Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Ψηφιακή Επεξεργασία Σήματος

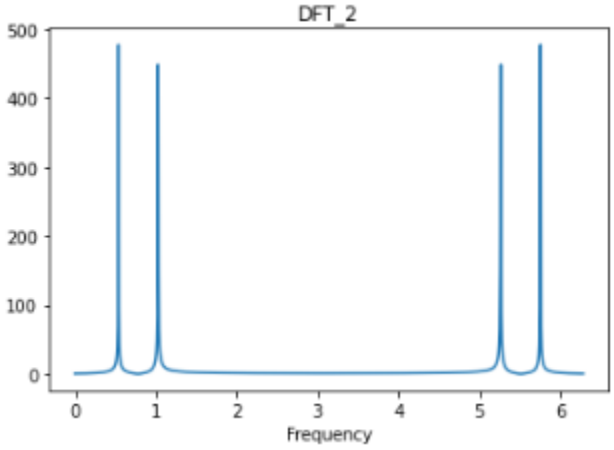
Εξάμηνο 6ο (Εαρινό Εξάμηνο 2020-2021)

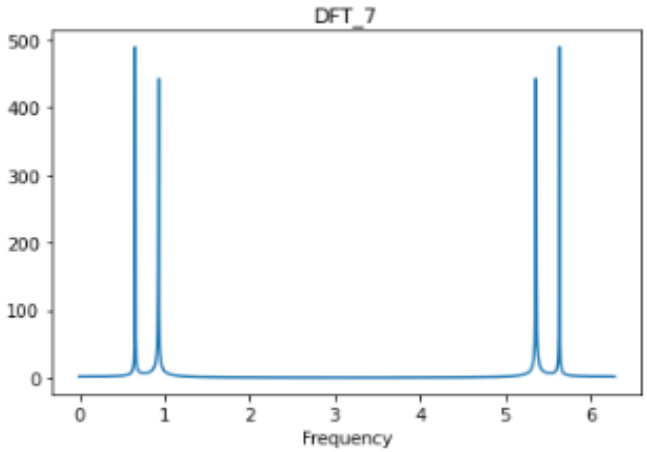
1ο Εργαστηριακό Project Σπανός Νικόλαος – el18822

Άσκηση 1

1.2:

Έχοντας δημιουργήσει τις συναρτήσεις χρησιμοποιώντας την συνάρτηση που φτιάξαμε στον κώδικα, ύστερα δημιουργούμε τις γραφικές παραστάσεις με την συνάρτηση **plot()** της **matplotlib** και παίρνουμε τα εξής αποτελέσματα:





1.4:

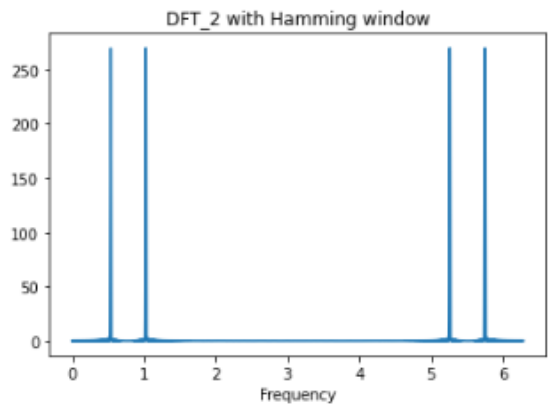
Έχοντας εκτελέσει την διαδικασία για όλα τα ψηφία της ακολουθίας και έχοντας αποθηκεύσει τα αποτελέσματα σε δύο λεξικά όπως φαίνεται παρακάτω:

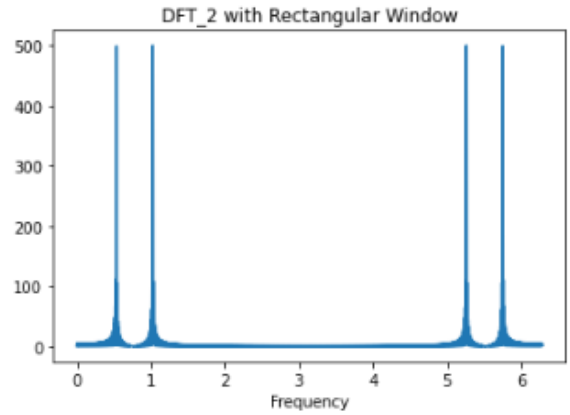
```
#Ορίζουμε τα παράθυρα με πλήθος ψηφίων για το ΑΜ δηλαδη 8700#
window_rec=np.array([1]*1000)
zero_pad=np.array([0]*7700)
window_rec=np.append(window_rec,zero_pad)
window_ham=np.array([])
for i in range(len(window_rec)):
    window_ham=np.append(window_ham,[window_rec[i]*(0.54-0.46*np.cos(2*np.pi*i/(1000)))])

#Δημιουργούμε δύο λεξικά με keys digit_num_(αριθμός ψηφίου) στο οποία αποθηκεύουμε τους DFT
#για την ανάλυση με κάθε παράθυρο
di_rec={}
di_ham={}

for i in range((len(sequence))):
    DFT_Sequence=tone_sequence*window_rec
    DFT_Sequence=np.fft.fft(DFT_Sequence)
    #Τοποθετούμε σε ένα λεξικό τους DFT για κάθε ψηφίο με το τετραγωνικό παράθυρο
    di_rec["digit_num_{}".format(i+1)]=DFT_Sequence
    #DFT_Tone=np.append(DFT_Tone,DFT_Sequence)
    window_rec=np.roll(window_rec,1100)

for i in range((len(sequence))):
    DFT_Sequence=tone_sequence*window_ham
    DFT_Sequence=np.fft.fft(DFT_Sequence)
    di_ham["digit_num_{}".format(i+1)]=DFT_Sequence
    #DFT_Tone=np.append(DFT_Tone,DFT_Sequence)
    window_ham=np.roll(window_ham,1100)
```

επιλέγουμε το έβδομο ψηφίο του sequence μας δηλαδή το 2 σε αυτήν την περίπτωση (03118822).Με την παραθύρωση της ακολουθίας παρατηρούμε για το κάθε παράθυρο τα εξής αποτελέσματα:

```
f=np.linspace(0,2*np.pi,8700)
f1=np.linspace(0,2*np.pi,1000)
plt.plot(f,np.abs(di_ham['digit_num_7']))


plt.xlabel('Frequency')
plt.title('DFT_2 with Hamming window')
```

```
plt.plot(f,np.abs(di_rec['digit_num_7']))
plt.xlabel('Frequency')
plt.title('DFT_2 with Rectangular Window')
```





Παρατηρούμε ότι η παραθύρωση και στις δύο περιπτώσεις είναι αρκετά επιτυχής, καθώς πήραμε κοντινά αποτελέσματα με τον DFT του 2. Βέβαια παρατηρούμε ότι στην ανάλυση με το Hamming παράθυρο έχουμε λιγότερο θόρυβο , αλλά και μικρότερο πλάτος στα peaks , σε σχέση με το τετραγωνικό παράθυρο.

1.5

Δημιουργούμε την λίστα κ σε μορφή λεξικού για να είναι πιο ευανάγνωστη:

```
{'peaks_number_0': array([0.71594238, 1.01477051]), 'peaks_number_1': array([0.5291748 , 0.92138672]), 'peaks_number_2': array
([0.5291748 , 1.01477051]), 'peaks_number_3': array([0.5291748 , 1.12060547]), 'peaks_number_4': array([0.58520508, 0.9213867
2]), 'peaks_number_5': array([0.58520508, 1.01477051]), 'peaks_number_6': array([0.58520508, 1.12060547]), 'peaks_number_7': ar
ray([0.64746094, 0.92138672]), 'peaks_number_8': array([0.64746094, 1.01477051]), 'peaks_number_9': array([0.64746094, 1.120605
47])}
```

Η δημιουργία της έγινε πρακτικά ακολουθώντας παρόμοια διαδικασία με το προηγούμενο ερώτημα με παράθυρο Hamming , αναλύωντας την ακολουθία των ψηφίων [0,1,2,3,4,5,6,7,8,9]. Παρατηρούμε ότι οι συχνότητες απέχουν ελάχιστα από τον ορισμό των ημιτόνων στο πρώτο ερώτημα,άρα έχουμε μια σχετικά καλή προσέγγιση.

1.6

Για να εντόπισουμε το σωστό ψηφίο , χρησιμοποιούμε μια βοηθητική συνάρτηση , η οποία μας επιστρέφει κάθε φορά μια λίστα με τις συχνότητες που προκύπτουν για κάθε ψηφίο. Ύστερα συγκρίνουμε τις συχνότητες αυτές με την λίστα κ και επιλέγουμε την δυάδα που βρίσκεται πλησιέστερα στις τιμές. Η δοκιμή έγινε χρησιμοποιώντας μια υπορουτίνα **make_tone()** για την δημιουργία της ακολουθίας 03118822, στην οποία ορίζουμε τα σήματα στο μήκος της ακολουθίας για να αποφύγουμε τα μηδενικά και παίρνουμε κάθε φορά τις τιμές που χρειαζόμαστε:

```
def make_tone(Sequence):
    tones=np.array([])
    t=np.linspace(0,(1000*len(Sequence))/8192,(1000*len(Sequence)))
    zero_pad=[0]*100
    D1_n=tone_func(0.5346*8192,0.9273*8192,t)
    D2_n=tone_func(0.5346*8192,1.0247*8192,t)
    D3_n=tone_func(0.5346*8192,1.1328*8192,t)
    D4_n=tone_func(0.5906*8192,0.9273*8192,t)
    D5_n=tone_func(0.5906*8192,1.0247*8192,t)
    D6_n=tone_func(0.5906*8192,1.1328*8192,t)
    D7_n=tone_func(0.6535*8192,0.9273*8192,t)
    D8_n=tone_func(0.6535*8192,1.0247*8192,t)
    D9_n=tone_func(0.6535*8192,1.1328*8192,t)
    D0_n=tone_func(0.7217*8192,1.0247*8192,t)
    #Συνάρτηση από την οποία θα παίρνουμε τα τονικά σήματα
    tones_notes=[D0_n,D1_n,D2_n,D3_n,D4_n,D5_n,D6_n,D7_n,D8_n,D9_n]
    tone_sequence=np.array([])
    check=0 #Check για να λήξει και να μην κάνει το τελευταίο zero_pad οταν φτάσει στο τέλευταιο ψηφίο
    a=0 #Index για να παίρνει κάθε φορά τα σωστά σημεία των τόνων
    for i in range(len(Sequence)):
        tone_sequence=np.append(tone_sequence,[tones_notes[Sequence[i]][a:1000+a]])
        check=check+1
        if(check == len(Sequence)):
            break
        a=a+1000
        tone_sequence=np.append(tone_sequence,[zero_pad])
    return tone_sequence
```

Η βοηθητική συνάρτηση **freq_creator():**

```
def freq_creator(signal):
    #Δημιουργούμε ενα τετραγωνικο παραθυρο με μηκος οσο το signal μας
    window_rec=np.array([1]*1000)
    #Ορίζουμε το index που τελειώνει το πρώτο ψηφίο και ξεκινάνε τα μηδενικά
    index=1000
    zero_pad=np.array([0]*(len(signal)-1000))
    window_rec=np.append(window_rec,zero_pad)
    window_temp=window_rec
    di_tones={}
    DFT_Tone=np.array([])
    energy=np.array([])
    CHECK=0
    count=0
    i=0
    while(CHECK!=1):
        CHECK_2=True
        #Οταν το παραθυρο φτάσει στο τελος σημαίνει πως φτάσαμε και στο τελευταίο ψηφίο
        #οπότε το check λήγει το recursion στο επόμενο
        if(window_rec[-1]==1):
            CHECK=CHECK+1
        #Υπολογίζουμε τον DFT για το ψηφίο
        DFT_Sequence=signal*window_rec
        DFT_Sequence=np.fft.fft(DFT_Sequence)
        #και το προσθέτουμε στο λεξικό
        di_tones["tone{}".format(i)]=DFT_Sequence
        #Εχουμε ενα counter για να κρατήσουμε το πλήθος των ψηφίων
        count=count+1
        count_roll=0
        #Το δεύτερο while υπολογίζει πόσα μηδενικά υπάρχουν ενδιαμέσου δυο διαδοχικών ψηφίων
        #και ύστερα ορίζει το index και το πλήθος που πρέπει να μετακινήσουμε το παράθυρο
        while(CHECK_2==True):
            if(index==len(signal)):
                break
            if(signal[index]!=0):
                CHECK_2=False
                break
            count_roll=count_roll+1
            index=index+1
        if(index==len(signal)):
            break
        window_rec=np.roll(window_rec,1000+count_roll)
        index=index+1000
        i=i+1
    peaks_tones={}
    CHECK=0
    i=0
    #Βρίσκουμε τα peaks των DFT και δημιουργούμε ένα λεξικό με τις συχνότητες τους
    for i in range(count):
        peaks,properties=sp.signal.find_peaks((np.abs(di_tones['tone{}'.format(i)])),400)
        peaks_tones["peaks_number_{}".format(i)]=peaks[0:2]
    frequency_tones=peaks_tones
    for i in peaks_tones:
        frequency_tones[i]=frequency_tones[i]*(2*np.pi*(8192/len(signal)))
    W_Frequency=frequency_tones
    for i in peaks_tones:
        W_Frequency[i]=(frequency_tones[i])/8192
    return W_Frequency
```

Ύστερα , η **ttdecode()** χρησιμοποιεί το αποτέλεσμα αυτής της συνάρτησης για να συγκρίνει τα αποτελέσματα με την λίστα κ που δημιουργήσαμε. Παρατίθονται η συνάρτηση και το αποτέλεσμα για την ακολουθία του ΑΜ:

```
def ttdecode(Seq):
    a=freq_creator(Seq)
    #Χρησιμοποιώντας το αποτέλεσμα της βοηθητικής συνάρτησης που ορίσαμε απο πάνω
    #Συγκρίνουμε τις συχνότητες των ψηφίων του σήματος με τις συχνότητες για το κάθε ψηφίο
    #και ύστερα κρατάμε τον αριθμό στον οποίο αντιστοιχεί
    index_ar=np.array([])
    for i in a:
        min_1=a[i][0]
        min_2=a[i][0]
        index_1=0
        index_2=0
        for j in k:
            temp1=abs(k[j][0]-a[i][0])
            temp2=abs(k[j][1]-a[i][1])
            if(temp1<min_1):
                min_1=temp1
                index_1=j
            if(temp2<min_2):
                min_2=temp2
                index_2=j
        for j in k:
            if(k[j][0]==k[index_1][0] and k[j][1]==k[index_2][1]):
                min_place=j
                index_ar=np.append(index_ar,[j[-1]])
    return index_ar
b=ttdecode(Seq)
print(b)
```

```
['0' '3' '1' '1' '8' '8' '2' '2']
```

1.7

Παρακάτω βλέπουμε τα αποτέλεσματα για την ανάλυση των easySig και hardSig με την **ttdecode()**:

```
sig_1=np.load("easySig.npy")
sig_2=np.load("hardSig.npy")

a=ttdecode(sig_1)
print(a)
b=ttdecode(sig_2)
print(b)
```

```
['8' '1' '0' '3' '9' '6' '3' '8']
['4' '8' '1' '9' '2' '1' '5' '3' '6' '3']
```

Άσκηση 2

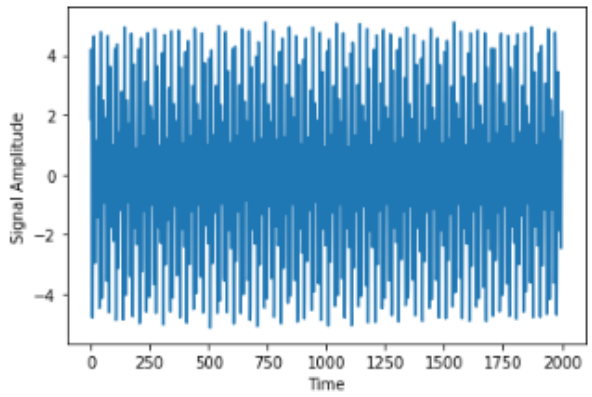
2.1:

(α)

Ορίζουμε την συνάρτηση μας χρησιμοποιώντας τις συναρτήσεις της numpy και παίρνουμε το εξής αποτέλεσμα:

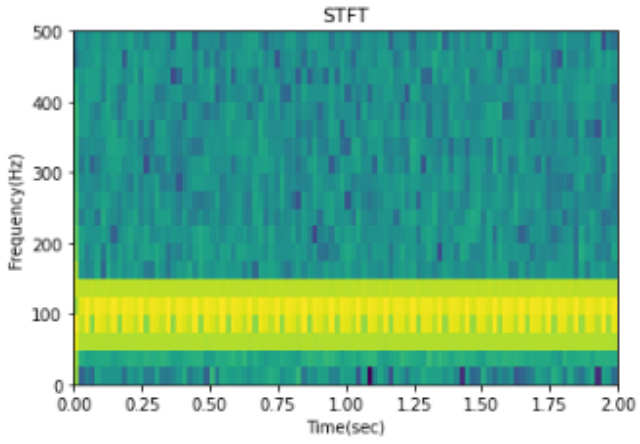
```
#PART_2#
t=np.linspace(0,2000,2000)
x_n=2*np.cos(2*np.pi*70*t/1000)+3*np.sin(2*np.pi*100*t/1000)+0.1*np.random.normal(size=(2000,))
plt.plot(t,x_n)
plt.xlabel('Time')
plt.ylabel('Signal Amplitude')
```

```
Text(0, 0.5, 'Signal Amplitude')
```



(β)

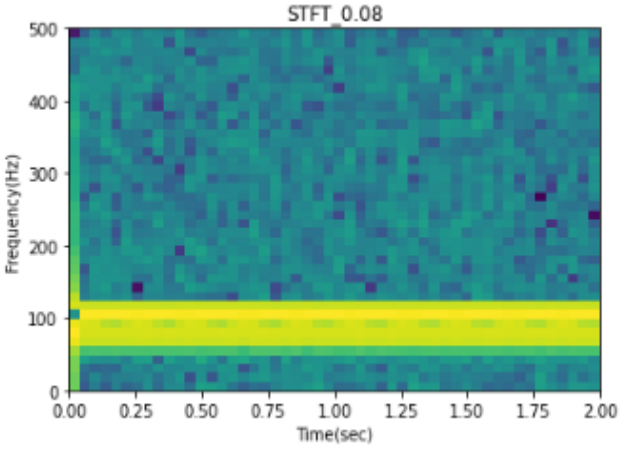
Υπολογίζουμε τον STFT του σήματος με παράθυρο 0.04 sec και επικάλυψη το μισό του μήκους του παραθύρου.Τα linspace δημιουργήθηκαν βάσει των αποτελεσμάτων που πήραμε από το shape του STFT:

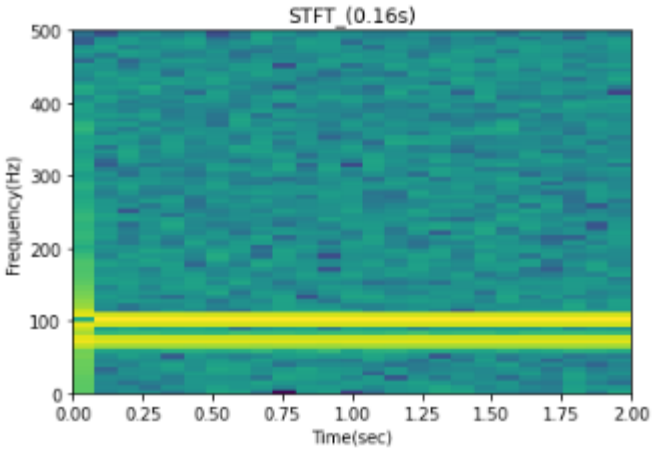


Μιας και το παράθυρο είναι μικρό σε μήκος παρατηρούμε ότι έχουμε αρκετά καλή ανάλυση στον χρόνο αλλά όχι τόσο στην συχνότητα.

(γ)

Παρακάτω βλέπουμε τα αποτελέσματα για παράθυρα ίδιας μορφής αλλά μήκους 0.08 και 0.16 δευτερολέπτων:

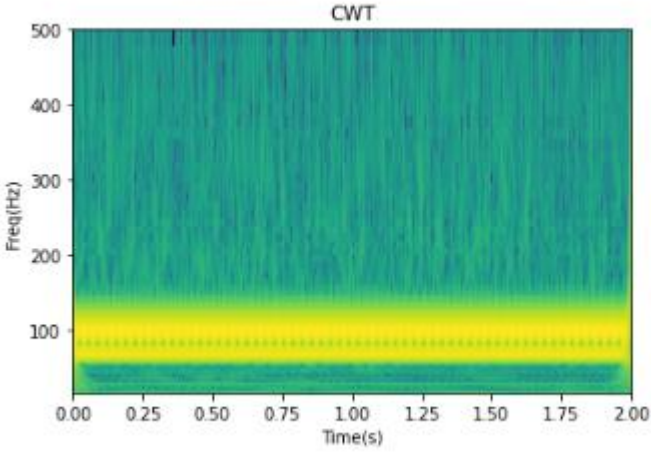


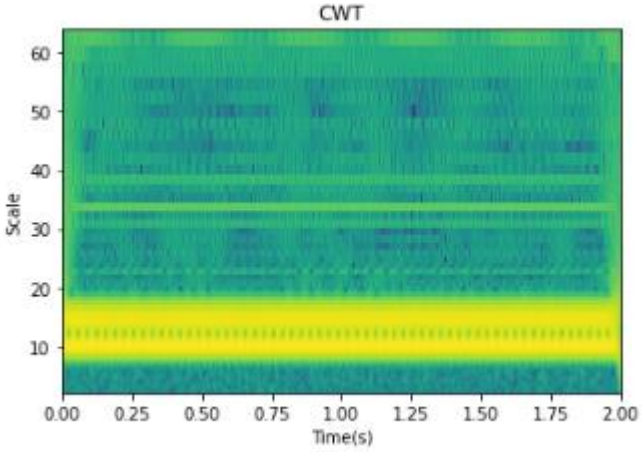


Παρατηρούμε ότι όσο αυξάνουμε το μήκος του παραθύρου η ανάλυση στο πεδίο της συχνότητας βελτιώνεται αρκετά, τόσο που για μήκος 0.16 βλέπουμε σχετικά καλή προσέγγιση για τις συχνότητες των ημιτονοειδών σημάτων μας, ενώ η ανάλυση στον χρόνο χειροτερεύει σημαντικά.

(δ,ε)

Εκτελούμε discretized CWT για το σήμα μας και παίρνουμε τα εξής αποτελέσματα:





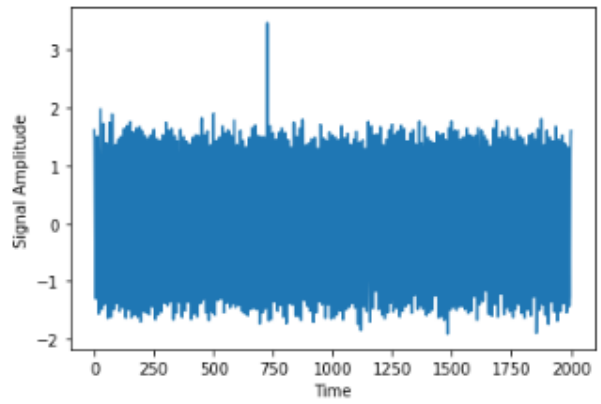
Παρατηρούμε μια σχετικά πιο ισορροπημένη ανάλυση και στα δύο πεδία με τον CWT, καθώς ο CWT δεν είναι uniform και μπορεί να μας παρέχει πληροφορίες ταυτόχρονα και για την ανάλυση στον χρόνο και για την ανάλυση στην συχνότητα.

2.2:

Καθώς δειγματοληπτούμε τα δ(t) ως δ[n] , απλώς προσθέτουμε και αφαιρούμε αντίστοιχα στα κατάλληλα σημεία την τιμή 1.7 και παράγουμε την γραφική παράσταση:

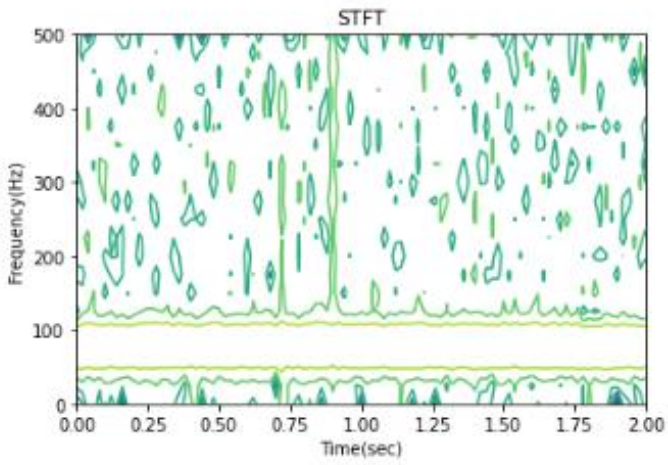
```
#2.2#
t=np.linspace(0,2000,2000)
x_n=1.5*np.cos(2*np.pi*80*t/1000)+0.15*np.random.normal(size=2000,)
x_n[725]=x_n[725]+1.7
x_n[900]=x_n[900]-1.7
plt.plot(t,x_n)
plt.xlabel("Time")
plt.ylabel("Signal Amplitude")
```

```
Text(0, 0.5, 'Signal Amplitude')
```



(β)

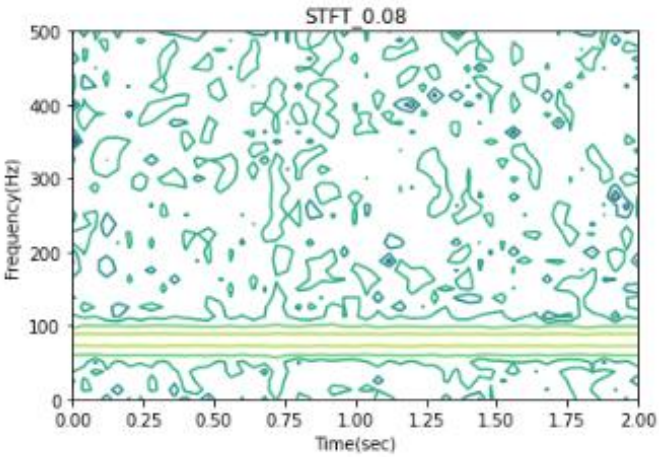
Υπολογίζουμε τον STFT και παίρνουμε το εξής αποτέλεσμα με την **contour()**:

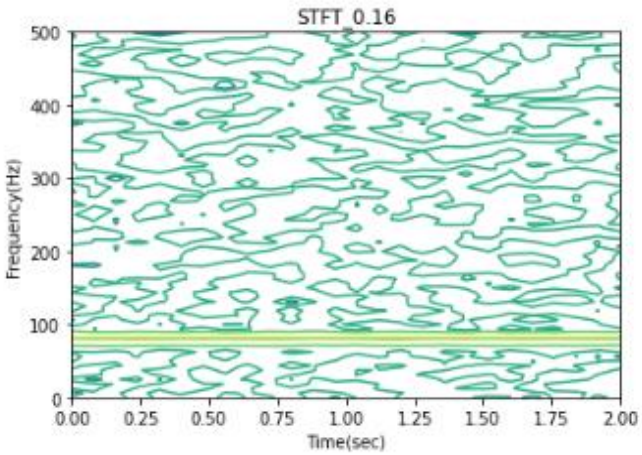


Πατατηρούμε δύο απότομες κατακόρυφες γραμμές στα σημεία τα οποία έχουμε απότομη μεταβολή.Άρα, μπορούμε εύκολα να εντοπίσουμε τις απότομες μεταβολές,παρατηρώντας τον STFT του σήματος.

(γ)

Υπολογίζουμε τον STFT ξανά με παράθυρα μήκους 0.08 και 0.16 δευτερολέπτων και παίρνουμε τα εξής αποτελέσματα:

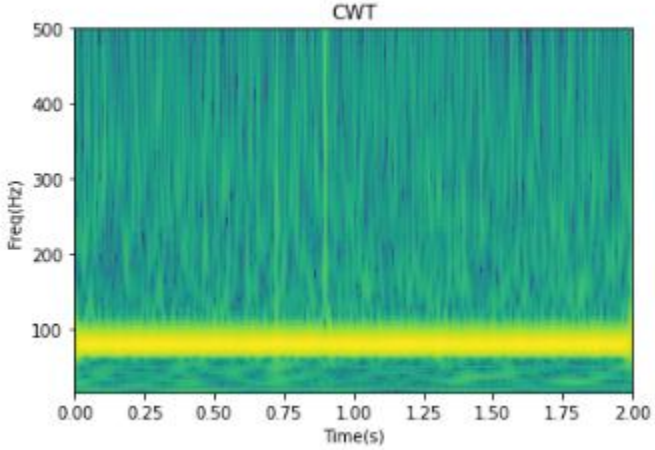


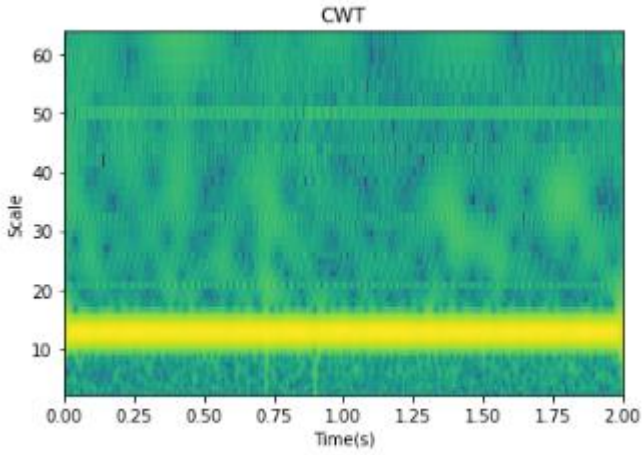


Παρατηρούμε ότι αν και στα στο παράθυρο 0.08 δευτερολέπτων η μεταβολή είναι ελάχιστα εμφανής, γενικά όσο αυξάνουμε το μήκος του παραθυρού δυσκολευόμαστε αρκετά να διακρίνουμε από την ανάλυση μας τις απότομες μεταβολές.

(δ)

Υπολογίζουμε το discretized CWT και παίρνουμε τα εξής αποτελέσματα:





Στην ανάλυση της συχνότητας παρατηρούμε δύο έντονες κατακόρυφες καμπύλες που βρίσκονται στα σημεία των έντονων μεταβολών του σήματος. Στην ανάλυση της κλίμακας αν και αδύναμες, πατατηρούμε στο κάτω μέρος άλλες δύο έντονες καμπύλες στα σημεία σημεία της μεταβολής.

<u>Άσκηση 3</u>

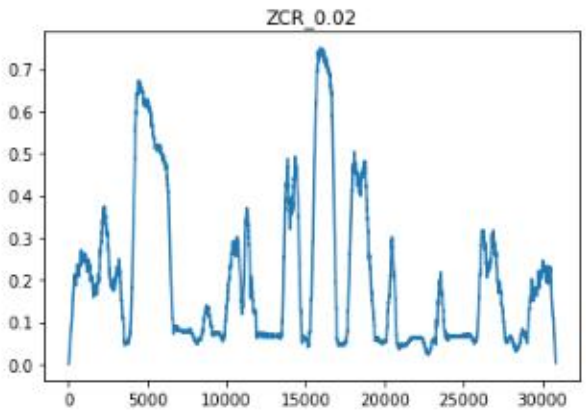
3.1:

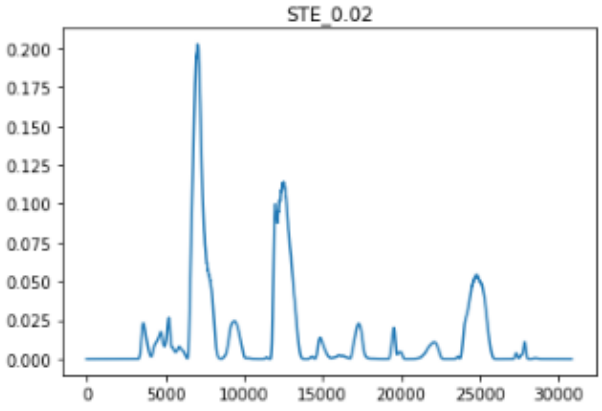
Βάσει των τύπων για την ενέργεια βραχέος χρόνου και τον ρυθμό εναλλαγή προσήμου , δημιουργούμε την εξής συνάρτηση με όρισμα το σήμα , τον χρόνο του παραθύρου Hamming και την συχνότητα δειγματοληψίας, η οποία μας επιστρέφει και τα δύο διανύσματα:

```
def En_Zcr_calc(sr,win_length,sig):
    win_ham_1=np.hamming(int(sr*win_length))
    window_rec=np.array([1]*(int(sr*win_length)))
    print(len(window_rec))
    zero_pad=np.array([0]*(len(sig)-(int(sr*win_length))))
    win_rec=np.append(window_rec,zero_pad)
    win_rec=window_rec*(1/(2*(int(sr*win_length))))
    sig_en=(np.abs(sig))**2
    En=np.array([])
    En=np.convolve(sig_en,win_ham_1)
    sig_zcr_1=np.sign(sig)
    sig_zcr_2=np.sign(np.roll(sig,1))
    sig_zcr=np.abs(sig_zcr_1-sig_zcr_2)
    Zn=np.array([])
    Zn=np.convolve(sig_zcr,win_rec)
    return En,Zn
```

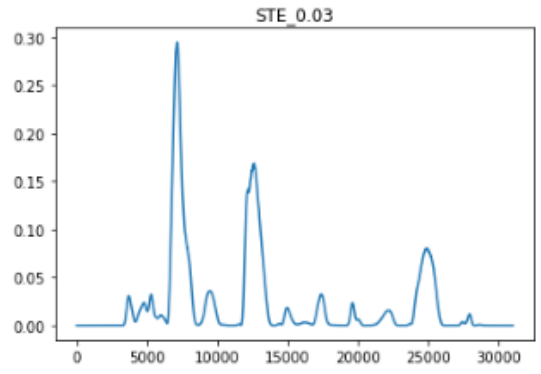
Εκτελούμε την αρχική ανάλυση για το σήμα “speech_utterance.wav”.

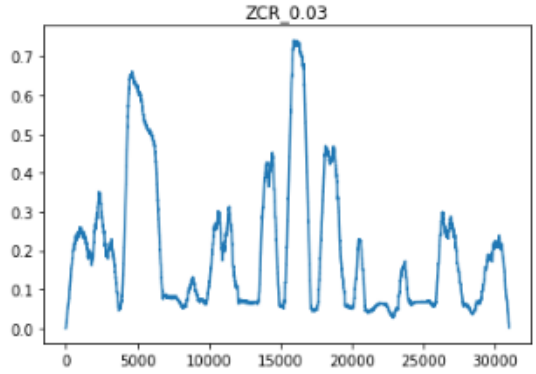
Για διάρκεια 0.02s παίρνουμε τα εξής αποτελέσματα:





Τρέχοντας για διάρκεια 0.03 παρατηρούμε τα εξής:



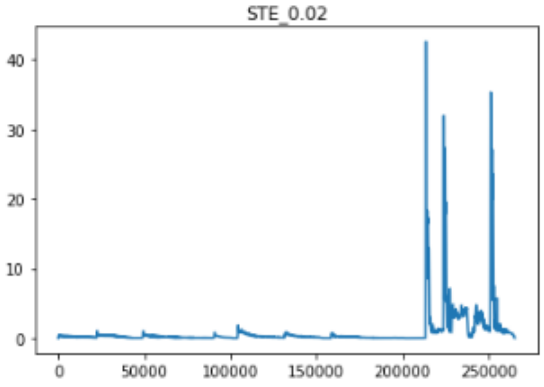


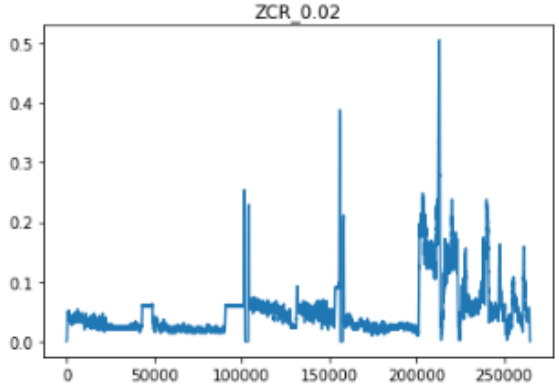
Πατατηρούμε πως όσο αυξάνομαι το μήκος του παραθύρου , οι γραφικές εξομαλύνονται και εξαλείφεται ο θόρυβος.

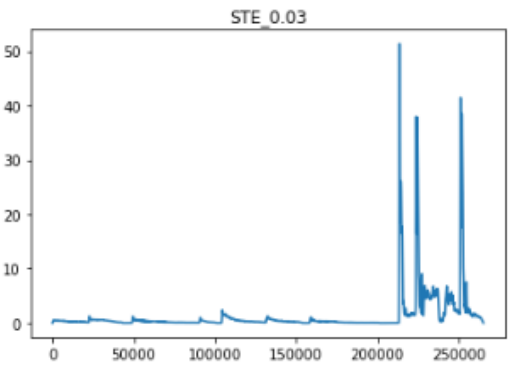
Από τις γραφικές μπορούμε να κατανοήσουμε αντιστοίχως αν ένας ήχος είναι έμφωνος ή αφωνος.Οι έμφωνοι ήχοι διαθέτουν μεγάλες τιμές και peaks στην ενέργεια βραχέος χρόνου , ενώ οι άφωνοι ήχοι διαθέτουν peaks στην γραφική του ZCR. Μπορούμε να παρατηρήσουμε , κιόλας, πως οι γραφικές ειναι αντιστρόφως ανάλογες, δηλαδή ένα peak στην γραφική του βραχέος χρόνου έχει πολύ χαμηλή τιμή στην γραφική του ZCR. Με αυτόν τον τρόπο μπορούμε να διαχωρίσουμε τους έμφωνους από τους άφωνους ήχους σε ένα σήμα.

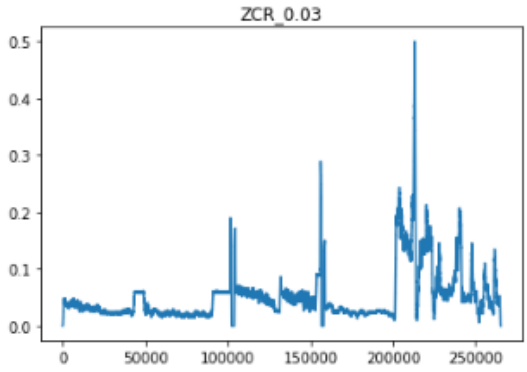
3.2

Επαναλαμβάνουμε την ίδια ακριβώς διαδικασία για το αρχείο “music.wav” και παίρνουμε τα εξής αποτελέσματα:







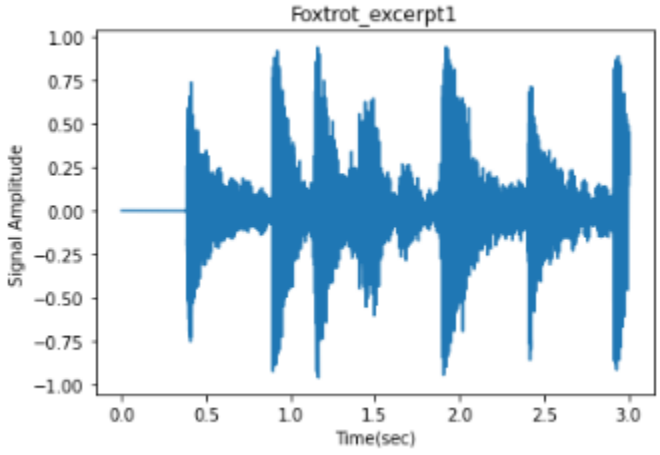


Παρατηρούμε πως τα χαμηλά έντασης όργανα και οι συνθετικές νότες στην αρχή του τραγουδιού δεν διαθέτουν εμφανή short_time ενέργεια, αλλά παρουσιάζουν μεγαλύτερη τιμή και θόρυβο στην γραφική του ZCR. Όμως, στο τέλος του μουσικού κομματιού , όταν ξεκινάνε τα λόγια και οι νότες από τα οργάνα , παρατηρούμε ραγδαία αύξηση στην short time ενεργεια.Άρα, τα όργανα και η φωνή διαθέτουν μεγαλύτερη short_time ενέργεια από τις synth νότες.

Άσκηση 4

4.1:

Φορτώνουμε το σήμα και το πλοττάρουμε:



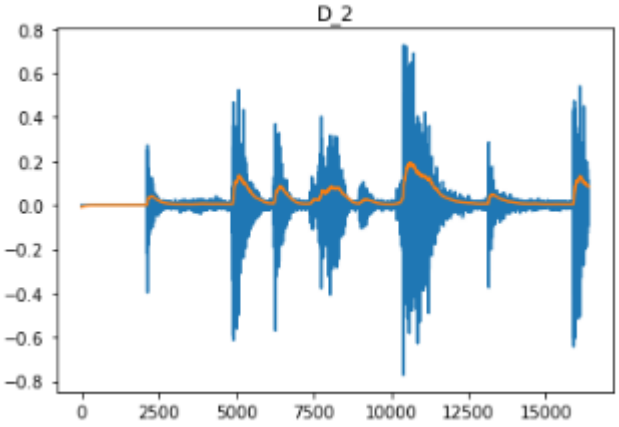
Εποπτικά παρατηρούμε ότι ο κάθε παλμός που μπορούμε να διακρίνουμε επαναλαμβάνεται κάθε 0.5 δευτερόλεπτα.

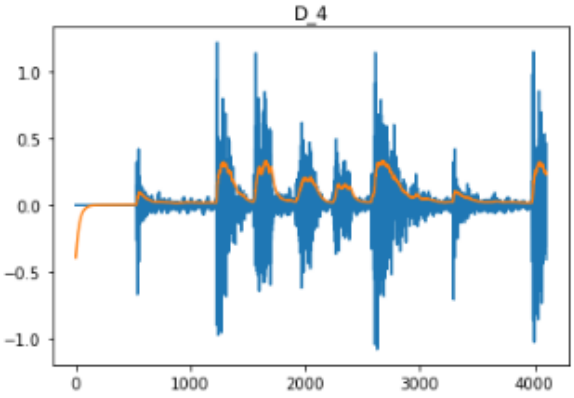
Δημιουργούμε την συνάρτηση η οποία θα μας δώσει την περιβάλλουσα του σήματος το οποίο δίνουμε σαν είσοδο:

```
def env_curve(sig,level,a):
    abs_sig=np.abs(sig)
    temp=np.array([])
    temp=np.append(temp,(1-a*(2**level))*sig[-1]+a*(2**level)*abs_sig[0])
    for i in range(1,len(sig)):
            temp=np.append(temp,(1-a*(2**level))*temp[i-1]+a*(2**level)*abs_sig[i])
    mean=np.mean(sig)
    temp=temp-mean
    return temp
```

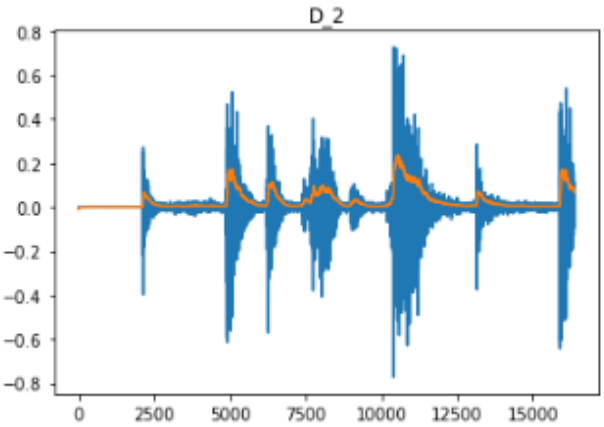
Όπου level το επίπεδο ανάλυσης και α ο συντελεστής α0.

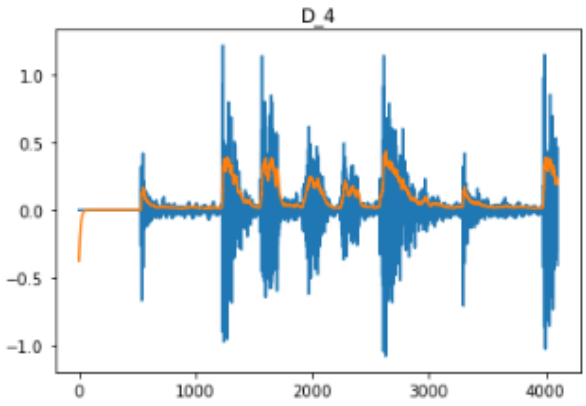
Παίρνουμε τα εξής αποτελέσματα για τα σήματα D2 και D4:





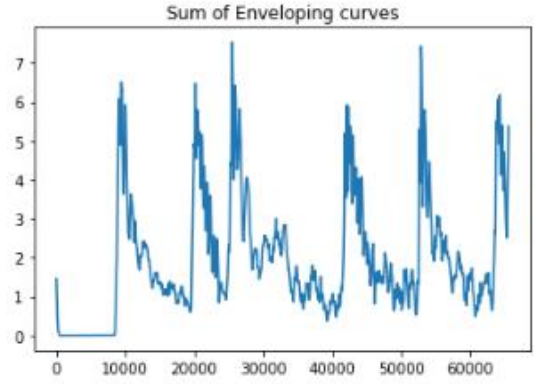
Αλλάζοντας για α=0.005, παίρνουμε τα εξής αποτελέσματα:





Βλέπουμε πως όσο αυξάνουμε το α , χάνεται η εξομάλυνση του σήματος στην περιβάλλουσα,καθώς θα έχει παραπάνω συνεισφορά το απόλυτο σήματος.

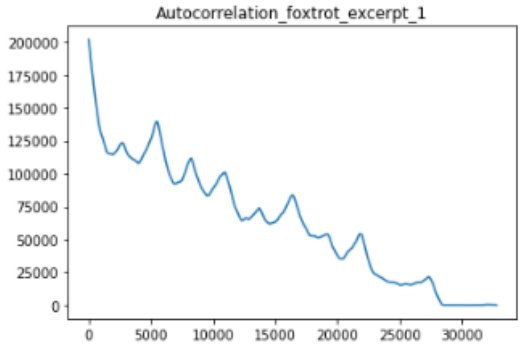
Το άθροισμα προκύπτει ως εξής:



4.5

Εκτελούμε φιλτράρισμα με φίλτρο Gauss , βάσει της διαδικασίας **scipy.ndimage.gaussian_filter1d()** για να ομαλοποιήσουμε την γραφική της αυτοσυσχέτισης του αθροίσματος.

Εκτελώντας την ανάλυση μας για το πρώτο σήμα, παίρνουμε το εξής αποτέλεσμα για την αυτοσυσχέτιση του σήματος:

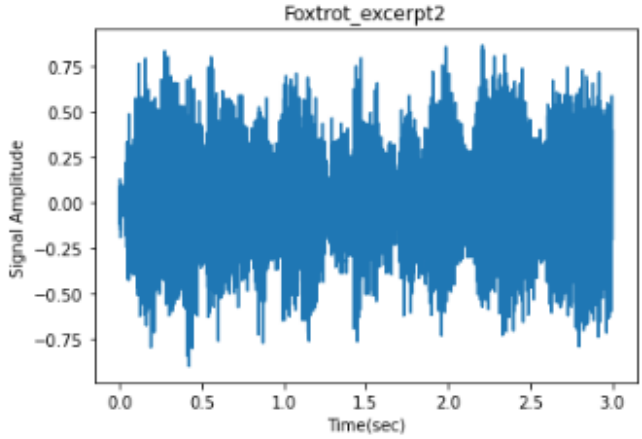


Παρατηρούμε ότι η αυτοσυσχέτιση ξεκινάει από μεγάλες τιμές και σιγά σιγά φθίνει , ενώ παρουσιάζει αρκετά τοπικά ακρότατα που δηλώνουν ότι σε εκείνο το δείγμα παρουσιάζεται περιοδικότητα.

Από την ανάλυση των peaks ,όπου επιλέγουμε το BPM ανάμεσα στα 60 και 200 με το μεγαλύτερο πλάτος , προκύπτει πως το BPM πλησιάζει τα 121 BPM.

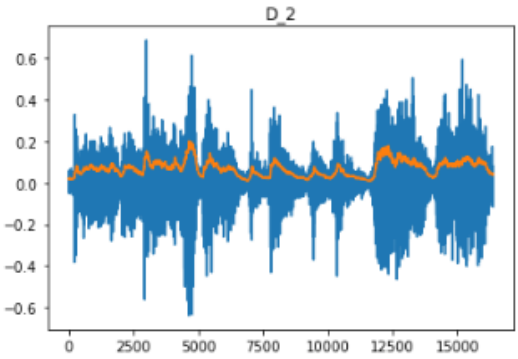
Εκτελούμε την ίδια διαδικασία και για τα άλλα δύο σήματα. Οι αναλύσεις των περιβαλλουσών έγιναν με α=0.005.

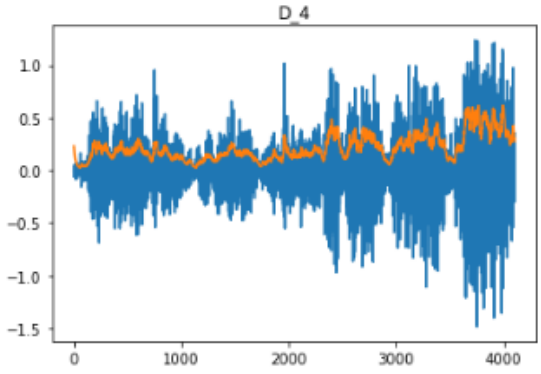
Δεύτερο σήμα:



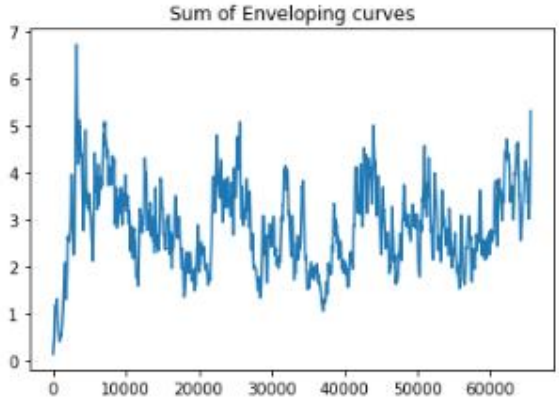
Παρατηρούμε ότι τα peaks του σήματος δεν συμβαίνουν αρκετά τακτικά ώστε να έχουμε ένα σίγουρο αποτέλεσμα, αλλα ένα μέσο διάστημα που παρατηρούμε αρκετά συχνά είναι στα 0.4-0.45 δευτερόλεπτα.

Περιβάλλουσες και σήματα D_2 και D_4:



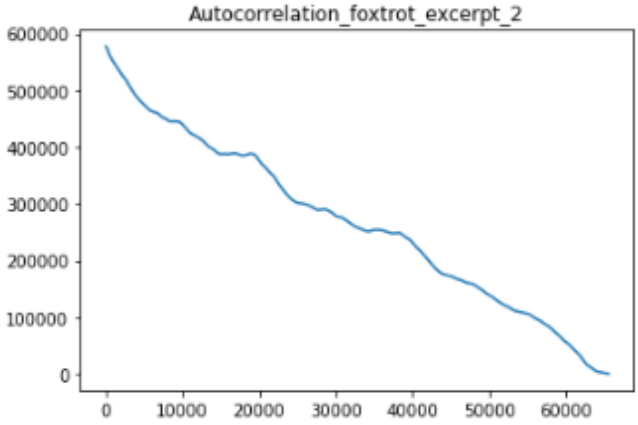


Αθροισμα περιβαλλουσων:



Αυτοσυσχέτιση:

Φιλτράρουμε την αυτοσυσχέτιση καθώς ήταν αρκετά τραχύς για να βγάλουμε μια καλύτερη προσέγγιση, με την διαδικασία **scipy.ndimage.gaussian_filter1d():**

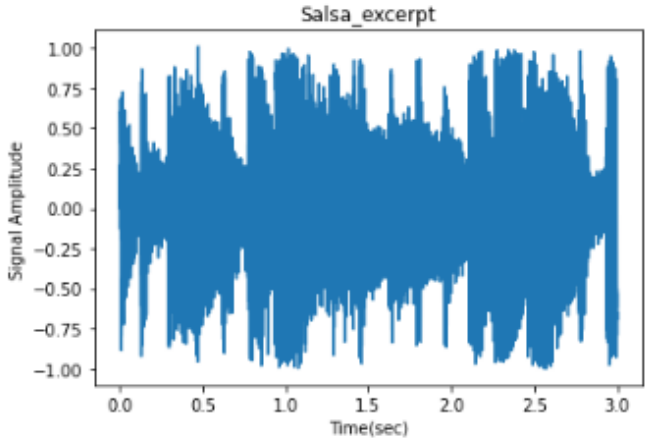


Η αυτοσυσχέτιση εδώ είναι αρκετά πιο ομαλή από το πρώτο σήμα , παρ'ολ'αυτά παρατηρούμε ότι συνεχίζει να παρουσιάζει peaks στα σημεία περιοδικότητας.

Βάσει της ανάλυσης μας, προκύπτει πως το BPM πλησιάζει τα 149 BPM.

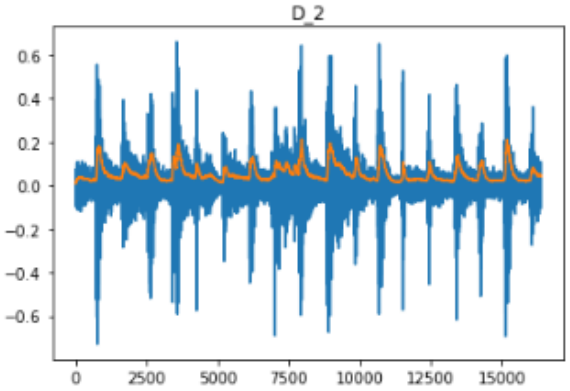
Τρίτο σήμα:

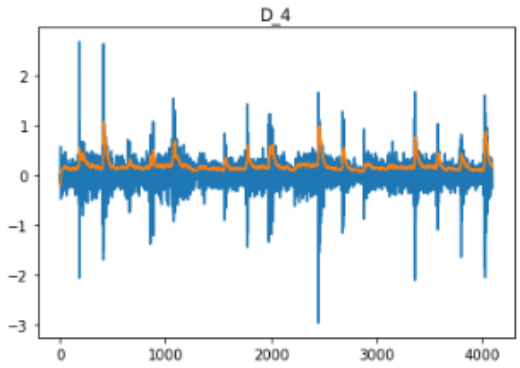
Επιλέγουμε το διάστημα από το πρώτο δευτερόλεπτα μέχρι το τέταρτο έτσι ώστε να αποφύγουμε το κενό στην αρχή του τραγουδιού:



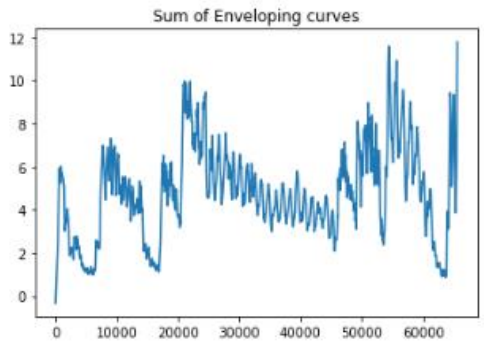
Παρατηρούμε ότι τα peaks έρχονται αρκετά γρήγορα το ένα μετά το άλλο, περίπου στα 0.3 δευτερόλεπτα μεταξύ τους.

<u>Περιβάλλουσες και σήματα D_2 και D_4:</u>

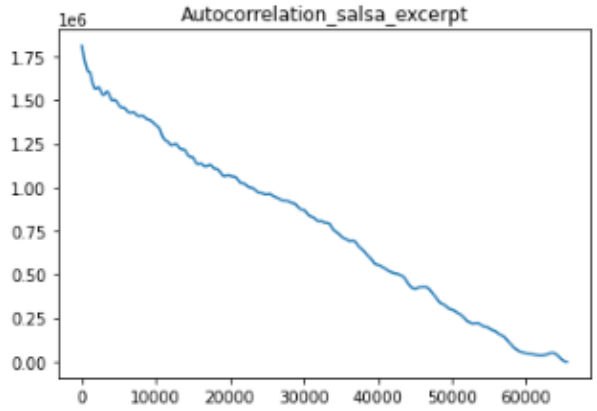




<u>Αθροισμα περιβαλλουσων</u>:



Αυτοσυσχέτιση:



Από την ανάλυση μας προκύπτει πως το BPM είναι κοντά στα 191 BPM.

Παρατηρούμε πως με αυτήν την διαδικασία μπορούμε μέσω του BPM του κάθε τραγουδιού να διακρίνουμε με ευκολία εάν πρόκεται για ένα τραγούδι με γρήγορα χορευτικό ρυθμό ή αργό. Όπως, είδαμε , τα σήματα foxtrot έχουν μεσαία BPM , ενώ το σήμα για τον χορό salsa , που εμφανώς είναι ένα αρκετά γρήγορο χορευτικό είδος, πλησιάζει τα υψηλά BPM.