Ε.Μ.Π. Σχολή Η.Μ.Μ.Υ.

Ψηφιακή Επεξεργασία Σημάτων

Ακαδημαϊκό Έτος 2019-2020

1η Εργαστηριακή Άσκηση

Εισαγωγή στην Ψηφιακή Επεξεργασία Σημάτων με Python και Εφαρμογές σε Ακουστικά Σήματα

Η παρακάτω εργαστηριακή αναφορά συντάχθηκε από τους φοιτητές:

- Κωστόπουλο Κωνσταντίνο με ΑΜ : 03117012
- Τσακανίκα Χριστίνα με ΑΜ : 03117012

Μέρος 1ο - Σύστημα Εντοπισμού Τηλεφωνικών Τόνων

Telephone Touch – Tones

Ερώτημα 1.1

Γράφουμε τον κάθε διαφορετικό τόνο ως άθροισμα ενός υψίσυχνου ημιτόνου, το οποίο εξαρτάται από τη στήλη που βρίσκεται το πλήκτρο, και ενός χαμηλόσυχνου ημιτόνου, το οποίο εκφράζει την αντίστοιχη γραμμή του πλήκτρου. Προς επίτευξη συγκέντρωσης χιλίων δειγμάτων, για τον κάθε ένα από τους δέκα τόνους, ορίζουμε το διακριτό χρόνο n χρησιμοποιώντας την εντολή “linspace” στην μορφή των τριών ορισμάτων, ως εξής: numpy.linspace(αρχή, τέλος, αριθμός δειγμάτων).

Γράφοντας, λοιπόν, n=np.linspace(1,1001,1000), εξασφαλίζουμε ότι ο κάθε τόνος θα έχει στο σύνολο 1000 δείγματα.

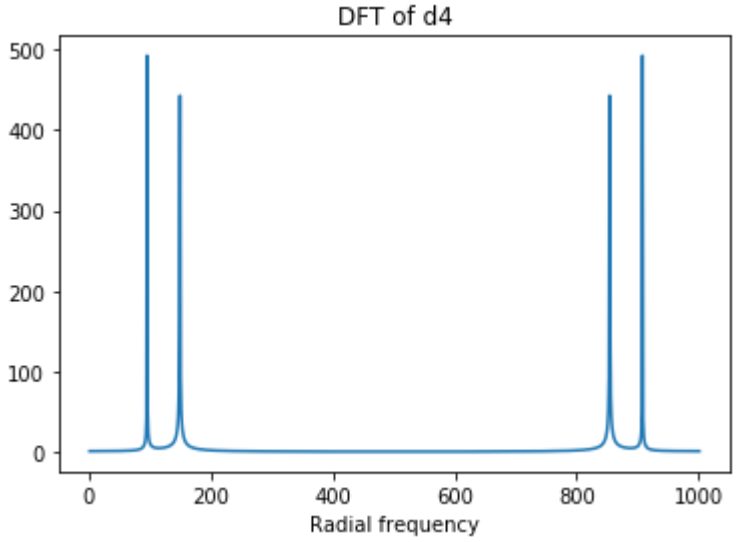
Βεβαιωνόμαστε για την ορθότητα της διαδικασίας χρησιμοποιώντας τη ρουτίνα “play()”, του πακέτου sounddevice, με την εντολή sd.play(d0,8192) ακούγοντας τον τόνο για το πλήκτρο 0, όπου 8192Hz η δεδομένη συχνότητα δειγματοληψίας.

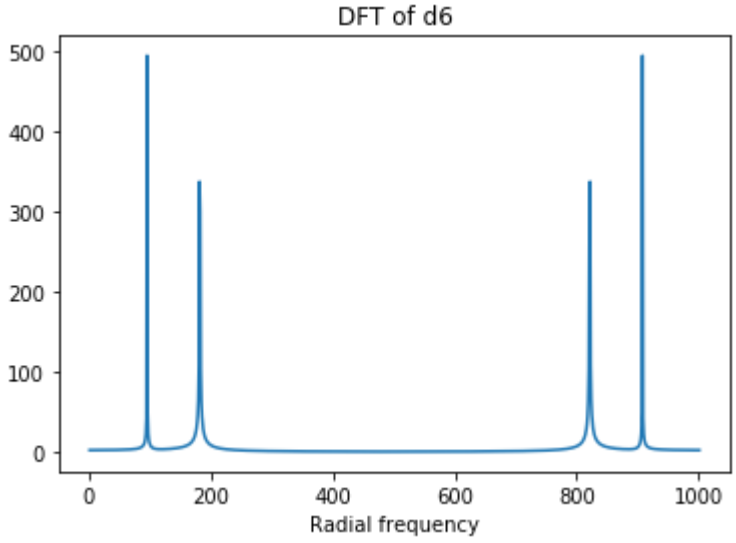
Ερώτημα 1.2

Για τον υπολογισμό των DFT των σημάτων d4[n] και d6[n], χρησιμοποιήθηκε η εντολή

Dftd4 = np.fft.fft(d4) & Dftd6 = np.fft.fft(d6), για το κάθε σήμα αντίστοιχα. Η γραφική αναπαράσταση των σημάτων γίνεται με την εντολή “plot” του πακέτου matplotlib,

ενώ την απόλυτη τιμή των σημάτων τη λαμβάνουμε με την εντολή np.abs(signal). Οι γραφικές παραστάσεις που λαμβάνουμε για το κάθε σήμα είναι αντίστοιχα:





Ερώτημα 1.3

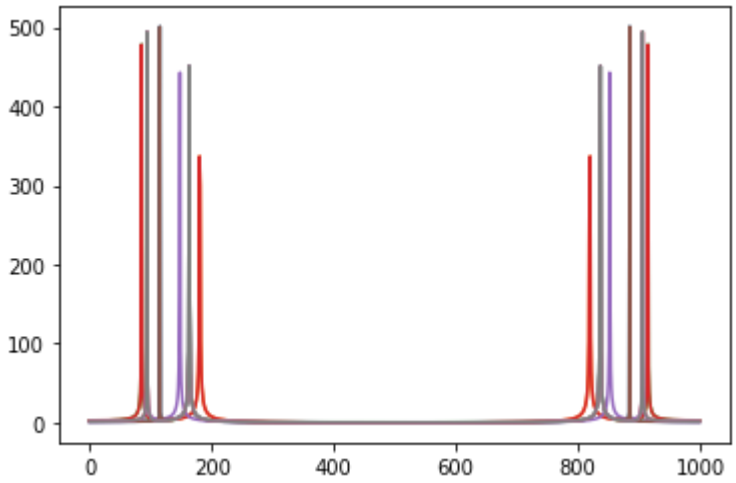
Αρχικά, δημιουργούμε το σήμα των διαδοχικών τηλεφωνικών τόνων, με τα ψηφία που προέκυψαν να είναι τα ακόλουθα: 0 6 2 3 4 0 5 5. Προκειμένου να διαχωρίσουμε κάθε ψηφίο του αθροίσματος από το προηγούμενό του με εκατό μηδενικά δείγματα, χρησιμοποιούμε την εντολή “concatenate”, η οποία, λαμβάνουσα δύο πίνακες ως ορίσματα, προσθέτει τον δεύτερο πίνακα στο τέλος του πρώτου. Χρησιμοποιώντας, λοιπόν, έναν πίνακα με εκατό μηδενικά ως δεύτερο όρισμα και μια for loop που θα επιλέγει κάθε φορά το κατάλληλο τόνο κατασκευάζουμε το ζητούμενο σήμα. Τέλος,

με την ρουτίνα write_wav(“αρχείο_ήχου.wav”, σήμα, συχνότητα δειγματοληψίας), δημιουργούμε και αποθηκεύουμε το αρχείο “tone_sequence.wav”.

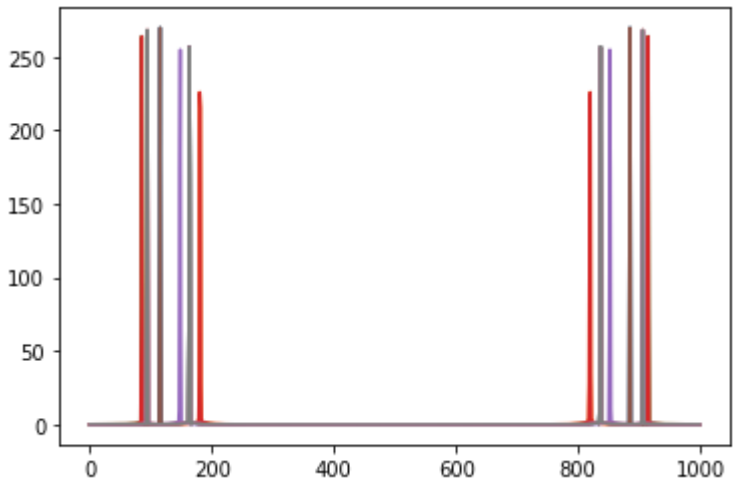
Ερώτημα 1.4

Δημιουργούμε τα χρονικά τετραγωνικά & Hamming παράθυρα με τις εντολές rect= signal.boxcar(1000) και hamm = signal.hamming(1000) αντίστοιχα. Με μία for με ορίσματα (0,8800, 1100) πολλαπλασιάζουμε τον κάθε τόνο με το αντίστοιχο χρονικό παράθυρο, παραλείποντας έτσι τα ανεπιθύμητα εκατό μηδενικά δείγματα. Αποθηκεύουμε τα αποτελέσματα στον πίνακα rect_windows/hamming αντίστοιχα. Στη συνέχεια, με μία ακόμη for με range οχτώ, που είναι ο αριθμός των παραθυροποιημένων σημάτων, υπολογίζουμε τον μετασχηματισμό Fourier του κάθε στοιχείου του πίνακα, γράφοντας “np.fft.fft(rect_windows/hamming[i]).

Το αποτέλεσμα που λάβαμε για τα τετραγωνικά παράθυρα είναι το ακόλουθο:



ενώ για τα Hamming παράθυρα αναπαρίσταται ως εξής :



Ερώτημα 1.5

Για το συγκεκριμένο ερώτημα, χρησιμοποιήσαμε τη συνάρτηση “freq_from_crossings” η οποία υπολογίζει τη συχνότητα ενός σήματος, βάσει τον αριθμό που τέμνει αυτό τον οριζόντιο άξονα. Τη συνάρτηση αυτή εντοπίσαμε στον διαδικτυακό ιστότοπο : https://gist.github.com/endolith/255291

Κατασκευάζοντας τους πίνακες “columns_sin” & “rows_sin” με τα υψίσυχνα και τα χαμηλόσυχνα ημίτονα αντίστοιχα και χρησιμοποιώντας δύο for loops τυπώνουμε τις αντίστοιχες συχνότητες που βρίσκονται εγγύτερα στις touch-tone συχνότητες.

Ερώτημα 1.6

Η συνάρτηση ttdecode καλεί την συνάρτηση windowing_and_fourier, η οποία σκοπό έχει να αφαιρέσει από το εκάστοτε σήμα τυχόν μηδενικά δείγματα. Για ένα σήμα sign, το παραπάνω επιτυγχάνεται με την εντολή sing=sign[sign!=0]. Έπειτα, τοποθετεί τον απόλυτο μετασχηματισμό Fourier του κάθε τόνου, με την εντολή abs(np.fft.fft()) στον πίνακα widnow. Τέλος, επιστρέφει τον πίνακα αυτόν. Για το σήμα που προκύπτει, χρησιμοποιείται η εντολή find_peaks η οποία βρίσκει τις συχνότητες για τις τέσσερις κορυφές του σήματος, ενώ στον πίνακα store_first_peaks αποθηκεύουμε τις δύο πρώτες συχνότητες εκ των τεσσάρων. Συγκρίνοντας τα στοιχεία του προηγούμενου πίνακα με αυτά του πίνακα των touch-tone συχνοτήτων που προέκυψαν στο ερώτημα 1.5, τυπώνουμε τα ψηφία του αθροίσματος των αριθμών μητρώου μας και επαληθεύουμε την ορθή λειτουργία της συνάρτησής μας.

Ερώτημα 1.7

Αρχικά, αποθηκεύουμε το συμπληρωματικό υλικό της άσκησης “dsp20 lab1 Data.zip”, στο ίδιο αρχείο που έχουμε αποθηκεύσει την άσκησή μας. Φορτώνουμε τα αρχεία easySig & hardSig με την εντολή easySig = np.load(“easySig ”) & hardSig = np.load(“hardSig ”) αντίστοιχα και καλούμε την συνάρτηση ttdecode() για το κάθε ένα από αυτά.

Το αποτέλεσμα τυπώνεται ως εξής:

The digits of easySig are : 1 3 2 6 3 9 0 0

The digits of hardSig are : 9 0 9 6 3 2 1 1 9 1

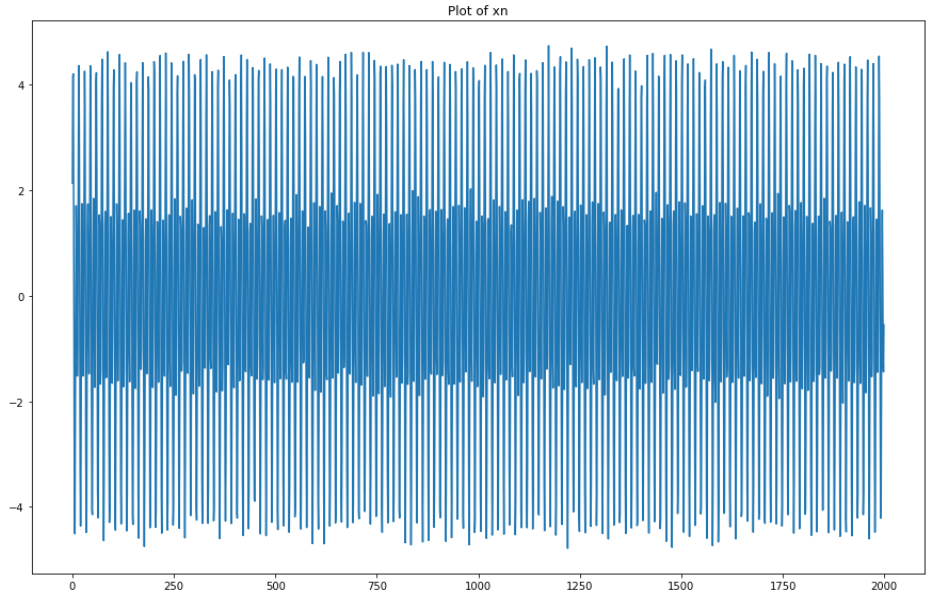
Μέρος 2ο - Φασματική Ανάλυση Ημιτονοειδών

Ανίχνευση Απότομων Μεταβάσεων με τον Μετ/σμό Fourier Βραχέος Χρόνου (STFT) και τον Μετ/σμό Wavelets (διακριτοποιημένο CWT)

Ερώτημα 2.1.

(α)

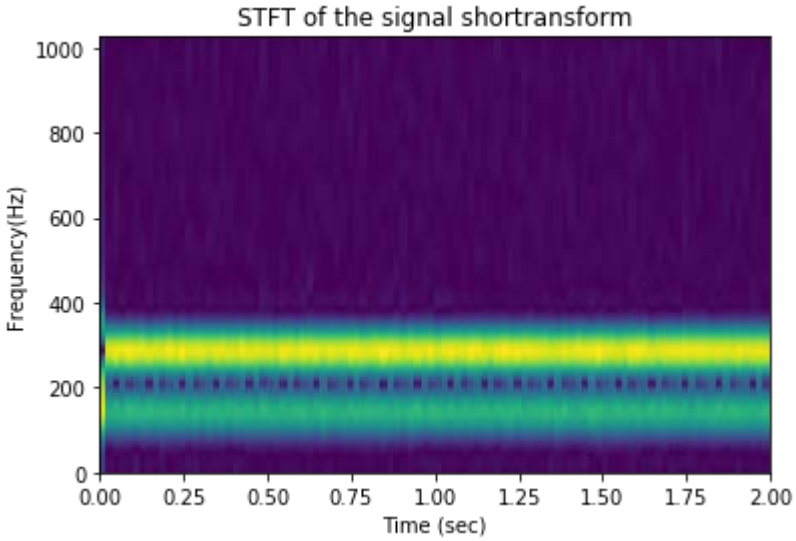
Αρχικά, ορίζουμε τον διακριτό χρόνο να εκκινεί από το μηδέν και να φτάνει μέχρι και το 1999 με την εντολή arrange. Σύμφωνα με τη δεδομένη ισότητα x[n] = x(nTs), μπορούμε να διαμορφώσουμε το σήμα x[n] αντικαθιστώντας στο x(t), όπου t το nTs, όπου Ts = $\frac{1}{Fs}$ = 0.001sec. Το διάγραμμα που λαμβάνουμε από την εντολή plot, για το σήμα x[n] αναπαριστάται παρακάτω:



(β)

Συμβουλευόμενοι το εργαστηριακό υλικό που είναι αναρτημένο στην ιστοσελίδα https://github.com/ntuacvsplab/DSP_LabSupport/blob/master/Audio_Analysis_Intro.ipynb καθορίζοντας στην εντολή stft τα ορίσματα ως (xn, 2048, 20, 40) και με συχνότητα δειγματοληψίας ίση με Fs=1000Hz, επιτυγχάνουμε να διαμορφώσουμε το μήκος παραθύρου ίσο με 0.04sec (0.04/0.001=40) και την επικάλυψη ίση με 0.02sec

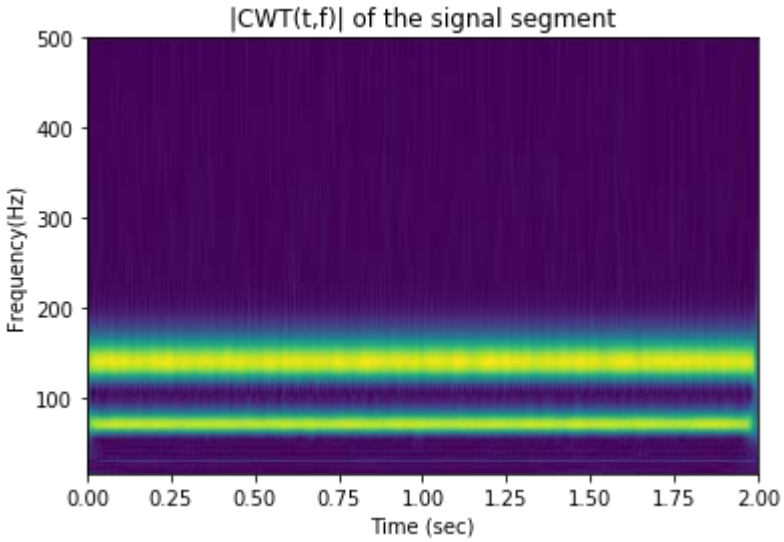
(0.02/0.001=20), ενώ η προκαθορισμένη τιμή των δειγμάτων στη βιβλιοθήκη της librosa ισούται με 2048. Προκειμένου να λάβουμε τις ορθές τιμές συχνότητας και χρόνου, χρησιμοποιούμε την εντολή print(shortransform.shape). Το παραπάνω μας αποδίδει στον άξονα των y τη συχνότητα ενώ στον άξονα των x τον χρόνο. Τα αποτελέσματα της print για το χρόνο και τη συχνότητα τα χρησιμοποιούμε ως βήματα για τις linspace που αφορούν το χρόνο και τι συχνότητα αντίστοιχα. Για τη δειγματοληψία στο διάστημα [0,2], γράφουμε t = np.linspace(0,2,101). Με την εντολή plt.pcolormesh(t, f,abs(shortransform)), αναπαριστούμε το πλάτος |STFT(τ, f)|. Η γραφική που προέκυψε παρουσιάζεται παρακάτω:



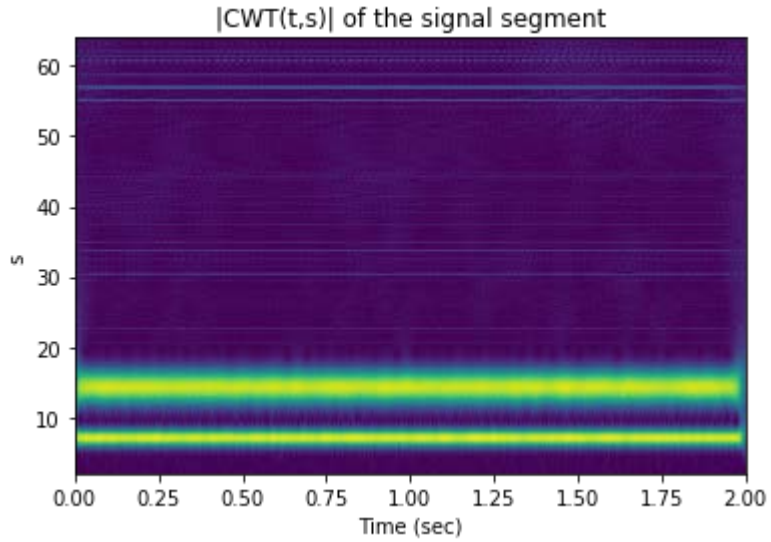
(γ)

Ορίζουμε το s ίσο με 20 κυμματίδια ανά οκτάβα με την εντολή s = np.power(2, np.linspace(1, 6, 1000)) και σύμφωνα με το εργαστηριακό υλικό επιλέγουμε το wavelet “cmor3.0-1.0” ως εξής : coefs, freqs = pywt.cwt(xn, s, ‘cmor3.0-1.0’).

Το αποτέλεσμα της print(coefs.shape) είναι 1000& 2000. Όμοια με πριν έχουμε t = np.linspace(0,2,2000) και f = freqs* 1000. Χρησιμοποιώντας την εντολή plt.pcolormesh(t, f, np.abs(coefs)), λαμβάνουμε την κάτωθι γραφική για τα πλάτη |CWT(τ, f)|.



Για τα πλάτη |CWT(τ, s)|, συναρτήσει των κλιμάκων του μετασχηματισμού, ακολουθούμε ακριβώς την ίδια διαδικασία, διαφοροποιώντας την εντολή plt μονάχα ως προς το δεύτερο όρισμα, δηλαδή: plt.pcolormesh(t, s, np.abs(coefs)). Στην προκειμένη περίπτωση, η γραφική διαμορφώνεται ως εξής:



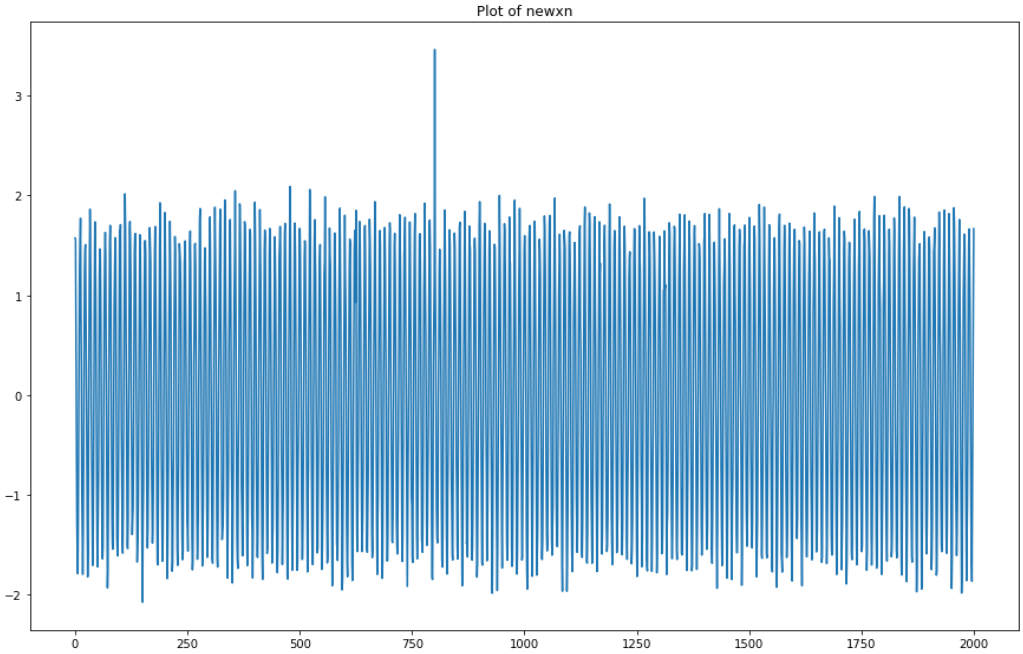
(δ)

Συγκρίνοντας την πρώτη γραφική, που αποτελεί το πλάτος του STFT του σήματος , με τις αντίστοιχες γραφικές του CWT καταλήγουμε στο ότι ο μετασχηματισμός των Wavelets έχει υψηλότερη ανάλυση στο χρόνο καθώς οι συχνότητες αυξάνονται και αντίστοιχα υψηλότερη ανάλυση στο πεδίο της συχνότητας για σήματα που διαρκούν περισσότερο από άλλα. Για τη χαμηλότερη ανάλυση του σήματος κατά τη φασματική ανάλυση με μετασχηματισμό Fourier βραχέος χρόνου ευθύνεται επίσης το μεγάλο παράθυρο στο χρόνο, που από τη μία πετυχαίνει καλή διακριτική ικανότητα στο πεδίο συχνοτήτων , υστερεί όμως σε ανάλυση στον χρόνο. Επίσης, οι DT-CWT δείχνουν να "μοιάζουν" με τον STFT.

2.2

(α)

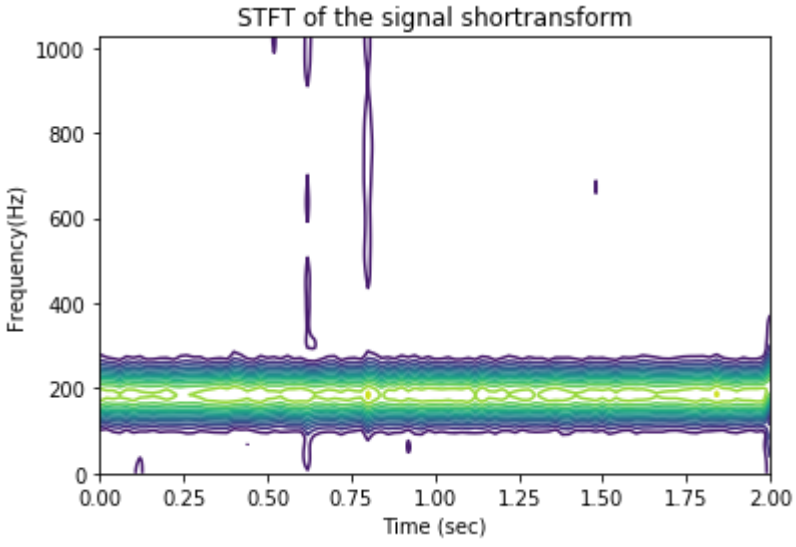
Το ερώτημα αυτό ως προς τη διαδικασία επίλυσης είναι απόλυτα ίδιο με το ερώτημα 2.1. (α). Η γραφική που προκύπτει, λοιπόν, είναι η ακόλουθη.



(β)

Η διαδικασία για την επίλυση του ερωτήματος αυτού είναι ακριβώς αντίστοιχη με εκείνη του ερωτήματος 2.1.β. Το μόνο που διαφέρει είναι ότι αναπαριστούμε το πλάτος |STFT(τ, f)| με τη εντολή plt.contour(newt, newf, abs(newshortransform), 15).

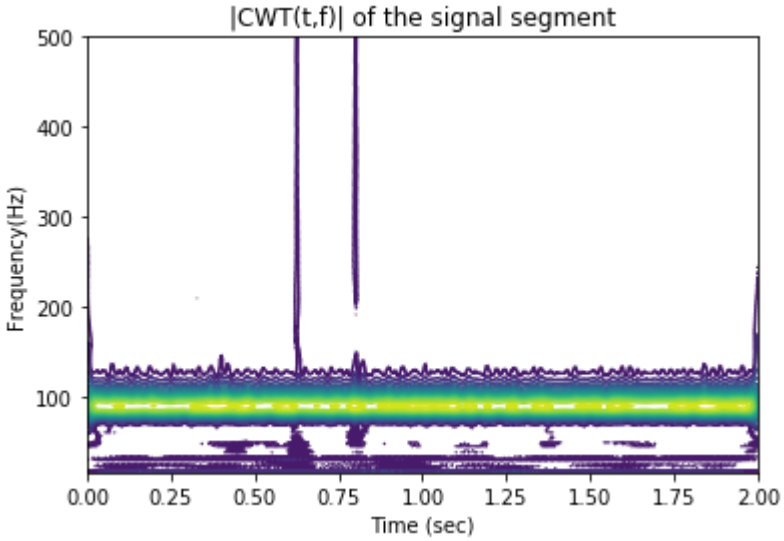
Έτσι λαμβάνουμε την κάτωθι γραφική:



(γ)

Η διαδικασία για την επίλυση του ερωτήματος αυτού είναι ακριβώς αντίστοιχη με εκείνη του ερωτήματος 2.1.γ. Το μόνο που διαφέρει είναι ότι αναπαριστούμε το πλάτος |STFT(τ, f)| με την εντολή plt.contour(newt, newf, np.abs(coefs), 15).

Η γραφική διαμορφώνεται ως εξής:



(δ)

Συγκρίνοντας τις δυο γραφικές καταλήγουμε στο ότι οι αποκλίσεις, διαφορετικά, παράσιτα των DT-CWT σε σχέση με των STFT είναι σαφέστατα εμφανέστερες.

Μέρος 3ο - *Χαρακτηριστικά Βραχέος Χρόνου Σημάτων Φωνής και Μουσικής (Ενέργεια και Ρυθμός Εναλλαγής Προσήμου)*

Ερώτημα 3.1.

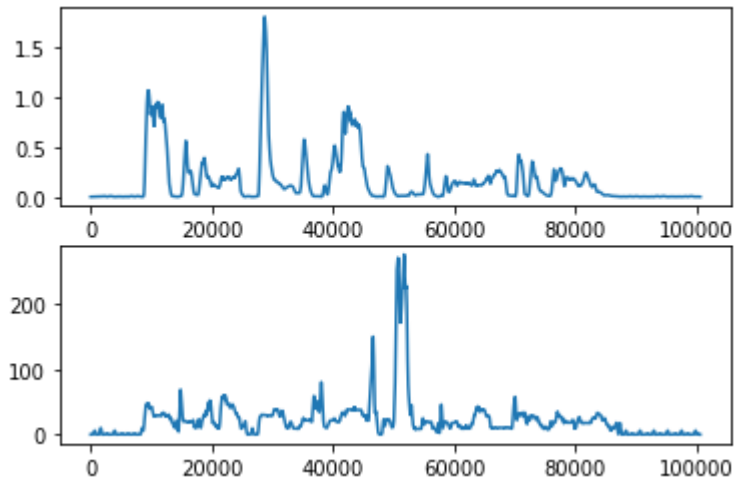
Για την επίλυση του ερωτήματος αυτού συνθέτουμε τη συνάρτηση short_time_energy_and_crossing_rate_of(signal). Στόχος είναι να καταφέρουμε να υπολογίσουμε με τις εντολές της Python, την ενέργεια βραχέος χρόνου E_n καθώς και τον ρυθμό εναλλαγής προσήμου Z_n. Για το σκοπό αυτό, δημιουργούμε τον πίνακα window_length με στοιχεία τους αριθμούς από το 20 έως και 30. Έπειτα, δημιουργούμε και τον πίνακα hamming_window που αποτελεί στην πραγματικότητα το Hamming παράθυρο μήκους 20 – 30 ns, αφού F_S = 16 kHz και $16*10^3*20*10^{-3}$ = 20*16. Υψώνουμε το σήμα στο τετράγωνο με την εντολή signal**2. Η συνέλιξη στο διακριτό χρόνο δίνεται από την εντολή np.convolve, η οποία λαμβάνει δύο ορίσματα,

που είναι, φυσικά, τα προς συνέλιξη σήματα. Προκειμένου να υπολογίσουμε τη διαφορά sgn[x(m)] − sgn[x(m − 1)], πρέπει να φροντίσουμε οι πίνακες των δύο σημάτων να έχουν το ίδιο μήκος. Αυτό σημειώνεται με τις εντολές:

shift_right_logical=np.zeros(1)

sign_of_number=np.concatenate((signal,shift_right_logical))
sign_of_previous_number=np.concatenate((shift_right_logical,signal)).

Τέλος, η συνάρτηση αυτή, κάνοντας load() το αρχείο “speech utterance.wav” επιστρέφει τους πίνακες με τις τιμές της ενέργειας και τις τιμές του ρυθμού εναλλαγής προσήμου, καθώς και τις γραφικές παραστάσεις αυτών οι οποίες αναπαριστώνται, αντίστοιχα, παρακάτω:



Από τα παραπάνω διαγράμματα καθίσταται σαφές ότι για μεγαλύτερο μήκος παραθύρου οι ενέργειες βραχέος χρόνου υπολείπονται σε ακρίβεια. Όσον αφορά τη δυνατότητά μας να ξεχωρίσουμε φωνή (έμφωνους) από σιωπή (άφωνους ήχους), δεδομένου ότι οι έμφωνοι χαρακτήρες είναι περιοδικοί και μεγάλου πλάτους σε αντίθεση με τους άφωνους που είναι απεριοδικοί-δεν μπορούμε να αποφανθούμε με ακρίβεια. Ο λευκός θόρυβος Gauss καθιστά τα σημεία σιγής να μειώσουν το Zero Crossing Rate τους το οποίο ειδάλλως είναι ιδιαίτερα υψηλό διατηρώντας παράλληλα τη χαμηλή ενέργειά τους. Το γεγονός αυτό σε συνδυασμό με το χαμηλό ρυθμό εναλλαγής προσήμου και τη μεγάλη ενέργεια βραχέος χρόνου των έμφωνων δεν μας επιτρέπει να εξάγουμε βέβαιο συμπέρασμα.

Ερώτημα 3.2

Εντελώς αντίστοιχα με το προηγούμενο ερώτημα, κάνουμε load() το σήμα μουσικής “music.wav” και καλούμε τη συνάρτηση short_time_energy_and_crossing_rate_of(signal). Με αυτόν τον τρόπο, λαμβάνουμε και πάλι τους νέους πίνακες με τις τιμές της ενέργειας και τις τιμές του ρυθμού εναλλαγής προσήμου, καθώς και τις γραφικές παραστάσεις αυτών οι οποίες αναπαριστώνται, αντίστοιχα, παρακάτω: