}$: τα δείγματα του ήχου στο i-οστό frame
- n : δεικτης δείγματος
- k : δεικτης συχνότητας
- $n_0 = \frac{\frac{N}{2}+1}{2}$

Ο μετασχηματισμός είναι πλήρως αντιστρεπτός, γι'αυτό και σε αυτό το επίπεδο δεν αναμενουμε μεγάλες απώλειες του αρχικού σήματος στο τελικό. Ο τύπος του αντιστροφικού μετασχηματισμού είναι γραμμένος στην εκφώνηση της εργασίας.

Στην υλοποίησή μου, το αρχείο filterbank περιέχει τις εξής συναρτήσεις:

- `mdct_matrix`: Υπολογίζει το κομμάτι

$$\cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right)$$

του MDCT.

- `mdct`: Υπολογίζει τον τελικό MDCT
- `imdct`: Κάνει τον αντιστροφή MDCT

- `kbd_window`: Υλοποιεί το παραθυρό Kasser Bessel Derived (KBD)
- `frame_window`: Βρίσκει το παραθυρό ενός συγκεκριμένου frame αναλόγα με τον τύπο του frame.
- `filter_bank`: Πολλαπλασιάζει τα δείγματα ενός frame με το παραθυρό, και εφαρμόζει mdct πάνω στο αποτέλεσμα.
- `i_filter_bank`: Πολλαπλασιάζει τους συντελεστές MDCT με το παραθυρό και εφαρμόζει αντιστροφή mdct πάνω στο αποτέλεσμα.

3.1 `mdct_matrix(N)`

Η συνάρτηση αυτή παίρνει ως ορίσμα το πλήθος δειγμάτων του frame.

Ορίζει τον δείκτη δειγματος n ως πίνακα της μορφής (1,N):

$$n = [0 \ 1 \ 2 \ \dots \ N - 1]$$

και το δείκτη συχνότητας k ως πίνακα της μορφής (N//2,1), όπου N//2 το ημίγειο της ευκλείδειας διαίρεσης του N με το 2,:

$$k = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ (N//2) - 1 \end{bmatrix}$$

Όπως είπαμε υπολογίζει το κομμάτι:

$$\cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right)$$

του MDCT. Μέσα στο σινημιτόνο, υπάρχουν οι πίνακες:

$$n + n_0 = [0 + n_0 \ 1 + n_0 \ 2 + n_0 \ \dots \ N - 1 + n_0]$$

$$k + 0.5 = \begin{bmatrix} 0.5 \\ 1.5 \\ 2.5 \\ (N//2) - 0.5 \end{bmatrix}$$

Η πράξη που γίνεται μέσα στο συνημιτονο μεταξύ αυτών των 2 πινάκων είναι το Outer Product, δηλαδή αν $N = n + n_0$ και $K = k + 0.5$ τότε:

$$A = N * K = \begin{bmatrix} N_{11}K_{11} & N_{12}K_{11} & \dots & N_{1N}K_{11} \\ N_{11}K_{21} & N_{12}K_{21} & \dots & N_{1N}K_{21} \\ \vdots & \vdots & \ddots & \vdots \\ N_{11}K_{(N//2)1} & N_{12}K_{(N//2)1} & \dots & N_{1N}K_{(N//2)1} \end{bmatrix} =$$

$$\begin{bmatrix} (0 + n_0)0.5 & (1 + n_0)0.5 & \dots & (N - 1 + n_0)0.5 \\ (0 + n_0)1.5 & (1 + n_0)1.5 & \dots & (N - 1 + n_0)1.5 \\ (0 + n_0)2.5 & (1 + n_0)2.5 & \dots & (N - 1 + n_0)2.5 \\ \vdots & \vdots & \ddots & \vdots \\ (0 + n_0)(\frac{N}{2} - 0.5) & (1 + n_0)(\frac{N}{2} - 0.5) & \dots & (N - 1 + n_0)(\frac{N}{2} - 0.5) \end{bmatrix}$$

Επειδή ο $N * K$ είναι πολ/μος διαστάσεων $(1, N) * (N/2, 1)$, η Python κάνει broadcasting εσωτερικά, το οποίο ουσιαστικά είναι σαν να υπολογίζει $(N/2, 1) * (1, N) = (N/2, N)$.

Ο τελικός πίνακας C προκύπτει από το συνημιτονο του πολ/μου του πίνακα A με τη σταθερά $\frac{2\pi}{N}$:

$$C = \cos\left(\frac{2\pi}{N} * A\right)$$

Κάθε γραμμή του C , $C[k]$, είναι μια συνημιτονική βάση για το δεικτική συχνότητας k .

3.2 mdct(x,N)

Η συνάρτηση αυτή παίρνει ως ορίσματα το frame x και το πλήθος N των δειγμάτων του.

Υπολογίζει τον τελικο μετασχηματισμο MDCT ως $MDCT = 2 * C * x$. Ετσι προκυπτει ενα column-vector X με στοιχεια:

$$X[k] = \sum_{n=0}^{N-1} C[k, n] * x[n], \quad 0 \leq k < \frac{N}{2}$$

διαστασεων $(N/2, N) \times (N,) = (N/2,)$, οπου καθε στοιχειο του ειναι ενας συντελεστης MDCT. Καθε συντελεστης, ειναι ενα εσωτερικο γινομενο μιας συνημιτονικης βασης με τα δειγματα του frame. Επομενως, μετρα την ομοιοτητα των δειγματων με αυτη τη βαση. Επειδη

$$f_k = (k + \frac{1}{2}) \cdot \frac{F_s}{N}$$

οπου:

- F_s : Sampling rate
- N : Μηκος παραθυρου MDCT

ενας συντελεστης MDCT θα μπορουσε να ερμηνευτει ως "ποσο απο τη συχνοτητα f_k υπαρχει σε αυτο το frame".

3.3 kbd_window(N,a)

Η συναρτηση αυτη υλοποιει το παραθυρο KBD η συναρτηση του οποιου δινεται στην εκφωνηση. Η υλοποιηση εγινε με χρηση της συναρτησης kaiser της Python.

3.4 frame_window(frame_type)

Η συναρτηση παιρνει ως ορισμα τον τυπο του frame και υπολογιζει το παραθυρο που αντιστοιχει σε αυτον τον τυπο με βαση τη θεωρια στις σελιδες 127-132 του w2203tfa αρχειου. Στην δικη μας εργασία θα χρησιμοποιουμε παντα σταθερο σχημα παραθυρου. Εγω υλοποιησα και χρησιμοποιησα μονο το KBD window. Επομενως, θα ειμαστε μονιμως στην περιπτωση $window_shape = 1$ (δηλαδη KBD). Δε θα κανουμε εναλλαγες σχηματος οπως γραφει στο αρχειο. Με βαση τα παραπανω:

Για frame type **OLS** (Only Long Sequence) θα έχουμε:

$$W(n) = \begin{cases} W_{\text{LEFT},2048}(n), & 0 \leq n < 1024 \\ W_{\text{KBD_RIGHT},2048}(n), & 1024 \leq n < 2048 \end{cases}$$

$$= W_{\text{KBD},2048}(n), \quad 0 \leq n < 2048$$

Για **LSS** (Long Start Sequence):

$$W(n) = \begin{cases} W_{\text{KBD},2048}(n), & 0 \leq n < 1024 \\ 1.0, & 1024 \leq n < 1472 \\ W_{\text{KBD},256}(n - 1472 + 128), & 1472 \leq n < 1600 \\ 0.0, & 1600 \leq n < 2048 \end{cases}$$

Για **LPS** (Long Stop Sequence):

$$W(n) = \begin{cases} 0.0, & 0 \leq n < 448 \\ W_{\text{KBD},256}(n - 448), & 448 \leq n < 576 \\ 1.0, & 576 \leq n < 1024 \\ W_{\text{KBD},2048}(n), & 1024 \leq n < 2048 \end{cases}$$

Για **ESH** (Eight Short Sequence):

$$W_0(n) = W_1(n) = \dots = W_7(n) = W_{\text{KBD},256}(n), \quad 0 \leq n < 256$$

Στην υλοποίηση μου, για κάθε περίπτωση frame, με τη συνάρτηση concatenate της numpy, ενώσα τους κλάδους της $W(n)$ διαδοχικά, φτιαχνοντας την τελική συνάρτηση παραθυρού W για αυτό το frame ως ένα διάνυσμα 2048 στοιχείων.

3.5 filter_bank(frame_T, frame_type)

Η συνάρτηση αυτή παίρνει ως ορίσματα τα δειγμάτα του frame και τον τυπο του frame.

Καλεί τη συνάρτηση frame_window που επιστρέφει το παραθυρο του frame. Αν ο τυπος του frame δεν είναι ESH, τότε απλά πολ/ζει τα δειγμάτα του frame για κάθε καναλι με το παραθυρο, και τα περνάει στην συνάρτηση MDCT οπου επιστρέφει τους 1024 συντελεστες (μιας και τα frames είναι παντα μεγεθους 2048 δειγμάτων). Αν το frame είναι ESH, τότε επιλεγονται μονο τα κεντρικα 1152 δειγμάτα (απο 448 μεχρι 1600, τα άλλα αγνοουνται) τα οποια σπαμε σε 8 επικαλυπτομενα κατα 50% τμηματα-subframes των 256 δειγμάτων ($256 + 7 \cdot 128 = 1152$). Καθε subframe πολ/ζεται με το αντιστοιχο παραθυρο του (στην περιπτωση μας ολα είναι ιδια), και το αποτελεσμα μπαινει στην mdct η οποια επιστρέφει τους 128 συντελεστες MDCT.

Σημειωνω οτι παρελειψα να εξηγησω τις i-συναρτησεις καθως κανουν απλα την αντιστροφη διαδικασια.

4 AAC_Coder_1

Το αρχειο aac_coder_1 υλοποιει τον κωδικοποιητη και αποκωδικοποιητη του Level 1. Σε αυτο το επιπεδο η συμπιεση είναι χωρις κβαντισμο και χωρις εντροπικη κωδικοποιηση, οποτε δεν περιμενουμε μεγαλες απωλειες ποιότητας.

Η βασικη ροη είναι:

- Χωριζουμε το σημα σε επικαλυπτομενα frames μηκους 2048 δειγμάτων με επικάλυψη 50% (hop 1024).
- Για καθε frame αποφασιζουμε frame_type με τη βαθμιδα SSC.
- Εφαρμοζουμε filterbank (MDCT με καταλληλα παραθυρα) και αποθηκευουμε τους συντελεστες.

- Στην αποκωδικοποίηση εφαρμόζουμε iFilterbank (IMDCT με τα ίδια παραθύρα) και ανακατασκευάζουμε το σήμα με overlap-add.

Στην υλοποίηση μου, το αρχείο περιέχει τις εξής συναρτήσεις:

- `prepare_signal`: Προσθετεί padding στο σήμα και φροντίζει να διαιρείται ακριβώς σε frames
- `aac_coder_1`: Κωδικοποιητής Level 1 (SSC + filterbank)
- `i_aac_coder_1`: Αποκωδικοποιητής Level 1 (iFilterbank + overlap-add + αφαίρεση padding)

4.1 `prepare_signal(x)`

Η συνάρτηση `prepare_signal` παίρνει ως ορίσμα το σήμα x στο πεδίο του χρόνου, σε μορφή πίνακα $(N, 2)$ (δύο κανάλια), και επιστρέφει το ίδιο σήμα με κατάλληλο padding ώστε να μπορεί να τεμαχιστεί σε frames μήκους 2048 με hop 1024.

1) **Padding στην αρχή και στο τέλος (1024 δειγμάτα)**

Στην αρχή εφαρμόζεται:

$$x \leftarrow pad(x, (HOP, HOP))$$

δηλαδή προστιθενται $HOP = 1024$ μηδενικά δειγμάτα στην αρχή και 1024 στο τέλος.

Ο λόγος είναι ότι στο AAC χρησιμοποιούμε επικάλυψη 50% (hop = 1024). Το πρώτο frame μήκους 2048 χρειάζεται να έχει και "αριστερό" περιεχόμενο-δειγμάτα. Επειδή όμως στην αρχή του σήματος δεν υπάρχουν προηγούμενα δειγμάτα, προσθετούμε μηδενικά ώστε:

- το πρώτο frame να ξεκινάει σωστά,
- να μπορεί να γίνει σωστή παραθυρώση και IMDCT χωρίς να χάνεται πληροφορία στην αρχή.

Για τον ίδιο λόγο κανούμε padding στο τέλος. Αρα το νέο μήκος μετά το 1ο padding είναι:

$$n = n_{initial} + 2048$$

2) Padding στο τέλος για ακριβή διαιρέση σε frames

Το πρώτο frame ξεκινά στη θέση 0 και έχει μήκος 2048, αρα καλύπτει τα δείγματα:

$$[0, 2048)$$

δηλαδή από 0 έως και 2047.

Για να υπάρξει δεύτερο frame, πρέπει να υπάρχει χώρος για άλλο παραθυρό μήκους 2048. Αν το συνολικό μήκος του σήματος είναι n , τότε το τελευταίο frame μπορεί να ξεκινήσει το πολύ μέχρι τη θέση:

$$n - 2048$$

γιατί αν ξεκινήσει πιο μετά, το παραθυρό μήκους 2048 δεν θα χωράει ολοκλήρο μέσα στο σήμα.

Οι θέσεις εκκίνησης των frames είναι:

$$i \cdot 1024, \quad i = 0, 1, 2, \dots$$

και θέλουμε το i -οστό frame να χωράει ολοκλήρο, αρα:

$$i \cdot 1024 + 2048 \leq n$$

ισοδυναμα:

$$i \cdot 1024 \leq n - 2048$$

Αρα:

$$i \leq \frac{n - 2048}{1024}$$

Ο μέγιστος ακέραιος i που ικανοποιεί την ανισότητα είναι:

$$\left\lfloor \frac{n - 2048}{1024} \right\rfloor$$

Επομενως ο αριθμος των frames ειναι:

$$num_frames = \left\lfloor \frac{n - 2048}{1024} \right\rfloor + 1$$

Το +1 υπαρχει γιατι μετραμε και το frame που ξεκινα στη θεση 0.

Οριζουμε:

$$rem = (n - 2048) \bmod 1024$$

Πρεπει:

$$rem = 0 \tag{1}$$

Αν $rem > 0$, τοτε υπαρχει ακεραιος q ωστε:

$$n - 2048 = q \cdot 1024 + rem$$

με:

$$0 < rem < 1024$$

Padding: Προσθετουμε $1024 - rem$ μηδενικα δειγματα στο τελος, αρα το νεο μηκος γινεται:

$$n' = n + (1024 - rem)$$

Θελουμε να υπολογισουμε το νεο υπολοιπο:

$$(n' - 2048) \bmod 1024$$

Αντικαθιστουμε:

$$n' - 2048 = (n - 2048) + (1024 - rem)$$

Και επειδη:

$$n - 2048 = q \cdot 1024 + rem$$

προκυπτει:

$$n' - 2048 = q \cdot 1024 + rem + 1024 - rem$$

Τα rem απλοποιουνται:

$$n' - 2048 = q \cdot 1024 + 1024 = (q + 1) \cdot 1024$$

Επομενως:

$$((q + 1) \cdot 1024) \bmod 1024 = 0$$

γιατι ειναι ακριβες πολλαπλασιο του 1024. Για $n = n_{initial} + 2048$, η συνθηκη (1) γινεται:

$$rem = n_{initial} \bmod 1024 = 0$$

Αν $rem > 0$ ακολουθεται η διαδικασια που ειπαμε παραπανω. Ετσι, με το διπλο αυτο padding, το αρχικο σημα τωρα χωριζεται σε ακεραιο αριθμο frames, ισο με:

$$num_frames = \lfloor \frac{n_{initial}}{1024} \rfloor + 1$$

Στην υλοποιηση εκανα $num_frames = \lceil \frac{n_{initial}}{1024} \rceil + 1$ ωστε να δημιουργειται παντα ενα επιπλεον frame (με μηδενικα δειγματα οπου δεν υπαρχει σημα). Ετσι, γλιτωνω τον ελεγχο του υπολοιπου rem.

4.2 aac_coder_1(filename_in)

Η συναρτηση aac_coder_1 διαβαζει ενα wav αρχαιο και παραγει μια λιστα aac_seq απο dictionaries, ενα για καθε frame.

Ορισματα

- filename_in: το wav αρχαιο εισοδου

Επιστρεφει

- aac_seq: λιστα frames, καθε στοιχειο περιεχει:
 - frame_type: OLS, LSS, ESH, LPS
 - chl["frame_F"]: συντελεστες MDCT αριστερου καναλιου
 - chr["frame_F"]: συντελεστες MDCT δεξιου καναλιου

Διαδικασια

1. Διαβαζει το σημα και το κανει 2D.
2. Καλει prepare_signal για padding.

3. Υπολογίζει το πλήθος frames:

$$\text{num_frames} = \frac{n - 2048}{1024} + 1$$

4. Για κάθε frame i παίρνει:

$$\text{frame_t} = x[i \cdot 1024 : i \cdot 1024 + 2048]$$

5. Υπολογίζει και το **next frame** για τη βαθμίδα SSC. Για το τελευταίο frame βάζει μηδενικό frame.

6. Καλεί:

$$\text{frame_type} = \text{SSC}(\text{frame_t}, \text{next_frame_t}, \text{prev_frame_type})$$

7. Καλεί:

$$\text{frame_f} = \text{filter_bank}(\text{frame_t}, \text{frame_type})$$

8. Αποθηκεύει τους συντελεστες:

- Αν ESH: τους αναδιατάσσει σε (128, 8) ανα κανάλι
- Αλλιώς: κρατά (1024, 1) ανα κανάλι

4.3 **i_aac_coder_1(aac_seq, filename_out)**

Η συνάρτηση `i_aac_coder_1` παίρνει τη λίστα frames `aac_seq` και ανακατασκευάζει το σήμα στο πεδίο του χρόνου.

Ορισματα

- `aac_seq`: λίστα frames (όπως παραχθηκε απο τον κωδικοποιητη)
- `filename_out`: wav εξοδου

Επιστρεφει

- `y`: το αποκωδικοποιημενο σήμα

1) Ανακατασκευή $frame_F$ απο το αποθηκευμενο $format$

Για καθε $frame$ δημιουργειται πινακας:

$$frame_f \in \mathbb{R}^{1024 \times 2}$$

- Αν $frame_type = ESH$, οι συντελεστες $(128, 8)$ καθε καναλιου γινονται flatten με column-major ($order="F"$) για να επανελθουν στη σειρα subframes.
- Αλλιως, οι συντελεστες $(1024, 1)$ γινονται reshape σε $(1024,)$.

2) Επιστροφη στο πεδιο χρονου

Καλει:

$$frame_t = i_filter_bank(frame_f, frame_type)$$

και παιρνει $frame$ μηκους 2048 δειγματων.

3) Overlap-add reconstruction

Εδω γινεται το πιο σημαντικό κομματι της ανακατασκευης. Επειδη τα frames ειναι επικαλυπτομενα (50%), καθε $frame_t$ πρεπει να προστεθει στην εξοδο στη σωστη θεση:

$$start = i \cdot HOP = i \cdot 1024$$

$$y[start : start + 2048] \leftarrow y[start : start + 2048] + frame_t$$

Δηλαδη, το δευτερο μισο του προηγουμενου $frame$ και το πρωτο μισο του επομενου $frame$ καταληγουν να αθροιζονται στο ιδιο σημειο του χρονου.

Αυτο το overlap-add δουλευει σωστα επειδη τα παραθυρα του MDCT/IMDCT εχουν σχεδιαστει ωστε να ικανοποιουν συνθηκη τελειας ανακατασκευης (Princen-Bradley). Ετσι, οταν προστεθουν τα επικαλυπτομενα τμηματα, οι παραθυρωμενες συνιστωσες συμπληρωνουν η μια την αλλη και δεν δημιουργουνται ``σπασιματα`` στα ορια των frames.

4) Αφαιρεση padding

Επειδη στην αρχη ειχαμε προσθεσει 1024 δειγματα στην αρχη και 1024 στο τελος, στο τελος αφαιρουμε:

$$y \leftarrow y[1024 : -1024]$$

Ετσι επιστρεφουμε στο αρχικο μηκος του σηματος (χωρις τα τεχνητα μηδενικα που χρησιμοποιηθηκαν για το framing).

5) Αποθηκευση

Τελος, γινεται εγγραφη του wav:

```
sf.write(filename_out, y, 48000)
```

5 AAC_Coder_2 - AAC_Coder_3

Στο Level 1 ο κωδικοποιητης περιλαμβανει μονο τις βαθμιδες:

$$SSC \rightarrow FilterBank$$

και ο αποκωδικοποιητης:

$$iFilterBank \rightarrow Overlap-Add$$

Στα επομενα επιπεδα προστιθενται επιπλεον βαθμιδες και πληροφορια ανα frame.

Τι προστεθηκε στο aac_coder_2 σε σχεση με το Level 1

Στο Level 2 προστεθηκε η βαθμιδα TNS (Temporal Noise Shaping) μετα το FilterBank.

Το pipeline γινεται:

$$SSC \rightarrow FilterBank \rightarrow TNS$$

Συγκεκριμενα προστεθηκαν:

- Κληση της συναρτησης `tns(frame_F, frame_type)` ανα καναλι.
- Αποθηκευση των `tns_coeffs` για καθε καναλι στο dictionary του frame.
- Στον αποκωδικοποιητη προστεθηκε η βαθμιδα `i_tns` πριν το `i_filter_bank`.

Τι προστεθηκε στο `aac_coder_3` σε σχεση με το Level 1

Στο Level 3 υλοποιειται ολο το πληρες pipeline του AAC:

$SSC \rightarrow FilterBank \rightarrow TNS \rightarrow Psycho \rightarrow Quantizer \rightarrow Huffman$

Σε σχεση με το Level 1, προστεθηκαν:

- **TNS** (οπως στο Level 2).
- **Psychoacoustic Model:**
 - Υπολογισμος SMR ανα μπαντα.
 - Χρηση των δυο προηγουμενων frames για προβλεψη.
- **Quantizer:**
 - Υπολογισμος scalefactor gains.
 - Παραγωγη κβαντισμενων συμβολων S .
 - Παραγωγη G και sfc .
- **Huffman encoding:**
 - Κωδικοποιηση των S .
 - Κωδικοποιηση των sfc με συγκεκριμενο codebook.
 - Αποθηκευση bitstreams και codebook.

Στον αποκωδικοποιητη προστεθηκαν οι αντιστροφες βαθμιδες:

$Huffman \rightarrow iQuantizer \rightarrow iTNS \rightarrow iFilterBank \rightarrow Overlap-Add$

6 Temporal Noise Shaping (TNS)

Στην κωδικοποιηση με MDCT, το σημα χωριζεται σε frames σταθερου μηκους και καθε frame επεξεργαζεται ξεχωριστα. Το ψυχοακουστικο μοντελο θεωρει οτι μεσα σε καθε frame το σημα δεν αλλαζει πολυ (ειναι περιπου στασιμο).

Στην πράξη όμως αυτό δεν ισχύει πάντα. Μπορεί μέσα στο ίδιο παραθυρό να εμφανιστεί μια αποτομή αλλαγή, όπως ένας δυνατός παλμός (π.χ. ένα χτύπημα τυμπανού).

Ο μετασχηματισμός MDCT "απλώνει" την ενέργεια αυτής της αποτομής αλλαγής σε όλο το frame. Όταν μετά γίνει κβάντιση, το σφάλμα κβάντισης επίσης απλώνεται χρονικά σε όλο το frame.

Αυτό μπορεί να δημιουργήσει το φαινόμενο του **pre-echo**: δηλαδή να ακουστεί θορύβος λίγο πριν από τον δυνατό παλμό. Αυτό είναι ενοχλητικό, γιατί το ανθρώπινο αυτί καλύπτει (masking) καλύτερα τον θορύβο μετά από έναν δυνατό ήχο, αλλά πολύ λιγότερο πριν από αυτόν. Αρα, αν εμφανιστεί θορύβος πριν από τον παλμό, γίνεται πιο ευκόλα αντιληπτός.

Το **Temporal Noise Shaping (TNS)** χρησιμοποιείται για να μειώσει αυτό το πρόβλημα. Η ιδέα είναι ότι πριν την κβάντιση εφαρμόζεται μια μορφή γραμμικής προβλεψής στους συντελεστές MDCT. Έτσι αφαιρείται η μεταξύ τους συσχέτιση και το σήμα γίνεται πιο "ομοιομορφό" (πιο κοντά σε λευκό θορύβο), ώστε να κβαντιστεί πιο αποτελεσματικά.

Στον αποκωδικοποιητή εφαρμόζεται το αντιστρόφο φίλτρο για να επανέλθει η αρχική φασματική μορφή. Όμως το σφάλμα κβάντισης έχει πλέον κατανεμηθεί διαφορετικά στον χρόνο: δεν συγκεντρώνεται πριν από τον παλμό, αλλά μεταφέρεται σε χρονικές περιοχές όπου καλύπτεται καλύτερα από το ίδιο το σήμα.

Με λίγα λόγια, το TNS δεν αλλάζει μονίμως το σήμα. Αλλάζει τον τρόπο που καταναίνεται χρονικά ο θορύβος κβάντισης, ώστε να γίνεται λιγότερο αντιληπτός.

Στην υλοποίησή μου, το αρχείο `tns` περιέχει τις εξής συναρτήσεις:

- `load_bands`: Φορτώνει από το `TableB219.mat` τα band tables για long και short frames.
- `get_band_edges`: Επιστρέφει τα όρια (w_{low} , w_{high}) των bands

αναλογα με τον τυπο frame.

- `tns`: Κεντρικη συναρτηση TNS (encoder). Χειριζεται long frames και ESH και επιστρεφει τους φιλτραρισμενους συντελεστες MDCT και τους κβαντισμενους συντελεστες TNS.
- `tns_subframe`: Υλοποιει το TNS για ενα frame/subframe (κανονικοποιηση, LPC, κβαντισμος, ελεγχος ευσταθειας, φιλτραρισμα).
- `is_stable`: Ελεγχει την ευσταθεια των συντελεστων προβλεψης μεσω των ριζων του αντιστοιχου πολυωνυμου.
- `apply_fir`: Εφαρμοζει το FIR φιλτρο TNS στους συντελεστες MDCT.
- `i_tns`: Αντιστροφη TNS (decoder). Εφαρμοζει το αντιστροφο φιλτρο χρησιμοποιωντας τους αποθηκευμενους συντελεστες.
- `apply_iir`: Εφαρμοζει το IIR φιλτρο (αντιστροφο του FIR) για την ανακατασκευη των αρχικων συντελεστων.

6.1 `load_bands()`

Η συναρτηση αυτη φορτωνει τους πινακες bands απο το αρχειο TableB219.mat. Συγκεκριμενα:

- B219a: 69 bands για long frames
- B219b: 42 bands για short frames

Οι πινακες αυτοι περιγραφουν υποζωνες συχνοτητας και χρησιμοποιουνται για να υπολογιστει η ενεργεια ανα band κατα την κανονικοποιηση.

6.2 `get_band_edges(frame_type)`

Η συναρτηση παιρνει ως ορισμα τον τυπο frame και επιστρεφει δυο πινακες (w_{low} , w_{high}) που περιεχουν τα κατω και ανω ορια (σε δεικτες MDCT) για καθε band.

Για **OLS/LSS/LPS** χρησιμοποιει τα long bands, ενώ για **ESH** χρησιμοποιει τα short bands.

6.3 tns(frame_F_in, frame_type)

Η συνάρτηση αυτή είναι η κεντρική συνάρτηση του TNS στον encoder. Παιρνει ως εισοδο τους συντελεστες MDCT πριν το TNS και τον τυπο frame.

Για long frames (**OLS/LSS/LPS**), η εισοδος είναι διαστάσεων (1024, 1). Η συνάρτηση μετατρέπει το frame σε μονοδιαστατη ακολουθία, καλεί τη _tns_subframe και επιστρέφει:

- frame_F_out: τους φιλτραρισμένους συντελεστες MDCT (ίδιες διαστάσεις)
- tns_coeffs: τους κβαντισμένους συντελεστες προβλεψής τάξης $p = 4$ (διαστάσεων (4, 1))

Για **ESH**, οι συντελεστες MDCT αντιμετωπίζονται ως 8 subframes. Η εισοδος θεωρείται πίνακας (128, 8) (128 συντελεστες ανά subframe). Η συνάρτηση εφαρμόζει TNS ανεξαρτήτως σε κάθε subframe καλώντας 8 φορές τη _tns_subframe και επιστρέφει:

- frame_F_out διαστάσεων (128, 8)
- tns_coeffs διαστάσεων (4, 8)

6.4 tns_subframe(X, frame_type)

Η συνάρτηση αυτή υλοποιεί όλη τη διαδικασία TNS για ένα frame (long) ή για ένα subframe (short). Τα βήματα που εκτελεί είναι:

- **Βήμα 1: Κανονικοποίηση ανά band (normalization).**

Υπολογίζεται η ενεργεια κάθε band:

$$P[b] = \sum_{k=w_{low}[b]}^{w_{high}[b]} X[k]^2$$

και στη συνεχεια υπολογιζεται ο συντελεστης κανονικοποιησης $S_w[k]$:

$$S_w[k] = \sqrt{P[b]}, \quad k \in [w_{low}[b], w_{high}[b]]$$

(αν $P[b] = 0$ τοτε τιθεται $S_w[k] = 1$). Μετα απο αυτο, εφαρμοζεται εξομαλυνση ωστε το $S_w[k]$ να μην αλλαζει αποτομα μεταξυ bands. Τελος, οι κανονικοποιημενοι συντελεστες ειναι:

$$X_w[k] = \frac{X[k]}{S_w[k] + \epsilon}$$

με ϵ μικρη σταθερα για αποφυγη διαιρεσης με το μηδεν.

• **Βημα 2: Υπολογισμος LPC συντελεστων (ταξη $p = 4$).**

Υπολογιζεται η αυτοσυσχετιση της X_w και κρατιουνται οι τιμες $r[0], r[1], \dots, r[p]$. Στη συνεχεια σχηματιζεται Toeplitz πινακας R και λυνεται το συστημα:

$$R a = r_{1:p}$$

ωστε να προκυψουν οι συντελεστες a_1, \dots, a_p . Για αριθμητικη σταθεροτητα προστεθηκε μικρη regularization διαγωνια.

• **Βημα 3: Κβαντισμος των συντελεστων προβλεψης.**

Οι συντελεστες κβαντιζονται με βημα 0.1:

$$a_q = \text{round} \left(\frac{a}{0.1} \right) \cdot 0.1$$

και περιοριζονται στο διαστημα $[-0.8, 0.7]$. Οι κβαντισμενοι συντελεστες ειναι αυτοι που αποθηκευονται και μεταδιδονται στον decoder.

• **Βημα 4: Ελεγχος ευσταθειας.**

Γινεται ελεγχος ευσταθειας στους κβαντισμενους συντελεστες. Αν το φιλτρο δεν ειναι ευσταθες, τοτε οι συντελεστες μηδενιζονται.

- **Βήμα 5: Εφαρμογή FIR φίλτρου στο αρχικό X .**

Με τους κβαντισμένους συντελεστές a_q εφαρμόζεται FIR φίλτρο:

$$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \dots - a_p z^{-p}$$

και το αποτέλεσμα Y αποτελεί τους συντελεστές MDCT μετά το TNS.

Η συνάρτηση επιστρέφει τους φιλτραρισμένους συντελεστές και τους κβαντισμένους συντελεστές προβλεψής.

6.5 `is_stable(a)`

Η συνάρτηση αυτή ελέγχει αν το προβλεπτικό φίλτρο είναι ευσταθές. Στην υλοποίηση μου σχηματίζεται ένα πολυώνυμο από τους συντελεστές a και υπολογίζονται οι ρίζες του. Αν όλες οι ρίζες βρίσκονται εντός του μοναδιακού δίσκου $|z| < 1$, τότε το φίλτρο θεωρείται ευσταθές.

Αν για οποιονδήποτε λόγο αποτυχεί ο υπολογισμός ριζών, η συνάρτηση επιστρέφει False.

6.6 `apply_fir(X, a)`

Η συνάρτηση αυτή εφαρμόζει το FIR φίλτρο του TNS στους συντελεστές MDCT.

Το φίλτρο ορίζεται στο πεδίο του μετασχηματισμού Z ως:

$$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \dots - a_p z^{-p}$$

Στην υλοποίηση μας χρησιμοποιούμε φίλτρο τάξης $p = 4$, άρα:

$$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - a_3 z^{-3} - a_4 z^{-4}$$

Από τον ορισμό του γραμμικού χρονικά αμεταβλήτου φίλτρου ισχύει:

$$H(z) = \frac{Y(z)}{X(z)}$$

Αρα:

$$Y(z) = H(z)X(z)$$

και αντικαθιστώντας το $H_{\text{TNS}}(z)$:

$$Y(z) = X(z) \left(1 - \sum_{\ell=1}^p a_{\ell} z^{-\ell} \right)$$

Αναπτύσσοντας:

$$Y(z) = X(z) - \sum_{\ell=1}^p a_{\ell} z^{-\ell} X(z)$$

Παιρνοντας αντιστροφο μετασχηματισμο Z , και γνωρίζοντας οτι

$$z^{-\ell} X(z) \longleftrightarrow X(k - \ell),$$

προκύπτει η σχέση στο διακριτο πεδιο:

$$Y(k) = X(k) - \sum_{\ell=1}^p a_{\ell} X(k - \ell)$$

Για $p = 4$:

$$Y(k) = X(k) - a_1 X(k-1) - a_2 X(k-2) - a_3 X(k-3) - a_4 X(k-4)$$

Παρατηρούμε οτι η εξοδος του φίλτρου Y είναι το σφάλμα $e(k) = X[k] - \text{προβλεψη}$, δηλαδή το TNS χρησιμοποιεί prediction error filter. Το φίλτρο εφαρμόζεται σε κάθε συντελεστή MDCT $X(k)$ της ακολουθίας, και έτσι προκύπτει μια νέα ακολουθία φίλτραρισμένων συντελεστών $Y(k)$ ίδιου μήκους. Αυτή η ακολουθία είναι που τελικά μπαίνει στον κβαντιστή.

Όπως φαίνεται από τη μορφή του στο πεδίο Z , το φίλτρο της εργασίας μας περιέχει παραγοντες μέχρι z^{-4} . Ο εκθετης στο $z^{-\ell}$ δηλώνει καθυστέρηση ℓ δειγμάτων. Επομένως το φίλτρο έχει μνήμη $p = 4$, δηλαδή κάθε δείγμα εξόδου $Y(k)$ εξαρτάται μόνο από τα $X(k)$, $X(k-1)$, $X(k-2)$, $X(k-3)$ και $X(k-4)$.

Για $k < 4$, η σχέση εφαρμόζεται χρησιμοποιώντας μόνο τους όρους που ορίζονται (δηλαδή για $k = 0$ δεν υπάρχει καθυστέρηση, για $k = 1$ χρησιμοποιείται μόνο ο όρος $X(k - 1)$ κ.ο.κ.), ώστε το φίλτρο να είναι αιτιατό.

6.7 `i_tns(frame_F_in, frame_type, tns_coeffs)`

Η συνάρτηση αυτή υλοποιεί την αντιστροφή διαδικασία στον decoder. Παιρνει ως εισοδο τους συντελεστες MDCT μετά το TNS και τους αποθηκευμένους κβαντισμένους συντελεστες `tns_coeffs`.

Για long frames, εφαρμόζεται το αντιστροφο φίλτρο σε όλη την ακολουθία των 1024 συντελεστών και επιστρέφεται πίνακας (1024, 1).

Για **ESH**, η εισοδος περιεχει 8 subframes, και η συνάρτηση εφαρμόζει το αντιστροφο φίλτρο ανεξαρτήτα σε κάθε subframe χρησιμοποιώντας τους αντιστοιχους συντελεστες (4, 8).

6.8 `apply_iir(Y, a)`

Η συνάρτηση αυτή εφαρμόζει το αντιστροφο IIR φίλτρο του TNS. Επειδη στον encoder εφαρμοστηκε:

$$Y[k] = X[k] - \sum_{\ell=1}^p a[\ell] X[k - \ell],$$

στον decoder ανακατασκευάζουμε το X αναδρομικά ως:

$$X[k] = Y[k] + \sum_{\ell=1}^p a[\ell] X[k - \ell].$$

Η αναδρομηση ξεκινά από $k = 0$ και για κάθε k αθροίζονται μόνο όροι με $\ell \leq k$. Το αποτέλεσμα είναι οι συντελεστες MDCT πριν την εφαρμογή TNS.

Συμπεραινοντας, το TNS υπολογίζει τους συντελεστες a_l έτσι ώστε:

$$X[k] \approx \sum_{l=1}^p a_l X[k - l]$$

Ψαχνει δηλαδή τη γραμμική εξάρτηση του κάθε συντελεστή από τους προηγούμενους, αφαιρεί αυτό το κομμάτι (προβλεψη) από το αρχικό σήμα, και δίνει στον κβαντιστή το υπόλοιπο (error). Έτσι οι συντελεστές γίνονται λιγότερο συσχετισμένοι - πιο "λευκοί", και περιορίζεται το φαινόμενο pre-echo που είπαμε, δηλαδή ο θορύβος να ακουγεται πριν την αποτομή αλλαγή στο σήμα, οπότε ο θορύβος γίνεται λιγότερο αντιληπτός.

7 Psychoacoustic Model

Η βαθμίδα αυτή υλοποιεί το Ψυχοακουστικό Μοντέλο του AAC. Σκοπός της είναι να υπολογίσει για κάθε ζώνη συχνότητας το κατώφλι ακουστότητας του θορύβου κβαντισμού, δηλαδή ποση παραμορφωση μπορεί να προστεθεί χωρίς να γίνει αντιληπτή από το ανθρώπινο αυτί.

Η βασική ιδέα είναι ότι το ανθρώπινο σύστημα ακοής δεν έχει ίδια ευαισθησία σε όλες τις συχνότητες και ότι ισχυρά φασματικά συστατικά μπορούν να καλύψουν ασθενέστερα (masking). Για τον λόγο αυτό το μοντέλο υπολογίζει ένα κατώφλι ενεργείας ανά μπάντα συχνότητας και τελικά το Signal to Mask Ratio (SMR), το οποίο καθοδηγεί τον κβαντιστή.

Η διαδικασία βασίζεται σε:

- Μετασχηματισμός FFT κάθε frame ή subframe
- Υπολογισμός προβλεψιμότητας από τα δύο προηγούμενα παραθύρα
- Υπολογισμός ενεργείας ανά μπάντα
- Εφαρμογή spreading function
- Υπολογισμός tonality index
- Υπολογισμός κατώφλιου masking
- Υπολογισμός SMR

Στην υλοποίηση μου, το αρχείο `psycho` περιέχει τις εξής συναρτήσεις:

- `load_bands`: Φορτώνει τους πίνακες μπαντών από το αρχείο `TableB219.mat`
- `spreading_fun`: Υπολογίζει την spreading function μεταξύ δυο μπαντών
- `precompute_spreading`: Προϋπολογίζει τον πίνακα spreading για ταχύτητα
- `psycho`: Κυρία συνάρτηση του ψυχοακουστικού μοντέλου
- `psycho_subframe`: Υλοποιεί όλο τον αλγόριθμο για ένα frame ή subframe

7.1 `load_bands()`

Η συνάρτηση αυτή φορτώνει από το αρχείο `TableB219.mat` τους πίνακες `B219a` και `B219b` που αντιστοιχούν στις μπάντες του ψυχοακουστικού μοντέλου για:

- Long frames (69 μπάντες)
- Short frames (42 μπάντες)

Κάθε γραμμή του πίνακα περιέχει πληροφορία για μια μπάντα:

- `wlow`: κάτω όριο FFT index
- `whigh`: ανώ όριο FFT index
- `bval`: κεντρική συχνότητα
- `qsthr`: κατωφλί σε ησυχία

7.2 spreading_fun(i,j,bval)

Η spreading function υπολογίζει τον συντελεστή εξασθενισης ο οποίος δειχνει ποσο η ενεργεια ενος ηχου σε μια μπαντα συχνοτητων i μασκαρει (καλυπτει) τον ηχο σε αλλες μπαντες συχνοτητων j .

Με αλλα λογια, δειχνει ποσο ανεβαινει το κατωφλι ακουστοτητας της μπαντας j λογω της ενεργειας της μπαντας i , δηλαδη ποσο δυσκολα θα ακουστει κατι στη μπαντα j λογω του ηχου στη μπαντα i .

Ο υπολογισμος γινεται με βαση τις εξισωσεις της εκφωνησης:

$$tmpx = \begin{cases} 3(bval_j - bval_i), & i \geq j \\ 1.5(bval_j - bval_i), & i < j \end{cases}$$

Ακολουθουν οι υπολογισμοι tmpz και tmpy και τελικα:

$$x = 10^{\frac{tmpz+tmpy}{10}}$$

αν $tmpy < -100$ τοτε $x=0$.

Το αποτελεσμα εκφραζει τον συντελεστη εξασθενισης.

7.3 precompute_spreading()

Η συναρτηση precompute_spreading υπολογίζει τον πινακα spreading διαστασεων $n_bands \times n_bands$ (69×69 για long frame και 42×42 για short frame), οπου το στοιχειο $[i][j]$ ισουται με την τιμη της spreading function για τη μπαντα i ως προς τη μπαντα j , δηλαδη:

$$SPREADING[i][j] = spreading_fun(i, j).$$

Για να υπολογισουμε τη μασκα της μπαντας j , δηλαδη το κατωφλι ακουστοτητας της, υπολογιζουμε τον γραμμικο συνδυασμο:

$$MASK(j) = \sum_i E(i) \cdot spreading(i, j),$$

οπου $E(i)$ ειναι η ενεργεια της μπαντας i .

Όσο μεγαλύτερη είναι η τιμή $MASK(j)$, τόσο μεγαλύτερο είναι το κατώφλι ακουστότητας της μπαντας j . Αυτό σημαίνει ότι ο κωδικοποιητής μπορεί να επιτρέψει μεγαλύτερο θορυβό κβαντισής στη μπαντα j , αφού δε θα ακουστεί τόσο ευκόλα.

Αρα, μπορεί να κάνει πιο χοντρή κβαντισή (λιγότερα bits), και πετυχαίνει καλύτερη συμπίεση χωρίς αντιληπτή υποβαθμισμό του σηματος.

Το τελικό κατώφλι ακουστότητας της μπαντας j είναι, σε απλοποιημένη μορφή, ουσιαστικά:

$$Threshold(j) = \max (MaskingThreshold(j), qsthr(j)),$$

όπου $qsthr(j)$ είναι το φυσικό (absolute) κατώφλι ακουστότητας του ανθρώπινου αυτιού για τη συγκεκριμένη μπαντα συχνοτήτων j .

7.4 **psycho(frame_T, frame_type, frame_T_prev_1, frame_T_prev_2)**

Η συνάρτηση αυτή είναι η κύρια συνάρτηση του ψυχοακουστικού μοντέλου.

Παίρνει ως ορίσματα:

- `frame_T`: το τρέχον frame στο πεδίο του χρόνου
- `frame_type`: τον τύπο του frame (OLS, LSS, LPS, ESH)
- `frame_T_prev_1`: το προηγούμενο frame
- `frame_T_prev_2`: το προ-προηγούμενο frame

Επιστρέφει:

- **SMR**: πίνακα διαστάσεων 69x1 για long frames ή 42x8 για ESH

Για ESH frames:

- Επιλέγονται τα κεντρικά 1152 δείγματα (448-1600)

- Χωρίζονται σε 8 επικαλυπτομενα κατα 50% subframes των 256 δειγματος
- Καλειται η `_psycho_subframe` για καθε subframe

Για long frames:

- Καλειται μια φορα η `_psycho_subframe` για ολοκληρο το frame

7.5 `_psycho_subframe(s, s_prev1, s_prev2, frame_type)`

Η συναρτηση αυτη υλοποιει ολοκληρο τον αλγοριθμο για ενα frame ή subframe.

1. Παραθυρωση και FFT

Καθε σημα πολλαπλασιαζεται με παραθυρο Hann:

$$w(n) = 0.5 - 0.5 \cos \left(\frac{\pi(n + 0.5)}{N} \right)$$

Στη συνεχεια υπολογιζεται ο FFT και κρατουνται τα $N/2$ θετικα φασματικα δειγματα:

$$r(w) = |FFT|, \quad f(w) = \angle FFT$$

2. Προβλεψη απο προηγουμενα παραθυρα

$$r_{pred} = 2r_{prev1} - r_{prev2}$$

$$f_{pred} = 2f_{prev1} - f_{prev2}$$

3. Υπολογισμος προβλεψιμοτητας

$$c(w) = \frac{\sqrt{(r \cos f - r_{pred} \cos f_{pred})^2 + (r \sin f - r_{pred} \sin f_{pred})^2}}{r + |r_{pred}|}$$

4. Ενεργεια ανα μπαντα

$$e(b) = \sum r(w)^2$$

$$c(b) = \sum c(w)r(w)^2$$

5. Εφαρμογή spreading function

$$ecb(b) = \sum_{bb=0}^{N_B-1} e(bb) \text{ spreading_function}(bb, b)$$

$$ct(b) = \sum_{bb=0}^{N_B-1} c(bb) \text{ spreading_function}(bb, b)$$

6. Tonality index

$$cb(b) = \frac{ct(b)}{ecb(b)}$$

$$en(b) = \frac{ecb(b)}{\sum_{bb=0}^{N_B-1} \text{spreading_function}(bb, b)}$$

$$t_b(b) = -0.299 - 0.43 \ln(cb(b))$$

7. Υπολογισμος SNR

$$SNR(b) = t_b(b) \cdot 18 + (1 - t_b(b)) \cdot 6$$

$$bc(b) = 10^{\frac{-SNR(b)}{10}}$$

$$nb(b) = en(b) \cdot bc(b)$$

$$npart = \max(nb(b), \hat{q}_{thr}(b))$$

8. Signal to Mask Ratio

$$SMR(b) = \frac{e(b)}{npart(b)}$$

Το SMR αποτελεί την εξοδο της συναρτησης και χρησιμοποιει-
ται στη βαθμιδα κβαντισης.

8 Quantizer

Η βαθμίδα αυτή υλοποιεί την κβαντισή και αποκβαντισή των συντελεστών MDCT. Σκοπός της είναι η συμπίεση του σήματος με έλεγχο του ακουστικού σφάλματος, χρησιμοποιώντας την πληροφορία του ψυχοακουστικού μοντέλου μέσω του δείκτη SMR (Signal to Mask Ratio).

Η βασική ιδέα είναι ότι για κάθε κριτική μπάντα συχνότητας επιτρέπεται διαφορετικό ποσοστό σφάλματος κβαντισής. Ο κβαντιστής ρυθμίζει τους συντελεστές scale factor $a(b)$ ώστε η ισχύς του σφάλματος κβαντισής $P_e(b)$ να μην υπερβαίνει το κατώφλι $T(b) = \frac{P(b)}{SMR(b)}$.

Στην υλοποίησή μου, το αρχείο quantizer περιέχει τις εξής συναρτήσεις:

- `get_last_quantizer_stats`: Επιστρέφει τις τελευταίες debug μετρικές του κβαντιστή.
- `load_bands`: Φορτώνει τους πίνακες μπαντών από το αρχείο `TableB219.mat`.
- `quantize_mdct`: Υλοποιεί τον μη ομοιομορφο κβαντιστή MDCT.
- `dequantize_mdct`: Υλοποιεί την αποκβαντισή των συντελεστών MDCT.
- `quantize_subframe`: Υλοποιεί ολό τον αλγόριθμο κβαντισής για ένα frame ή subframe.
- `dequantize_subframe`: Κάνει την αντιστροφή διαδικασία για ένα frame ή subframe.
- `aac_quantizer`: Υλοποιεί ολό το pipeline από είσοδο wav μέχρι decoded σήμα για κάθε frame.
- `i_aac_quantizer`: Αντιστροφή του `aac_quantizer`.

8.1 get_last_quantizer_stats()

Η συνάρτηση αυτή επιστρέφει ένα αντιγραφο του global dictionary LAST_DEBUG_STATS. Περιλαμβάνει πληροφορίες όπως τις ελάχιστες και μεγιστες τιμες των $a(b)$, τον αριθμο επαναληψεων, τον τροπο τερματισμου (Converged λόγω Pe ή Δα ή max_iter), καθώς και μετρικες για SMR , $P(b)$, $T(b)$ και $P_e(b)$.

8.2 load_bands()

Η συνάρτηση φορτώνει απο το αρχείο TableB219.mat τους πίνακες B219a και B219b που αντιστοιχουν στις 69 μπαντες long frame και στις 42 μπαντες short frame αντιστοιχα.

Για καθε μπαντα οριζονται τα ορια δεικτων $w_{low}(b)$ και $w_{high}(b)$, τα οποια καθοριζουν ποιοι συντελεστες MDCT ανηκουν σε καθε κριτικη μπαντα.

8.3 quantize_mdct(X, alpha)

Η συνάρτηση αυτή υλοποιει τον μη ομοιομορφο κβαντιστη των συντελεστων MDCT.

Παιρνει ως ορισματα:

- X : συντελεστες MDCT ενός frame ή subframe (στην πραγματικότητα οι συντελεστες Y που είναι οι φιλτραρισμενοι MDCT, δηλαδή αυτους που δινει στην εξοδο το TNS)
- α : scale factor της μπαντας

Υπολογιζεται αρχικα ο παραγοντας κλιμακωσης:

$$scale = 2^{-\alpha/4}$$

Στη συνεχεια οι συντελεστες $X(k)$ κβαντιζονται ως:

$$S(k) = sign(X(k)) \cdot \left[|X(k) \cdot 2^{-\alpha/4}|^{0.75} + 0.4054 \right]$$

Και η αποκβαντιση είναι:

$$\hat{X}(k) = sign(S(k)) |S(k)|^{4/3} 2^{\alpha/4}$$

Το αποτέλεσμα είναι οι ακεραίοι κβαντισμένοι συντελεστές $S(k)$. Ο κβαντιστής λέγεται μη ομοιομορφος λόγω της παρουσίας του εκθετη 0.75. Όσο μεγαλώνει το a , η κβαντιση γίνεται πιο χοντρή, δηλαδή μεγαλώνει το βήμα κβαντισης. Αυτό φαίνεται ως εξής:

$$\begin{aligned}\hat{X}(k) &= \text{sign}(S(k)) |S(k)|^{4/3} 2^{a/4} \\ \Delta x &\approx \left| \frac{d\hat{X}}{dS} \right| = \left| \frac{d}{dS} (\text{sign}(S) |S|^{4/3} 2^{a/4}) \right| = \frac{4}{3} |S|^{1/3} 2^{a/4} \\ S(k) &= \text{sign}(X(k)) \lfloor |X(k) 2^{-a/4}|^{3/4} + 0.4054 \rfloor \approx \text{sign}(X) |X 2^{-a/4}|^{3/4} \\ |S| &\approx (|X| 2^{-a/4})^{3/4} = |X|^{3/4} 2^{-3a/16} \\ |S|^{1/3} &\approx (|X|^{3/4} 2^{-3a/16})^{1/3} = |X|^{1/4} 2^{-a/16} \\ \Delta x &\approx \frac{4}{3} |S|^{1/3} 2^{a/4} \approx \frac{4}{3} (|X|^{1/4} 2^{-a/16}) 2^{a/4} = \frac{4}{3} |X|^{1/4} 2^{a/4-a/16} \\ \Delta x &\approx \frac{4}{3} |X|^{1/4} 2^{3a/16}\end{aligned}$$

Επειδή $2^{3a/16}$ είναι γνησίως αυξουσα συνάρτηση του a , προκύπτει ότι Δx αυξάνεται όταν αυξάνεται το a . Όσο λοιπόν αυξάνεται το a , το $S(k)$ μικραίνει ολο και περισσότερο. Αμα παρουν αρκετα μεγαλες τιμες, τοτε $S(k) \rightarrow 0$, δηλαδή τα $X(k)$ σπρωχονται-κβαντιζονται προς τη σταθμη 0 του κβαντιστη. Τοτε εχουμε το μεγιστο σφαλμα κβαντισμου, δηλαδή:

$$\begin{aligned}a \rightarrow \infty &\Rightarrow S(k) \rightarrow 0 \Rightarrow \hat{X}(k) \rightarrow 0 \Rightarrow \\ &\Rightarrow P_e = \sum (X(k) - \hat{X}(k))^2 \rightarrow X(k)^2 = P_{e,max}\end{aligned}$$

Η συνάρτηση επιστρεφει τα κβαντισμενα $S(k)$.

8.4 dequantize_mdct(S, alpha)

Η συνάρτηση αυτή υλοποιεί την αποκβαντιση των συντελεστών MDCT.

Εφαρμόζεται ο τύπος:

$$\hat{X}(k) = \text{sign}(S(k)) \cdot |S(k)|^{4/3} \cdot 2^{a/4}$$

Επιστρεφεται η αποκβαντισμενη τιμη $\hat{X}(k)$.

8.5 quantize_subframe(X, frame_type, SMR)

Η συνάρτηση αυτή υλοποιεί τον κύριο αλγόριθμο κβαντισμού για ένα frame ή subframe.

Παίρνει ως ορίσματα:

- X : συντελεστές MDCT (στην πραγματικότητα οι συντελεστές Y που είναι οι φιλτραρισμένοι MDCT, δηλαδή αυτούς που δίνει στην έξοδο το TNS)
- $frame_type$: τύπος frame (OLS, LSS, LPS, ESH)
- SMR : Signal to Mask Ratio ανά μπαντα, υπολογισμένο από το psychoacoustic model

Επιστρέφει:

- S : κβαντισμένους συντελεστές
- sfc : scale factors σε μορφή DPCM
- G : global gain

1. Υπολογισμός ενεργειας και κατωφλίου

Για κάθε μπαντα υπολογίζεται:

$$P(b) = \sum_{k=w_{low}(b)}^{w_{high}(b)} X(k)^2$$

και

$$T(b) = \frac{P(b)}{SMR(b)}$$

2. Αρχική εκτίμηση των scale factors

Ορίζεται μια αρχική τιμή:

$$\hat{a} = \frac{16}{3} \log_2 \left(\frac{(\max |X|)^{0.75}}{8191} \right)$$

και τίθεται $a(b) = \hat{a}$ για όλες τις μπαντες.

3. Επαναληπτική αύξηση των a

Σε κάθε επαναληψη:

- γίνεται κβαντιση και αποκβαντιση
- υπολογίζεται η ισχυς σφαλματος

$$P_e(b) = \sum_{k=w_{low}(b)}^{w_{high}(b)} (X(k) - \hat{X}(k))^2$$

Αν $P_e(b) < T(b)$ γίνεται αυξηση του $a(b)$ κατα 1, αν ικανοποιε-
ται ο περιορισμος:

$$|a(b+1) - a(b)| \leq 60$$

για τις 2 γειτονικες μπαντες $b-1$, $b+1$ της μπαντας b . Η διαδικα-
σια τερματιζει οταν δεν μπορεί να αυξηθει κανενα $a(b)$ ή οταν
ο αλγοριθμος φτασει στο `max_iter`.

4. DPCM κωδικοποιηση

Μετα την τελικη τιμη των $a(b)$:

$$G = a(0)$$

$$sfc(b) = a(b) - a(b-1)$$

8.6 dequantize_subframe(S, sfc, G, frame_type)

Η συναρτηση αυτη κανει την αντιστροφη διαδικασια.

Αρχικα ανακατασκευαζει τα $a(b)$ απο το DPCM:

$$a(0) = G, \quad a(b) = a(b-1) + sfc(b)$$

Στη συνεχεια εφαρμοζει την `dequantize_mdct` ανα μπαντα και
επιστρεφει τους αποκβαντισμενους συντελεστες.

8.7 aac_quantizer(frame_F, frame_type, SMR)

Η συναρτηση αυτη διαχειριζεται την κβαντιση για ολοκληρο
frame.

Για ESH frames:

- χωριζει σε 8 subframes

- καλει την `quantize_subframe` για καθε subframe
- ενωνει τα αποτελεσματα

Για long frames:

- καλει απλα μια φορα την `quantize_subframe`

8.8 i_aac_quantizer(S, sfc, G, frame_type)

Η συναρτηση αυτη υλοποιει την αποκαντιση σε επιπεδο frame.

Για ESH frames αποκβαντιζει καθε subframe ξεχωριστα, ενω για long frames καλει μια φορα την `dequantize_subframe` και επιστρεφει τους αποκβαντισμενους συντελεστες.

9 AAC_Coder_3

Το αρχειο αυτο υλοποιει τον κωδικοποιητη και τον αποκωδικοποιητη του AAC Level 3. Ουσιαστικα υλοποιει ολες τις βαθμιδες frame-by-frame.

$$\begin{aligned}
 \textbf{Encoder: } x[n] \text{ (WAV, time)} &\xrightarrow[\text{frame blocking}]{\text{SSC}} \{x_i[n]\}_i, \text{ frame_type}_i \\
 \{x_i[n]\}_i &\xrightarrow[\text{MDCT + window}]{\text{FilterBank}} X_i(k) \xrightarrow[\text{prediction filter}]{\text{TNS}} Y_i(k) \\
 Y_i(k) &\xrightarrow[\text{masking model}]{\text{Psycho}} SMR_i(b) \xrightarrow{\text{Quantizer}} (S_i(k), sfc_i(b), G_i) \\
 (S_i(k), sfc_i(b), G_i) &\xrightarrow[\text{entropy coding}]{\text{Huffman}} \text{bitstream}_i \xrightarrow[\text{store}]{\text{.mat}} \text{AAC_seq}_3 \\
 \textbf{Decoder: } \text{AAC_seq}_3 &\xrightarrow[\text{load}]{\text{.mat}} \text{bitstream}_i \xrightarrow[\text{entropy decoding}]{\text{iHuffman}} (S_i(k), sfc_i(b), G_i) \\
 (S_i(k), sfc_i(b), G_i) &\xrightarrow[\text{dequantization}]{\text{iQuantizer}} \hat{Y}_i(k) \xrightarrow[\text{inverse filter}]{\text{iTNS}} \hat{X}_i(k) \\
 \hat{X}_i(k) &\xrightarrow[\text{iMDCT + window}]{\text{iFilterBank}} \hat{x}_i[n] \xrightarrow[\text{50\% overlap-add}]{\text{Overlap-Add}} \hat{x}[n] \text{ (WAV, time)}
 \end{aligned}$$

Ο κωδικοποιητης δημιουργει ενα αντικειμενο `aac_seq_3`, δηλαδη μια λιστα απο dictionaries, οπου για καθε frame αποθηκευονται:

- ο τυπος frame (OLS, LSS, LPS, ESH)
- για καθε καναλι (chl, chr): το huffman bitstream των S , το codebook, το huffman bitstream των scalefactors sf_c , το global gain G , και οι συντελεστες TNS

Επιπλεον, αποθηκευεται και ενα CSV αρχαιο με μετρικες ανα frame (bits, codebook, non-zero S , αριθμος επαναληψεων quantizer).

Στην υλοποιηση μου, το αρχαιο περιεχει τις εξης συναρτησεις:

- `aac_coder_3`: Κωδικοποιει ενα WAV αρχαιο σε AAC Level 3 και επιστρεφει τη δομη `aac_seq_3`.
- `i_aac_coder_3`: Αποκωδικοποιει τη δομη `aac_seq_3` και παραγει το ανακατασκευασμενο WAV αρχαιο.

9.1 `aac_coder_3(filename_in, filename_aac_coded)`

Η συναρτηση αυτη υλοποιει τον κωδικοποιητη AAC Level 3. Διαβαζει το αρχαιο ηχου, το σπαι σε επικαλυπτομενα frames και κωδικοποιει καθε frame ξεχωριστα.

1. Φορτωση και παραμετροι

Διαβαζεται το WAV αρχαιο (στερεο) και οριζονται:

- $N = 2048$ (frame_size)
- `hop_size = 1024` (50% overlap)

Επισης φορτωνεται ο πινακας Huffman codebooks (LUT) με τη συναρτηση `load_LUT()`.

2. Padding και αριθμος frames

Υπολογιζεται ο αριθμος frames που απαιτουνται ωστε να καλυφθει ολο το σημα. Αν τα τελευταια δειγματα δεν επαρκουν για πληρες frame, γινεται padding με μηδενικα δειγματα.

3. Μεταβλητες καταστασης

Ο κωδικοποιητης κραται μνημη απο:

- τον προηγουμενο τυπο frame (`prev_frame_type`)

- τα 2 προηγούμενα frames στο πεδίο του χρόνου ($prev_frame_T$), τα οποία χρειάζονται στο psychoacoustic model (predictability)

4. Επεξεργασία ανα frame

Για κάθε frame:

- Εξάγεται το τρεχόν frame $frame_T$ μήκους 2048.
- Εξάγεται και το επομένο frame $next_frame_T$ (ή μηδενικό αν δεν υπάρχει) για να γίνει η επιλογή τύπου frame από το SSC.
- Υπολογίζεται ο τύπος frame (OLS, ESH, LSS, LPS) με τη συνάρτηση $SSC(frame_T, next_frame_T, prev_frame_type)$.
- Δημιουργείται ένα dictionary για το frame με πεδία για τα δύο κανάλια.

5. Επεξεργασία ανα κανάλι

Για κάθε κανάλι (chl, chr) εκτελούνται διαδοχικά τα βήματα:

5a. Filter Bank (MDCT)

Εφαρμόζεται η βαθμίδα filter bank ώστε να πάρουμε τους συντελεστές MDCT. Για ESH frames, γίνεται reshape σε 8 subframes (128x8).

5b. TNS

Εφαρμόζεται το Temporal Noise Shaping. Επιστρέφονται οι φιλτραρισμένοι συντελεστές (που θα είναι εισόδος στον quantizer) και οι συντελεστές a_1, a_2, a_3, a_4 του φίλτρου TNS, οι οποίοι αποθηκεύονται ώστε να χρησιμοποιηθούν στον αποκωδικοποιητή.

5c. Psychoacoustic Model

Υπολογίζεται το SMR για το κανάλι, με εισόδο:

- το τρεχόν frame στο χρόνο
- τον τύπο frame
- τα 2 προηγούμενα frames για predictability

Το SMR αποθηκεύεται στο dictionary του καναλιού.

5d. Quantizer

Κβαντίζονται οι TNS συντελεστες με τη συνάρτηση `aac_quantizer`.

Παραγονται:

- S : οι κβαντισμένοι συντελεστες
- sfc : τα DPCM scalefactors
- G : το global gain

Το G αποθηκεύεται, ενώ επιπλέον γίνεται εκτύπωση debug πληροφοριών ανά τακτά διαστήματα (SMR, P, T, Pe, alpha, bits) για long frames, χρησιμοποιώντας τη `get_last_quantizer_stats()`.

5e. Huffman Encoding

Κωδικοποιούνται ξεχωριστά:

- Τα scalefactors sfc με `force_codebook = 11` (όπως ζητείται), παραγοντας ένα bitstream `sfc_stream`.
- Οι κβαντισμένοι συντελεστες S με αυτοματη επιλογή codebook με βάση το `maxAbs`, παραγοντας bitstream `stream` και το codebook.

Τα bitstreams και το codebook αποθηκεύονται στο dictionary του καναλιού.

6. Logs μετρικών

Για κάθε frame και κανάλι, καταγράφονται σε CSV:

- `bits_S`, `bits_sfc`
- `nonzero` (πληθος μη μηδενικών S)
- `maxAbs` (μεγιστη απολυτη τιμη S)
- `codebook` που επιλεχθηκε
- αριθμος επαναληψεων του quantizer

7. Ενημέρωση καταστάσης

Στο τέλος κάθε frame ενημερώνονται:

- prev_frame_type
- τα 2 προηγούμενα frames prev_frame_T

και προστιθεται το dictionary του frame στη λιστα aac_seq_3.

Τελικα η συναρτηση επιστρεφει τη λιστα aac_seq_3 και αποθηκευει το CSV μετρικων.

9.2 i_aac_coder_3(aac_seq_3, filename_out)

Η συναρτηση αυτη υλοποιει τον αποκωδικοποιητη AAC Level 3. Παιρνει ως εισοδο τη δομη aac_seq_3 που παρηχθη απο τον κωδικοποιητη και ανακατασκευαζει το σημα στο χρονο.

1. Αρχικοποιηση

Φορτωνεται το Huffman LUT και οριζονται $N = 2048$, hop_size=1024. Δημιουργειται πινακας εξοδου audio_out με το συνολικο πληθος δειγματος.

2. Αποκωδικοποιηση ανα frame και καναλι

Για καθε frame:

- Διαβαζεται ο τυπος frame απο το dictionary.
- Για καθε καναλι γινεται Huffman decoding:
 - Τα scalefactors απο το codebook 11.
 - Τα S απο το codebook που εχει αποθηκευτει. Αν το codebook ειναι 0, τοτε σημαινει all-zero section και τα S θετονται μηδενικα.
- Γινεται reshape/padding ωστε τα S να εχουν παντα 1024 συντελεστες για long frames (η να αντιστοιχουν σε ESH οπως απαιτειται).
- Γινεται inverse quantizer (i_aac_quantizer) για να παρουμε τους αποκβαντισμενους TNS συντελεστες.
- Εφαρμοζεται inverse TNS (i_tns) χρησιμοποιωντας τους αποθηκευμενους tns_coeffs.

- Οι τελικοί συντελεστες MDCT τοποθετούνται στον πίνακα `frame_F_decoded`.

3. Inverse Filter Bank

Εφαρμόζεται η `i_filter_bank` για να επιστρέψει το `frame` στο πεδίο του χρόνου.

4. Overlap-Add

Το ανακατασκευασμένο `frame` προστίθεται στη σωστή θέση του `audio_out` με `overlap-add`, λόγω 50% επικάλυψης μεταξύ διαδοχικών `frames`.

5. Αποθήκευση

Στο τέλος, το `audio_out` γραφεται σε WAV αρχείο και επιστρέφεται.

10 Demo_AAC_3

Το αρχείο αυτό υλοποιεί ένα demo για τον AAC Level 3. Σκοπός του είναι να τρέξει ολοκληρωτό pipeline κωδικοποίησης και αποκωδικοποίησης, και στη συνέχεια να υπολογίσει βασικές μετρικές αξιολόγησης της συμπίεσης, δηλαδή:

- SNR (Signal to Noise Ratio) μεταξύ αρχικού και ανακατασκευασμένου σήματος
- bitrate του κωδικοποιημένου stream (bits/sec)
- compression ratio σε σχέση με το αρχικό PCM WAV

Η συνάρτηση του demo καλεί τον κωδικοποιητή `aac_coder_3` και τον αποκωδικοποιητή `i_aac_coder_3`. Μετά την αποκωδικοποίηση, γίνεται ευθυγραμμισμός μήκους του σήματος ώστε να είναι δυνατή η σύγκριση δείγμα προς δείγμα, και τέλος υπολογίζονται οι μετρικές.

Στην υλοποίησή μου, το αρχείο περιέχει την εξής συνάρτηση:

- `demo_aac_3`: Εκτελεί κωδικοποίηση, αποκωδικοποίηση, και υπολογίζει SNR, bitrate και compression.

10.1 `demo_aac_3(filename_in, filename_out, filename_aac_coded)`

Η συνάρτηση αυτή υλοποιεί την επίδειξη του AAC Level 3.

Παίρνει ως ορίσματα:

- `filename_in`: το αρχικό WAV αρχείο εισόδου
- `filename_out`: το WAV αρχείο εξόδου (decoded)
- `filename_aac_coded`: το αρχείο στο οποίο αποθηκεύεται το κωδικοποιημένο αποτέλεσμα (π.χ .mat)

Επιστρέφει:

- SNR: το Signal to Noise Ratio σε dB
- bitrate: το bitrate του κωδικοποιημένου stream σε bits/sec
- compression: το compression ratio σε σχέση με το αρχικό PCM WAV

1. Φορτώση αρχικού ηχού και ρύθμιση stereo

Διαβάζεται το WAV αρχείο εισόδου. Υπολογίζονται:

- $n_samples$: πλήθος δειγμάτων
- $duration = \frac{n_samples}{f_s}$: διάρκεια σε δευτερόλεπτα

2. Κωδικοποίηση και αποκωδικοποίηση

Καλούνται διαδοχικά:

```
aac_seq_3 = aac_coder_3(filename_in, filename_aac_coded)
```

```
audio_decoded = i_aac_coder_3(aac_seq_3, filename_out)
```

Η δομή `aac_seq_3` περιέχει τα κωδικοποιημένα frames και ο decoder παράγει το ανακατασκευασμένο σήμα στο πεδίο του χρόνου.

3. Ευθυγραμμισμός μηκών για σύγκριση

Επειδή η αποκωδικοποίηση μπορεί να παράγει σήμα ελαφρώς μεγαλύτερου ή μικρότερου μήκους (λόγω padding), το decoded σήμα ρυθμίζεται ώστε να έχει ακριβώς $n_samples$:

- αν είναι μεγαλύτερο, κοβεται
- αν είναι μικρότερο, γίνεται padding με μηδενικά

4. Υπολογισμος SNR

Υπολογιζεται η ισχυς του αρχικου σηματος:

$$P_s = \sum x^2$$

και η ισχυς του θορυβου (διαφορα input-output):

$$P_n = \sum (x - \hat{x})^2$$

και τελικα:

$$SNR = 10 \log_{10} \left(\frac{P_s}{P_n} \right)$$

Αν το P_n είναι πολυ μικρο, το SNR θεωρειται απειρο. Εδω φαινεται και αυτο που λεγαμε, πως οταν $a \rightarrow \infty \Rightarrow S \rightarrow 0 \Rightarrow \hat{X}(k) \rightarrow 0 \Rightarrow P_e \rightarrow \sum X(k)^2 = P_s \Rightarrow SNR \rightarrow 0 \text{ dB}$

5. Υπολογισμος bitrate

Υπολογιζεται το συνολικο πληθος bits του κωδικοποιημενου stream διατρεχοντας ολα τα frames στη δομη aac_seq_3. Για καθε frame προστιθενται bits που αντιστοιχουν σε metadata και στα κωδικοποιημενα δεδομενα καθε καναλιου:

- bits για frame_type / win_type (σταθερο μικρο overhead)
- bits για τους TNS coeffs
- bits για το global gain G
- bits για scalefactors (sfc), το Huffman stream (stream) και το codebook

Το bitrate υπολογιζεται ως:

$$bitrate = \frac{total_bits}{duration}$$

6. Υπολογισμος compression ratio

Το αρχικο bitrate του PCM WAV (stereo, 16-bit) ειναι:

$$original_bitrate = 2 \cdot 16 \cdot f_s$$

και το compression ratio:

$$compression = \frac{original_bitrate}{bitrate}$$

Τελικα η συναρτηση επιστρεφει τις μετρικες (SNR, bitrate, compression).

11 Αποτελεσματα Level 1 - Level 2

Τα αποτελεσματα που πηρα για το Level 1 και Level 2 ειναι ιδια. Αυτο συμβαινει διοτι στο Level 1 εφαρμοζουμε απλα input -> SSC -> Filterbank -> iFilterbank -> output, ενω στο Level 2: input -> SSC -> Filterbank -> TNS -> iTNS -> iFilterbank -> output. Επειδη οι Filterbank και TNS βαθμιδες ειναι πληρως αντιστρεψιμες, εχουμε ελαχιστες απωλειες που οφειλονται σε στρογγυλοποιησεις και υπολογισμους, γι'αυτο και πετυχαινουμε πολυ μεγαλο και ιδιο SNR.

Level	SNR (dB)
Level 1	254.03629
Level 2	254.0363

```
PS C:\Users\STAMATIS\Documents\POLYMESA> & C:/Users/STAMATIS/Scripts/level1.py
SNR = 254.0362962527834 dB
PS C:\Users\STAMATIS\Documents\POLYMESA> & C:/Users/STAMATIS/Scripts/level2.py
Level 2 SNR = 254.0363 dB
PS C:\Users\STAMATIS\Documents\POLYMESA> 
```

Επισης στα παραδοτεα αρχεια, υπαρχει και το output wav των Level 1 και Level 2. Ακουγεται ιδιο με το αρχικο, αφου οπως ειπαμε εχουμε ελαχιστες απωλειες πληροφοριας. Τα αποτελεσματα μπορειτε να τα παρετε αν τρεξετε το run_level_1.py και run_level_2.py αρχεια.

12 Αποτελέσματα Level 3

Ενδεικτικά αποτελέσματα μερικων frames και των μετρικων SNR, bitrate, Compression για το τελικο παραγομενο αρχειο φαινονται παρακατω:

```
sfc values: [ -19, 22] | Non-zero S: 342/1024 | maxAbs: 3 | codebook: 5

Frame 243 | Type: OLS | Channel: chl
SMR:      [ 0.09, 35.34]
P(b):     [3.67e-02, 1.38e+04] (MDCT energy)
npart(b): [2.34e-02, 2.87e+02] (threshold apo psycho)
T(b):     [1.24e-02, 9.82e+03] (=P/SMR)
Pe initial: [1.45e-06, 1.75e-04]
alpha:    [ -13.83, 18.17] | Iter: 59 | reached max iterations

[Band 20 Check] SMR=1.86e+01, P=1.26e+04, T=6.75e+02
T = P/SMR = 6.75e+02 (Match: True)
Pe(initial) = 1.88e-05 (Pe < T: True)
[Bits] sfc: 291 bits | S: 1452 bits | Total: 1743 bits
sfc values: [ -13, 18] | Non-zero S: 330/1024 | maxAbs: 3 | codebook: 5

Frame 270 | Type: OLS | Channel: chl
SMR:      [ 0.04, 38.07]
P(b):     [1.15e-03, 3.63e+04] (MDCT energy)
npart(b): [1.42e-01, 3.22e+02] (threshold apo psycho)
T(b):     [2.38e-04, 7.68e+03] (=P/SMR)
Pe initial: [2.05e-06, 1.98e-04]
alpha:    [ -29.54, 20.46] | Iter: 59 | reached max iterations

[Band 20 Check] SMR=2.10e-01, P=3.47e+01, T=1.65e+02
T = P/SMR = 1.65e+02 (Match: True)
Pe(initial) = 3.30e-05 (Pe < T: True)
[Bits] sfc: 303 bits | S: 1348 bits | Total: 1651 bits
sfc values: [ -21, 20] | Non-zero S: 303/1024 | maxAbs: 4 | codebook: 5

[INFO] Metrics saved to: coded_level3_metrics.csv
Level 3
SNR = 10.80 dB
Bitrate = 177.02 kbps
Compression = 8.68x
PS C:\Users\STAMATIS\Documents\POLYMESA>
```

Εδω βλεπουμε τα διαστηματα στα οποια ανηκουν οι μετρικες για καθε μπαντα, με τα ακρα του διαστηματος να ειναι η ελαχιστη και μεγιστη τιμη τους. Προφανως, οι τιμες στα ακρα δεν αντιστοιχουν απαραιτητα στην 1η και τελευταια μπαντα συχνοτητων, γι'αυτο απο κατω εκανα και εναν ελεγχο για την μπαντα 20, ωστε να επαληθευσω οτι ισχυει η σχεση $T = \frac{P}{SMR}$. Οι γραμμες στα logs

που αφορούν αυτό τον έλεγχο είναι από εκεί που λέει [Band 20 Check] μέχρι εκεί που λέει ($P_e < T$: True). Οι μετρικές που βλέπουμε για κάθε frame είναι:

- $SMR(b) = e(b)/npart(b)$ που δίνει το ψυχοακουστικό μοντέλο
- $P(b) = \sum_{k=w_{low}}^{w_{high}} X(k)^2$ η ενεργεια των MDCT συντελεστών
- $npart(b)$: Ουσιαστικά το κατώφλι ακουστότητας του θορύβου για τη μπάντα b .
- $T(b)$: το κατώφλι ακουστότητας του κωδικοποιητή
- $P_e(b)$: Η αρχική ισχύς του σφαλματος θορύβου κβαντισμού, δηλαδή πριν την αύξηση του $a[b]$
- α : Οι συντελεστές scalefactor gain για κάθε μπάντα b
- sfc values: Οι κωδικοποιήσεις $sfc(b) = \Delta a(b)$
- Non-zero S : Οι μη μηδενικές κωδικοποιήσεις S των $X(k)$ (των $Y(k)$ στην πραγματικότητα, γιατί ο quantizer παίρνει τα φίλτραρισμένα X , δηλαδή την έξοδο Y του TNS).
- $maxAbs$: Το μέγιστο κατά απόλυτη τιμή S
- $codebook$: Το codebook που χρησιμοποίησε ο αλγόριθμος για την huffman κωδικοποίηση των S (το οποίο επιλέγεται με βάση το $maxAbs$). Για το huffman encoding των sfc χρησιμοποιήθηκε το codebook 11 από εκφώνηση.
- $Bits$ row: sfc bits, S bits, Total bits, τα αντίστοιχα bits που χρειάστηκαν για την κωδικοποίηση της κάθε τιμής.

Σημειώνω εδώ ότι, για τα scalefactors, το $force_codebook = 11$ που ζητήσατε, μου πέτυξε indexing error. Οι τιμές των scalefactor έβγαιναν μεγάλες, και επειδή ο αλγόριθμος πήγαινε κατευθείαν στην $huff_LUT_code_1$, ο τρόπος με τον οποίο αυτή υπολογίζει το huffman index για την κωδικοποίηση, έδινε index out of bounds,

δηλαδή index που δεν υπήρχε στο huffman codebook. Γι'αυτό τροποποιήσα το κομμάτι της encode_huff από:

```
if force_codebook is not None:
    return huff_LUT_code_1(huff_LUT_list[force_codebook],
                           coeff_sec), force_codebook
```

σε :

```
if force_codebook is not None:
    if force_codebook == 11 and np.max(np.abs(coeff_sec)) > 15:
        return huff_LUT_code_ESC(huff_LUT_list[force_codebook],
                                   coeff_sec), force_codebook
    return huff_LUT_code_1(huff_LUT_list[force_codebook],
                           coeff_sec), force_codebook
```

έτσι ώστε να μπαίνει σε ESC huff αν για αυτό το frame βρίσκεται $|a| > 15$ σε κάποιο band.

Για τον quantizer, έχουμε 2 κριτήρια τερματισμού αύξησης του $a[b]$ στην εκφώνηση:

Το $Pe(b) \geq T(b)$ και το $a(b+1) - a(b), a(b) - a(b-1) > 60$

Την επαναληπτική αύξηση των a την υλοποιήσα κατά βάση ως εξής:

```
for iteration in range(60):
    all_done = True

    for b in range(n_bands):
        Pe = np.sum((X[w_low[b]:w_high[b]+1] - X_hat[w_low[b]:
                                                       w_high[b]+1])**2)

        # An Pe < T(b), thelouve na auxhsoume to alpha[b]
        if Pe < T[b]:
            can_increase = True

            if b > 0:
                if abs((alpha[b] + 1) - alpha[b-1]) > 60:
                    can_increase = False

            if b < n_bands - 1:
                if abs(alpha[b+1] - (alpha[b] + 1)) > 60:
                    can_increase = False

            if can_increase:
                alpha[b] += 1
                all_done = False
```

```
# Den eixame allagh sta alpa se kanena band, opote exoume th
    veltisth sympiesh
if all_done:
    break
```

Οπως βλέπουμε, ο αλγοριθμος αποτελείται απο ενα εξωτερικο loop επαναληψεων, και ενα εσωτερικο για καθε μπαντα b .

Το εξωτερικο loop, το εβαλα γιατι μπορεί τα 2 κριτηρια να θελουν παρα πολλές επαναληψεις να ικανοποιηθούν, και μπορεί τα a να αυξηθούν τοσο πολυ που να μου κανουν πρακτικα το σημα αχρηστο (ναι μεν θα εχω πολυ μεγαλη συμπιεση, αλλα ο θορυβος κβαντισμου μπορεί να αυξηθει υπερβολικα και το decoded σημα να ειναι απλα θορυβος).

Ξεκινώντας απο το εσωτερικο loop των μπαντων, πρωτα ελεγχουμε αν $P_e(b) < T(b)$. Αν ισχυει, τοτε μονο ελεγχουμε το 2ο κριτηριο ωστε να δουμε αν μπορεί να αυξηθει το $a(b)$. Χρησιμοποιησα δυο flags, το `all_done`, και το `can_increase`. Το `can_increase`, ειναι flag που δειχνει αν ικανοποιεται το 2ο κριτηριο. Αν δηλαδη $a(b) \leq 60$ και για τους 2 γειτονες του $a(b)$, τοτε το $a(b)$ αυξανεται κατα 1. Διαφορετικα μενει σταθερο και προχωραμε στο επομενο band.

Ο τροπος που ελεγχσα το 2ο κριτηριο, ειναι με το σκεπτικο "Αν η αυξηση του $a[b]$ κατα 1 οδηγησει σε διαφορα μεγαλυτερη του 60, τοτε μην το αυξησεις". Χωρισα τον ελεγχο σε 2 if, για τις ακραιες περιπτωσεις $b = 0$ και $b = last_band$. Αν $b = 0$, δεν υπαρχει προηγουμενη μπαντα αρα πρεπει να ελεγχθει μονο η διαφορα του $a[b]$ με την επομενη. Αντιστοιχα αν $b = last_band$.

Το 2ο flag: `all_done` δειχνει αν ικανοποιουνται και τα 2 κριτηρια τερματισμου ταυτοχρονα. Αν οχι, τοτε υπαρχει καποιο b που μπορεί ακομα να αυξηθει. Αλλιως, σταματαμε την αυξηση ολων των a . Αρχικοποιεται ως `True`, και αν για καποιο band το `can_increase` ειναι `true`, τοτε το $a(b)$ μπορεί να αυξηθει και το `all_done` γινεται `False`. Αν για ολα τα bands b , το `can_increase` δεν γινει ποτε `true`, τοτε σημαινει οτι τελειωσαμε, δηλαδη δεν μπορεί καποιο $a(b)$

να αυξηθει περαιτερω. Οποτε βγαινουμε απο τα 2 loops. Αν το `all_done` δεν γίνει ποτε True, τότε ο αλγοριθμος σταματαει στο `max_iteration` που εθεσα, δηλαδη στην παραπανω περιπτωση, στις 60 επαναληψεις.

Με τον αλγοριθμο μου, συναντησα ενα παραδοξο. Οπως ειπαμε, οι `max iterations`, ειναι οι φορες που θα σαρωσουμε ολα τα $a(b)$, δηλαδη τα a για καθε μπαντα, ωστε να δουμε αν καποιο απο αυτα μπορει να αυξηθει. Αν π.χ $max_iter = 30$, τότε 30 φορες θα σαρωσουμε τις μπαντες b , και σε καθε σαρωση ελεγχουμε αν καθε $a(b)$ μπορει να αυξηθει. Επομενως, καθε $a(b)$ θα αυξηθει το μεγα στο κατα 30, αφου αυξανουμε κατα 1.

Οσο πιο μικρο λοιπον το `max_iter`, τοσο πιο λιγο θα αυξηθουν τα `scalefactor gain`, αρα τοσο πιο λιγη θα ειναι η συμπιεση, και τοσο πιο λιγος ο θορυβος κβαντισμου δηλαδη μεγαλυτερο το SNR (καλυτερο decoded σημα). Αντιστροφα, οσο πιο μεγαλο το `max iterations`, τοσο μεγαλωνει η συμπιεση, αρα μικραινει το SNR και το Bitrate (λιγοτερα bits ανα δευτερολεπτο σηματος).

Τα αποτελεσματα που ειχα με $max_iter = 60$, ειναι αυτα που ειδαμε παραπανω:

- $SNR = 10.8dB$
- $Bitrate = 177.02kbps$
- $Compression = 8.68x$

Το παραδοξο που συναντησα ειναι πως οταν μεγαλωσα αρκετα τα `max iterations`, το SNR επεφτε οπως αναμεναμε, ομως επεφτε και το Compression. Για να καταλαβω γιατι συνεβαινε αυτο, εβαλα στα logs τα bits που χρειαστηκαν για την κωδικοποιηση των sfc, των S, και το αθροισμα τους: Total bits για καθε frame. Επισης τα sfc values και τα Non-zero S, δηλαδη ποσα $S(k)$ ειναι μη μηδενικα. Προφανως με αυξηση του α , τα S μικραινουν, οποτε περιμενουμε λιγοτερα Non-Zero S.

Παρατηρησα, οτι μετα απο ενα σημειο (60 max iterations) τα non-zero S σωστα μειωνωνταν, ομως τα sfc bits αυξανονταν αρκετα. Αυτο οφειλεται στις μεγαλες τιμες που παιρνουν τα α και συγκεκριμενα στις μεγαλες διαφορες α που δημιουργουνται, οποτε αυξανονται τα bits που απαιτουνται για την κωδικοποιηση των sfc. Και μαλλον η encode huff δεν κανει καλη δουλεια για μεγαλα scalefactors-sfc (π.χ δεν επιλεγεται καλο codebook, ή τα codebooks δεν ειναι βελτιστοποιημενα για μεγαλα α). Οποτε τα συνολικα Bits αυξανονται και γι αυτο αυξανεται το Bitrate και πεφτει η συμπιεση.

Δοκιμασα διαφορες λυσεις, οπως να μην κωδικοποιω το $sfc(0)=G=a(0)$, καθως δεν το χρειαζομαι στην κωδικοποιηση, και νομιζω πως επαιρνε μεγαλες τιμες. Το συνειδητοποιησα απο τα sfc values, για τα οποια αναρωτηθηκα "Δε θα πρεπε αφου ειναι διαφορες γειτονικων α , να ειναι ολα απο -60 μεχρι 60 στην χειροτερη περιπτωση? ". Σε μερικα frames το μεγαστο sfc ηταν αρκετα πανω απο το 60. Αυτο μαλλον ηταν το $G=a(0)$. Ωστοσο δεν βοηθησε η αφαιρεση του απο την κωδικοποιηση.

Επισης δοκιμασα να βαλω ενα cap στις τιμες του α , ως ενα εξτρα κριτηριο τερματισμου. Δηλαδη το $\alpha(b)$ να μπορει να αυξηθει μονο αν $\alpha(b) < 130$. Αλλα ουτε αυτο βοηθησε στο προβλημα. Παρακατω παραθετω τα αποτελεσματα που ελαβα οταν ετρεξα το προγραμμα για διαφορετικα max iterations:

Max Iter	SNR (dB)	Bitrate (kbps)	Compression
40	24.71	372.59	4.12x
45	22.20	354.03	4.34x
50	18.82	255.18	6.02x
55	14.73	243.57	6.31x
60	10.80	177.02	8.68x
65	7.68	206.07	7.45x
70	6.44	207.86	7.39x
75	5.97	209.72	7.32x
100	5.97	214.14	7.17x
250	5.97	218.00	7.05x

Παρατηρούμε ότι μετά τις 60 επαναλήψεις εμφανίζεται το παραδοξο που ανέφερα. Για κάποιο λόγο το $max_iter = 60$ αποτελεί ένα cap βελτιστοποίησης. Δε μπορώ να καταλάβω γιατί συμβαίνει αυτό. Προφανώς κάτι έχω κάνει λάθος. Ωστόσο ακολουθήσα τις οδηγίες κατά γραμμή. Ίσως φταιει η αλλαγή στο huff που έκανα, όμως μου πέτυχε error όταν δεν το έβαζα. Ίσως το κριτήριο τερματισμού με max iterations δεν είναι σωστό. Δε ξέρω. Πάντως μέχρι τις 60 επαναλήψεις ο αλγόριθμος δουλεύει καλά και πετυχαίνω μια καλή συμπίεση.

Παρακάτω βλέπουμε ένα αποσπασμα του CSV αρχείου με μετρικές για κάθε frame, για $max_iterations = 60$, και ένα για $max_iter = 250$. Το αρχείο το έχω συμπεριλάβει στην παραδοχή.

Max_iter = 60 :

```
coded_level3_metrics.csv
1  frame,channel,frame_type,bits_S,bits_sfc,nonzero,maxAbs,codebook,iterations
2  0,ch1,OLS,1482,282,371,3,5,59
3  0,chr,OLS,1505,263,373,3,5,59
4  1,ch1,OLS,1435,299,349,3,5,59
5  1,chr,OLS,1555,277,386,3,5,59
6  2,ch1,OLS,1366,280,313,3,5,59
7  2,chr,OLS,1332,299,305,3,5,59
8  3,ch1,OLS,1382,301,317,3,5,59
9  3,chr,OLS,1351,282,308,3,5,59
10 4,ch1,OLS,1419,278,333,3,5,59
11 4,chr,OLS,1392,276,323,3,5,59
12 5,ch1,OLS,1348,286,299,3,5,59
13 5,chr,OLS,1409,266,318,3,5,59
14 6,ch1,OLS,1363,288,310,3,5,59
15 6,chr,OLS,1274,300,280,3,5,59
16 7,ch1,OLS,1367,289,304,3,5,59
17 7,chr,OLS,1327,281,289,3,5,59
18 8,ch1,OLS,1309,311,289,3,5,59
19 8,chr,OLS,1361,315,304,3,5,59
20 9,ch1,OLS,1289,300,284,3,5,59
```

Max_iter = 250 :

```
coded_level3_metrics.csv
1  frame,channel,frame_type,bits_S,bits_sfc,nonzero,maxAbs,codebook,iterations
2  0,ch1,OLS,2071,423,348,2,3,189
3  0,chr,OLS,2097,402,357,2,3,185
4  1,ch1,OLS,2074,374,344,2,3,123
5  1,chr,OLS,2139,371,374,2,3,173
6  2,ch1,OLS,1974,351,298,2,3,179
7  2,chr,OLS,1956,381,295,2,3,236
8  3,ch1,OLS,2004,396,305,2,3,126
9  3,chr,OLS,1992,386,300,2,3,125
10 4,ch1,OLS,1999,382,309,2,3,126
11 4,chr,OLS,2017,392,307,2,3,126
12 5,ch1,OLS,1297,390,285,3,5,125
13 5,chr,OLS,1998,389,301,2,3,125
14 6,ch1,OLS,1969,420,294,2,3,121
15 6,chr,OLS,1939,379,274,2,3,173
16 7,ch1,OLS,2017,406,292,2,3,124
17 7,chr,OLS,1948,395,276,2,3,127
18 8,ch1,OLS,1957,419,284,2,3,178
19 8,chr,OLS,1310,410,291,3,5,170
20 9,ch1,OLS,1252,393,276,3,5,120
```

Τελος, μεσα στα αρχεια παραδοσης υπαρχει και το output_level3.wav που ειναι το decoded wav του wav που μας δωσατε.

Ολα τα levels τρεχουν ανοιγωντας και εκτελωντας το αντιστοιχο run_level_x.py αρχειο, ειτε απο terminal, π.χ:

- python run_level_1.py αν ειστε μεσα στο root folder του project
- ή με πληρες path, π.χ: python C:/Users/.../run_level_1.py