In [1]:

```
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
import librosa.output as lb
import librosa as lb
import librosa
import pywt
import scipy as sc
import scipy as sp
import time
import math
```

In [2]:

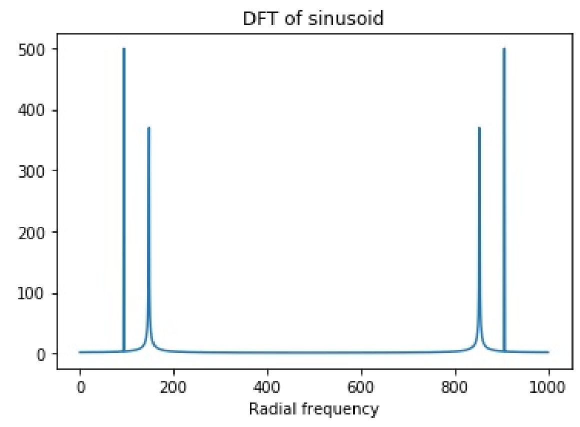
```
# 1.1

column = np.array([0.9273, 1.0247, 1.1328])
row = np.array([0.5346, 0.5906, 0.6535, 0.7217])
digit_tones = np.empty(10, dtype = object)

end = 1000
n = np.arange(1, end + 1, end / 1000) #no 0 in digit_sounds
digit_tones[0] = np.sin(0.7217*n) + np.sin(1.0247*n)

length = len(row) - 1
for i in range(length) :
    a = row[i]
    for j in range(len(column)) :
        b = column[j]
        digit_tones[length * i + j + 1] = np.sin(a*n) + np.sin(b*n)


sd.play(digit_tones[9], 8192)
time.sleep(0.2) #In order to distinguish the digit tones
```

In [3]:

```
# 1.2

get_ipython().run_line_magic('matplotlib', 'inline')
y = np.fft.fft(digit_tones[4])
f = np.linspace(0,1000,1000)
plt.plot(f,np.abs(y))
plt.xlabel('Radial frequency')
plt.title('DFT of sinusoid')
```

Out[3]:

```
Text(0.5, 1.0, 'DFT of sinusoid')
```

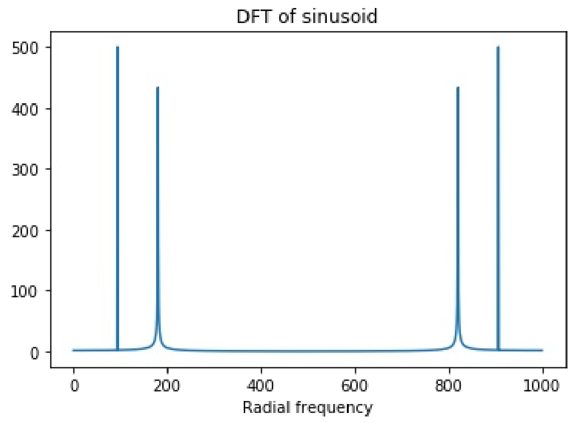


In [4]:

```
get_ipython().run_line_magic('matplotlib', 'inline')
y = np.fft.fft(digit_tones[6])
f = np.linspace(0,1000,1000)
plt.plot(f,np.abs(y))
plt.xlabel('Radial frequency')
plt.title('DFT of sinusoid')
```

Out[4]:

```
Text(0.5, 1.0, 'DFT of sinusoid')
```



In [5]:

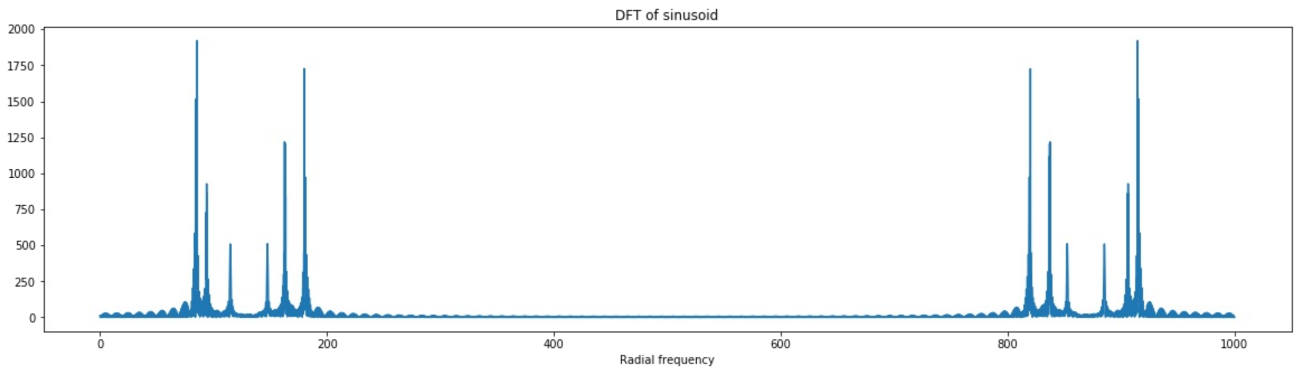
```
# 1.3

# A.M. sum = 03117057 + 03117176 = 06234233

tone_list = [0, 6, 2, 3, 4, 2, 3, 3]
tone = np.empty(0)
for i in tone_list:
    tone = np.concatenate((tone, digit_tones[i]))
    tone = np.concatenate(((tone, np.zeros(100))))

sd.play(tone, 8192)
librosa.output.write_wav("tone_sequence.wav", tone, 8192)
```

In [6]:

```
# 1.4

get_ipython().run_line_magic('matplotlib', 'inline')
f = np.linspace(0,1000,8800)
y = np.fft.fft(tone)
plt.figure(figsize=(20, 5))
plt.plot(f,np.abs(y))
plt.xlabel('Radial frequency')
plt.title('DFT of sinusoid')
```

Out[6]:

```
Text(0.5, 1.0, 'DFT of sinusoid')
```

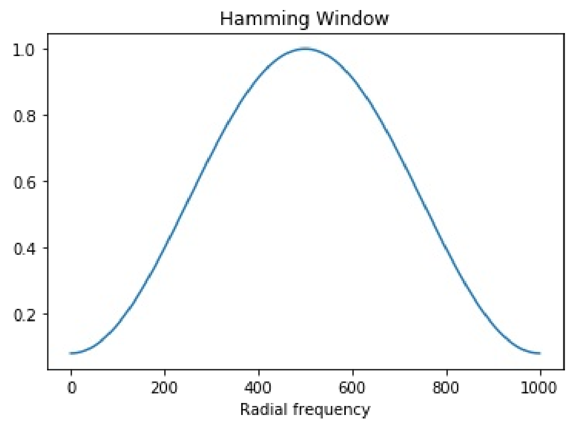


In [7]:

```
get_ipython().run_line_magic('matplotlib', 'inline')
hamming = np.hamming(1000)
f = np.linspace(0,1000,1000)
plt.plot(f,np.abs(hamming))
plt.xlabel('Radial frequency')
plt.title('Hamming Window')
```

Out[7]:

```
Text(0.5, 1.0, 'Hamming Window')
```

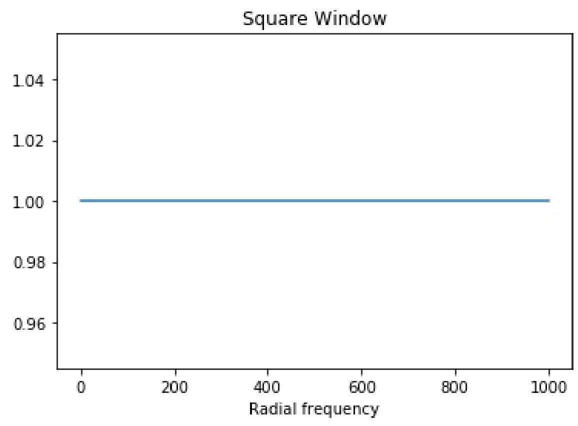


In [8]:

```
from scipy import signal
t = np.linspace(0,1000,1000)
get_ipython().run_line_magic('matplotlib', 'inline')
Un = sp.signal.square(2*np.pi*t,1)
plt.plot(t,Un)
plt.xlabel('Radial frequency')
plt.title('Square Window')
```

Out[8]:

```
Text(0.5, 1.0, 'Square Window')
```

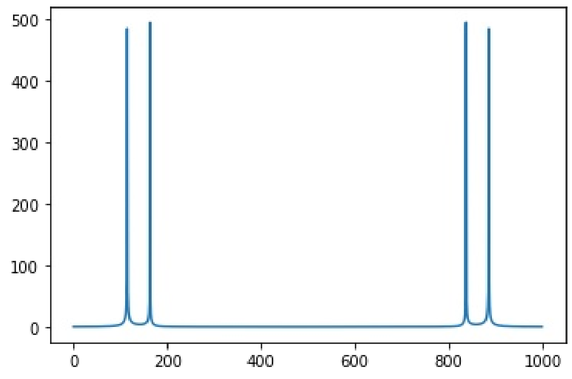


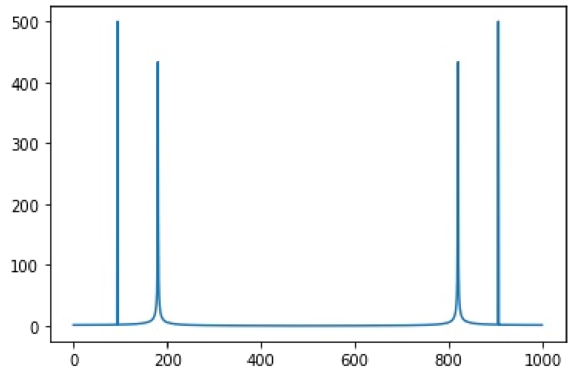
In [9]:

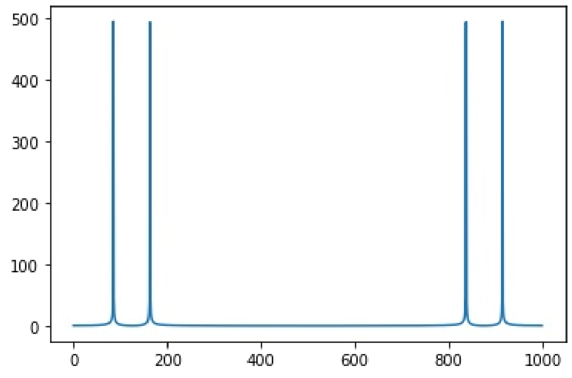
```
# Hamming window
hamm_win = np.hamming(1000)

# Rectangular window
rect_win = np.ones(1000)

# Windows Function
def fourier_window_signal (sig, zeros_between, wind_fun) :
    len_window = np.size(wind_fun)
    len_sig = np.size(sig)

    windows = np.array([sig[i:(i+len_window)] * wind_fun[:] for i in range(0, len_sig, len_window + zeros_between
)])

    windows = np.array([np.abs(np.fft.fft(x)) for x in windows])
    return windows

hamming_windows_fourier = fourier_window_signal (tone, 100, hamm_win)
rect_windows_fourier = fourier_window_signal (tone, 100, rect_win)
```

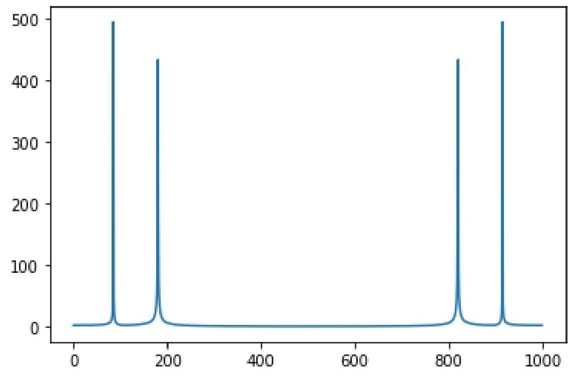
In [10]:

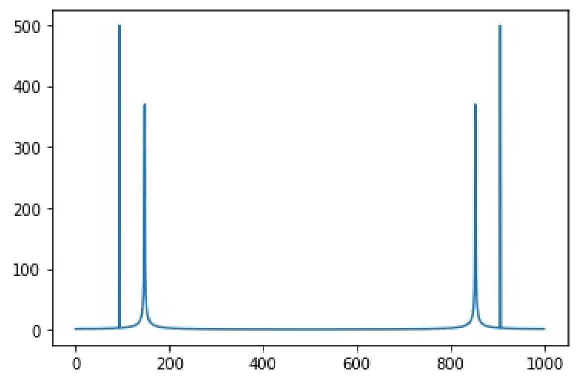
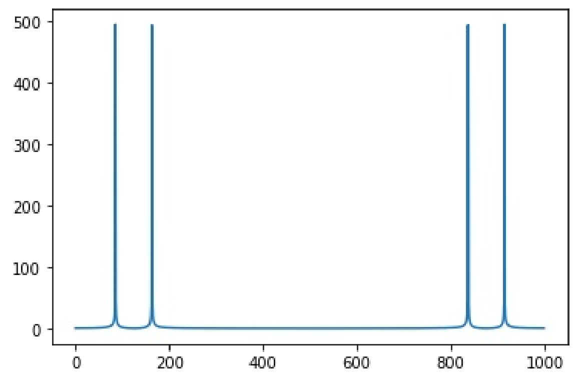
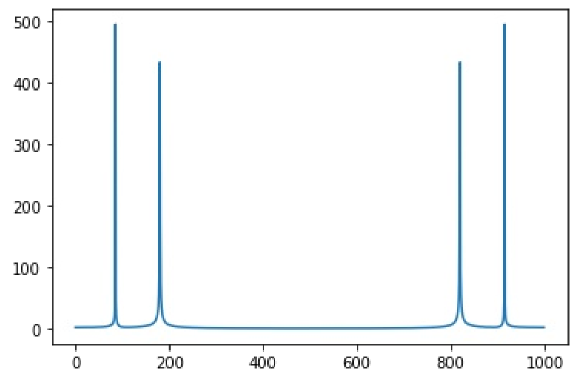
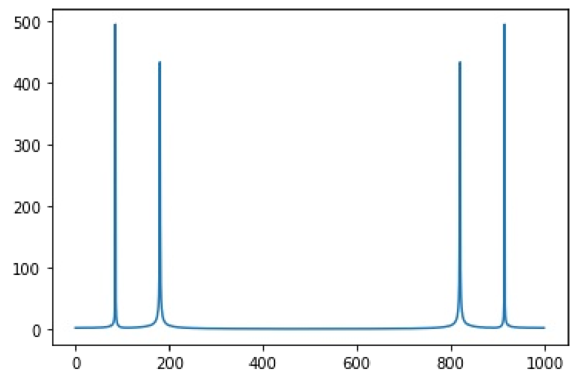
```
# Rectangular Window

for i in range(8) :
    plt.plot(abs(rect_windows_fourier[i]))
    plt.show()
```





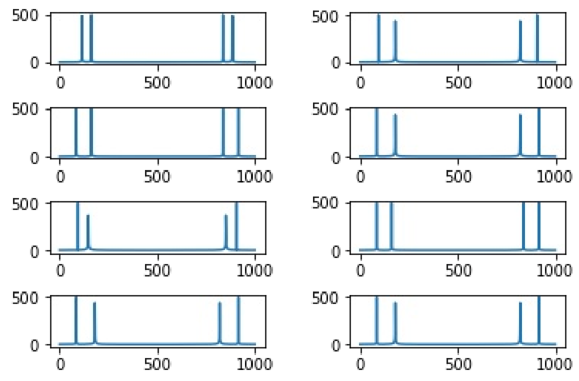




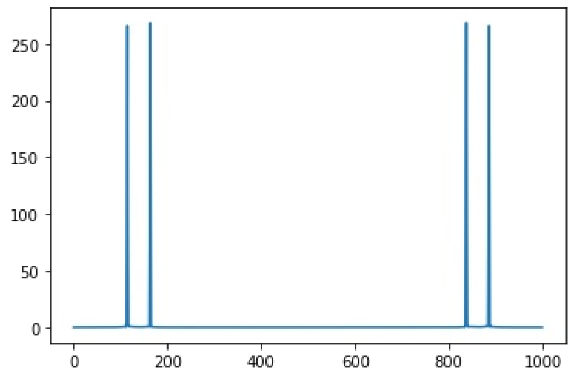
In [11]:

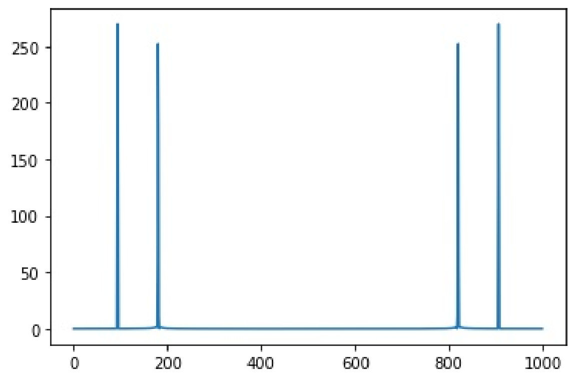
```
# Alternative Demonstration

for i in range(8) :
    plt.subplot(4, 2, i + 1)
    plt.plot(abs(rect_windows_fourier[i]))

plt.subplots_adjust(wspace = 0.4, hspace = 0.8)
plt.show()
```

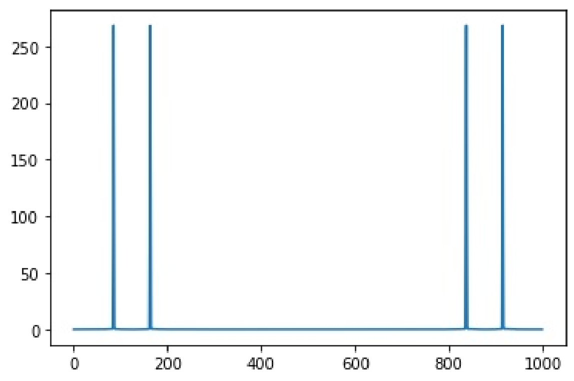


In [12]:

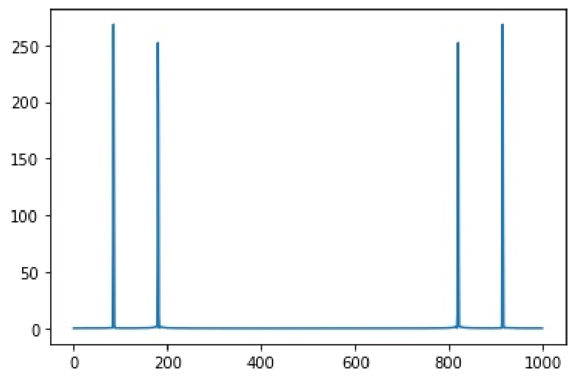
```
# Hamming Window

for i in range(8) :
    #plt.subplot(4, 2, i + 1)
    plt.plot(abs(hamming_windows_fourier[i]))
    plt.show()

#plt.subplots_adjust(wspace = 0.4, hspace = 0.8)
#plt.show()
```



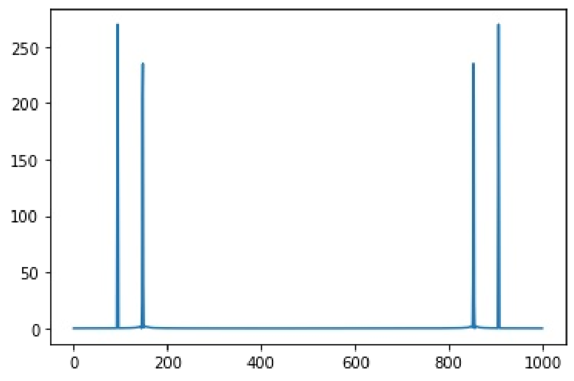




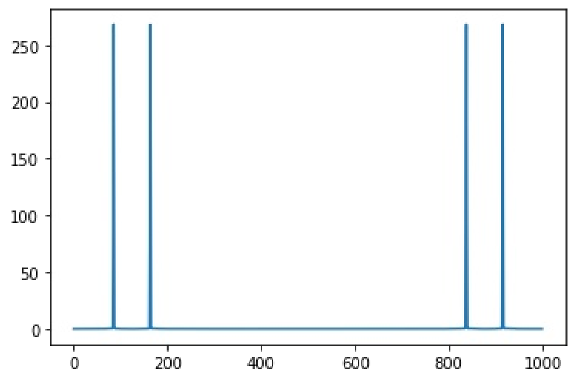
0
50
100
150
200
250
0
200
400
600
800
1000



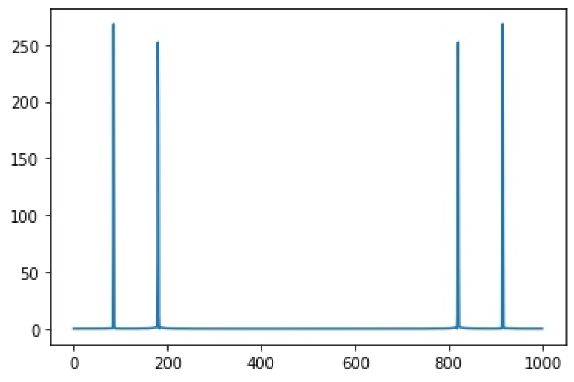
0
50
100
150
200
250
0
200
400
600
800
1000



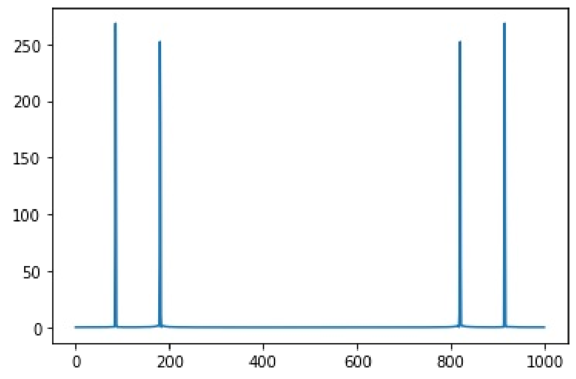
0
50
100
150
200
250
0
200
400
600
800
1000



0
50
100
150
200
250
0
200
400
600
800
1000

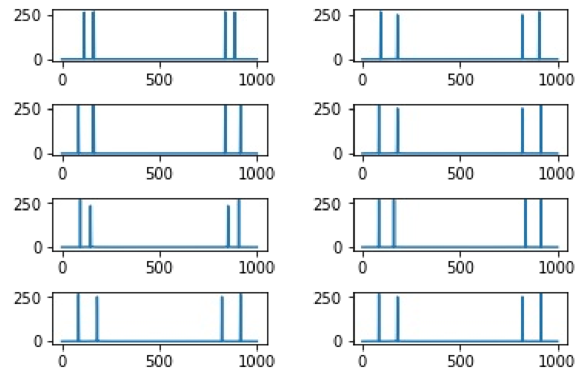


0
50
100
150
200
250
0
200
400
600
800
1000



In [13]:

```
# Alternative Demonstration

for i in range(8) :
    plt.subplot(4, 2, i + 1)
    plt.plot(abs(hamming_windows_fourier[i]))

plt.subplots_adjust(wspace = 0.4, hspace = 0.8)
plt.show()
```



In [14]:

```
# 1.5

fourier_of_column = np.array([abs(np.fft.fft(np.sin(x*n))) for x in column])
k_column = np.array([sc.signal.find_peaks(x)[0][0] for x in fourier_of_column])

fourier_of_row = np.array([abs(np.fft.fft(np.sin(x*n))) for x in row])
k_row = np.array([sc.signal.find_peaks(x)[0][0] for x in fourier_of_row])

#Create k
k = np.empty((10, 2))
k[0] = np.array([k_row[3], k_column[1]])

for i in range(length) :
    for j in range(np.size(k_column)) :
        k[length * i + j + 1][0] = k_row[i]
        k[length * i + j + 1][1] = k_column[j]

for i in range(np.shape(k)[0]):
    print("For the digit", i, "peaks are located at:", int(k[i][0]), int(k[i][1]))
```

```
For the digit 0 peaks are located at: 115 163
For the digit 1 peaks are located at: 85 148
For the digit 2 peaks are located at: 85 163
For the digit 3 peaks are located at: 85 180
For the digit 4 peaks are located at: 94 148
For the digit 5 peaks are located at: 94 163
For the digit 6 peaks are located at: 94 180
For the digit 7 peaks are located at: 104 148
For the digit 8 peaks are located at: 104 163
For the digit 9 peaks are located at: 104 180
```

In [15]:

```
# 1.6

# The signal without zeros
def remove_zeros(sig) :
    return sig[sig != 0]
```

In [16]:

```
# Absolute value DFT
def fourier_windowing (sig) :
    tones = remove_zeros(sig)
    len_tones = np.size(tones)
    num_of_tones = len_tones // 1000
    windows = np.empty(num_of_tones, dtype = object)

    for i in range(0, len_tones, 1000) :
        windows[i // 1000] = tones[i : (i+1000)]

    windows = np.array([np.abs(np.fft.fft(x)) for x in windows])
    return windows
```

In [17]:

```
# List of digits pressed
def ttdecode(sig) :
    ans = []
    windows = fourier_windowing(sig)

    for x in windows :
        len_k = np.shape(k)[0]
        peaks = sc.signal.find_peaks(x, height = 100)[0] # Energy max when |DFT| max
        temp = np.array([peaks[0], peaks[1]]) # temp stores the first two peaks
        for i in range(len_k) :
            if np.array_equal(temp, k[i]) : ans.append(i)
    for x in ans : print(x, end = " ")
    print("\n")
    return

print("The digits of the created signal are :", end = " ")
ttdecode(tone)
```

```
The digits of the created signal are : 0 6 2 3 4 2 3 3
```

In [18]:

```
# 1.7

hardSig = np.load("hardSig.npy")
easySig = np.load("easySig.npy")

print("The digits of the hardSig signal are:", end = " ")
ttdecode(hardSig)
print("The digits of the easySig signal are:", end = " ")
ttdecode(easySig)
```

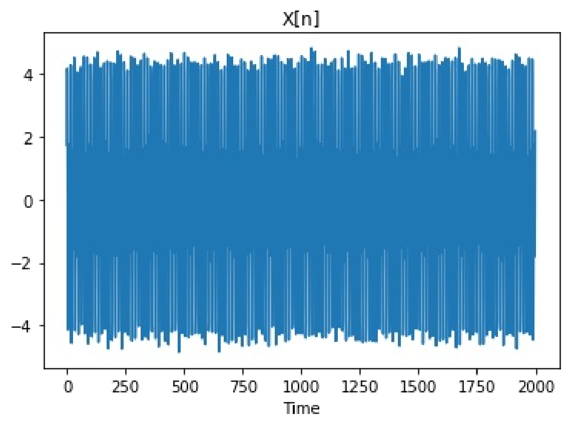
```
The digits of the hardSig signal are: 9 0 9 6 3 2 1 1 9 1

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

In [19]:

```
# 2.1

# a
# Sampling with Fs = 1000Hz the x(t) in [0,2] sec:
# x(t) = 2 cos(2π70t) + 3 sin(2π140t) + 0.15v(t)

Fs = 1000;
Ts = 1/Fs
n = np.linspace(0,2000,2000)
x = 2*np.cos(2*np.pi*70*n*Ts) + 3*np.sin(2*np.pi*140*n*Ts) + 0.15*np.random.normal(0, 1, 2000)
plt.plot(n, x)
plt.xlabel('Time')
plt.title('X[n]')
```

Out[19]:

```
Text(0.5, 1.0, 'X[n]')
```



In [20]:

```
# b

STFT = lb.core.stft(x, 2048, 20, 40)
print(STFT.shape)
```

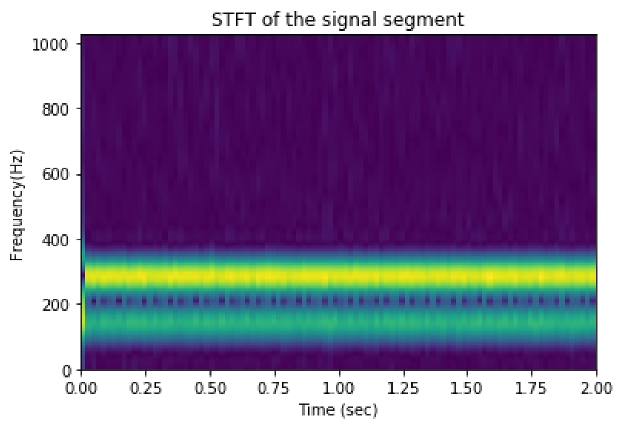
```
(1025, 101)
```

In [21]:

```
t = np.linspace(0, 2, 101) # split time in 101 samples
f = np.linspace(0, 1025, 1025)
plt.pcolormesh(t, f, np.abs(STFT))
plt.xlabel('Time (sec)')
plt.ylabel('Frequency(Hz)')
plt.title('STFT of the signal segment')
```

Out[21]:

```
Text(0.5, 1.0, 'STFT of the signal segment')
```



In [22]:

```
# c

s = np.power(2, np.linspace(1, 6, 100))
coefs,freqs = pywt.cwt(x, s, 'cmor3.0-1.0')
print(coefs.shape)
```

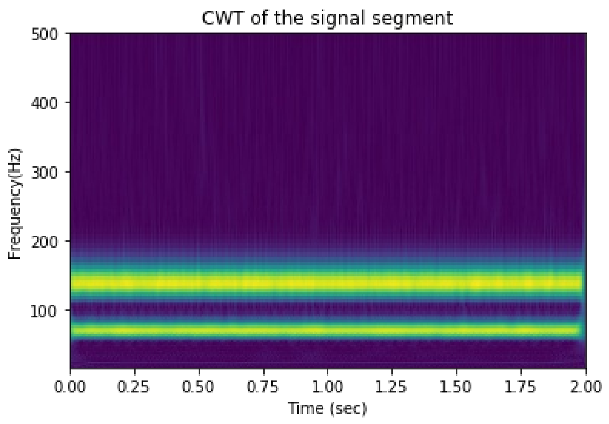
```
(100, 2000)
```

In [23]:

```
t = np.linspace(0,2,2000)
f = freqs*1000
plt.pcolormesh(t,f,np.abs(coefs))
plt.xlabel('Time (sec)')
plt.ylabel('Frequency(Hz)')
plt.title('CWT of the signal segment')
```

Out[23]:

```
Text(0.5, 1.0, 'CWT of the signal segment')
```



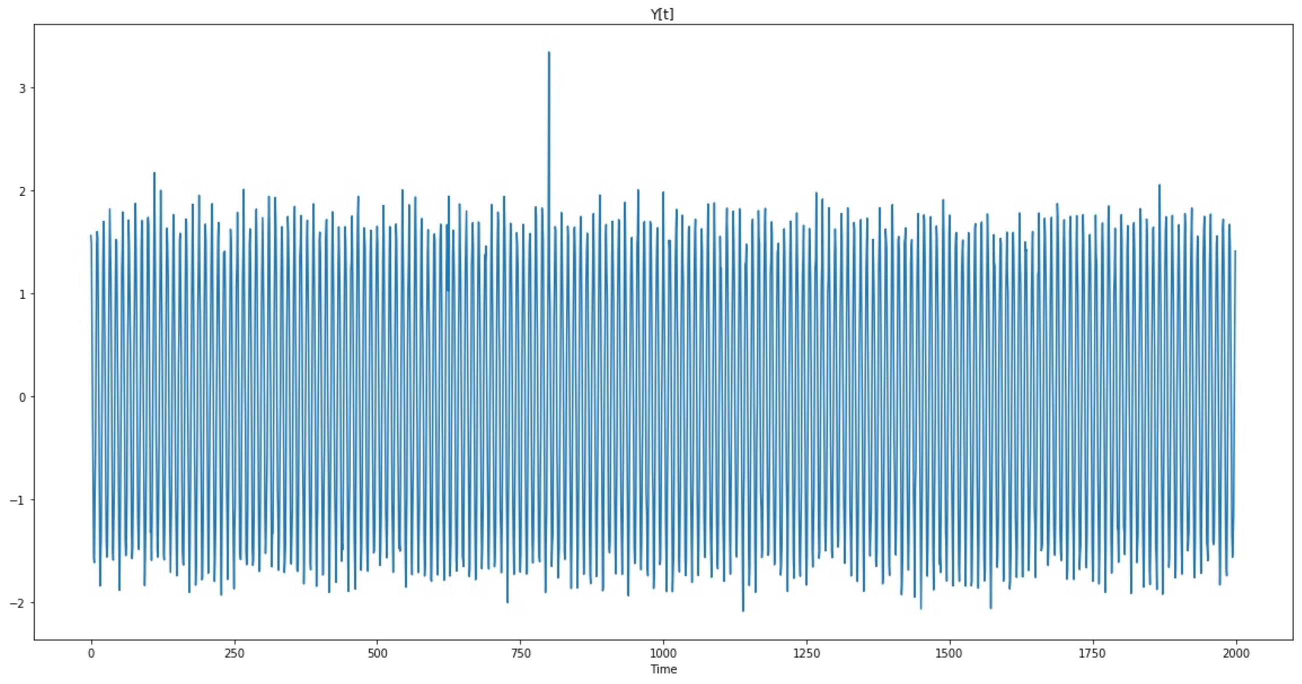
In [24]:

```
#d

# Ουσιαστικά με την χρήση των κυματιδίων μπορούμε να διακρίνουμε καλύτερα
# τις συχνότητες των δύο ημιτόνων οι οποίες δεν αλλάζουν στον χρόνο ενώ
# παράλληλα έχουμε λιγότερη επίδραση του λευκού θορύβου.
# Επίσης, ο DT-CWT ή Wavelets Transformation, έχει καλύτερη ανάλυση
# στο χρόνο για υψηλές συχνότητες αλλά και στη συχνότητα για μεγάλης
# διάρκειας σήματα εξαιτίας του μήκους παραθύρου.
```

```
In [25]:
```

```
# 2.2

# a

Fs = 1000;
Ts = 1/Fs
n = np.arange(2000)
y = 1.7*np.cos(2*90*np.pi*n*Ts) + 0.15*np.random.normal(0,1,2000) + 1.7*(sc.signal.unit_impulse(2000, 625) + sc.s
ignal.unit_impulse(2000, 800))
plt.figure(figsize=(20,10))
plt.plot(n, y)
plt.xlabel('Time')
plt.title('Y[t]')
```

```
Out[25]:

Text(0.5, 1.0, 'Y[t]')
```



```
In [26]:
```

```
# b

STFT = lb.stft(y, 1024, 20, 40)
print(STFT.shape)
```

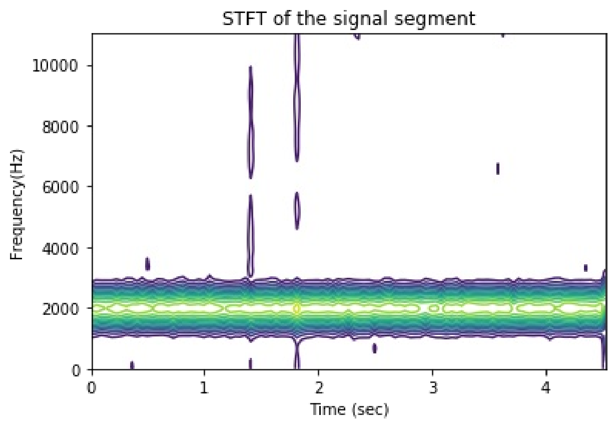
```
(513, 101)
```

In [27]:

```
new_t=np.linspace(0,100000/22050,101)
new_f=np.linspace(0,11025,513)
plt.contour(new_t,new_f,abs(STFT), 15)
plt.xlabel('Time (sec)')
plt.ylabel('Frequency(Hz)')
plt.title('STFT of the signal segment')
```

Out[27]:

```
Text(0.5, 1.0, 'STFT of the signal segment')
```



In [28]:

```
# c

s = np.power(2, np.linspace(1, 6, 1000))
coefs,freqs = pywt.cwt(y, s, 'cmor3.0-1.0')
print(coefs.shape)
```

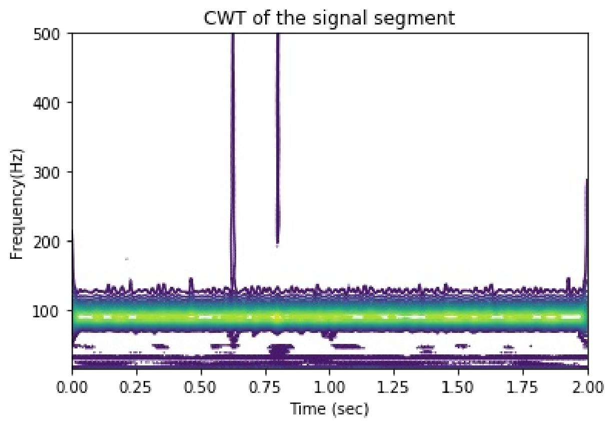
```
(1000, 2000)
```

In [29]:

```
t = np.linspace(0,2,2000)
f = freqs*1000
plt.contour(t,f,np.abs(coefs), 15)
plt.xlabel('Time (sec)')
plt.ylabel('Frequency(Hz)')
plt.title('CWT of the signal segment')
```

Out[29]:

```
Text(0.5, 1.0, 'CWT of the signal segment')
```



In [30]:

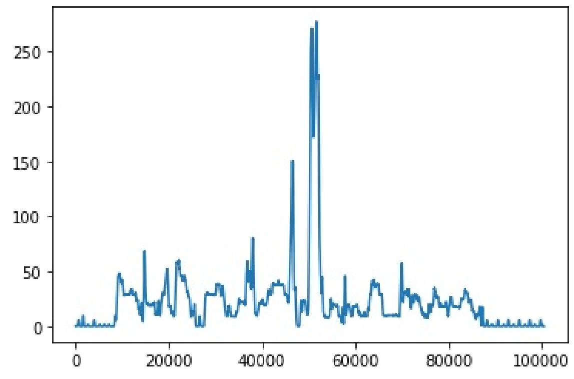
```
# d

# Ο DT-CWT παρουσιάζει κάποιες ανωμαλίες σε σχέση με τον STFT.
```

In [31]:

```
# 3.1

def energy1(sig):
    len_window = np.array([20,30])
    j = 1
    for i in len_window:
        wi = np.hamming(i*16)
        b = np.zeros(1)
        ok1 = np.concatenate((sig, b))
        ok2 = np.concatenate((b, sig))
        y = np.abs(np.sign(ok1) - np.sign(ok2))
        Z = np.convolve(y, wi)

    plt.plot(Z)
    return

def energy2(sig):
    len_window = np.array([20,30])
    j = 1
    for i in len_window:
        wi = np.hamming(i*16)
        tone_sq = sig**2
        E = np.convolve(wi, tone_sq)

    plt.plot(E)
    return
```

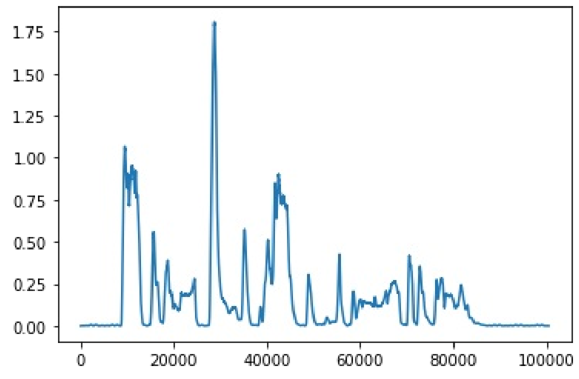
In [32]:

```
x,fs = lb.load('speech_utterance.wav', 22050)
energy1(x)
```



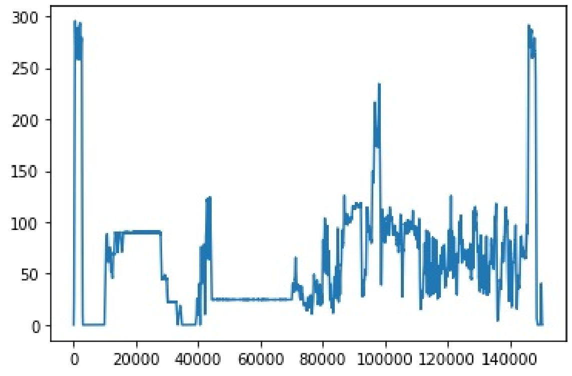
In [33]:

```
energy2(x)
```



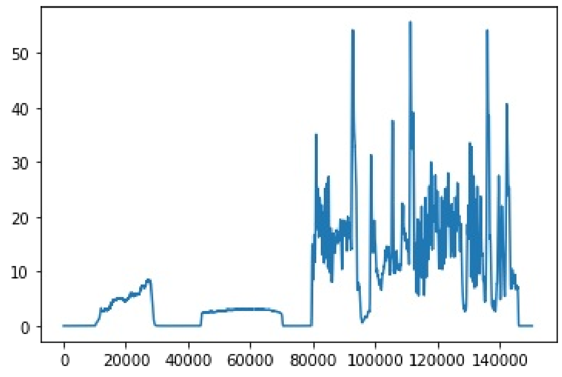
In [34]:

```
# 3.2

y,fs = lb.load('music.wav', 22050)
energy1(y)
```



In [35]:

```
energy2(y)
```



In [36]:

```
# Παρατηρήσεις

# Ο διαχωρισμός φωνής από σιωπή ή/και έμφωνους από άφωνους ήχους δεν μπορεί
# να γίνει με ακρίβεια διότι, όσο πιο μεγάλο είναι το μήκος παραθύρου, τόσο
# μεγαλύτερη είναι η πιθανότητα απώλειας κάποιας μετάβασης από φωνή σε σιωπή
# ή και το αντίθετο. Επιπλέον, είναι γνωστό ότι: Οι έμφωνοι χαρακτήρες
# χαρακτηρίζονται από μια περιοδικότητα και σχετικά μεγάλο πλάτος, άρα θα
# έχουν μεγαλύτερη ενέργεια βραχέος χρόνου και μικρότερο ρυθμό εναλλαγής
# προσήμου, ενώ οι άφωνοι ήχοι που είναι απεριοδικοί, έχουν χαμηλή ενέργεια
# βραχέος χρόνου και πολύ υψηλό ρυθμό εναλλαγής προσήμου. Ακόμη, τα σημειά
# σιωπής έχουν πολύ μικρή μέση τιμή (οριακά μηδενική), αλλά παρουσιάζουν
# περιοδικότητα, για αυτό το λόγο έχουν χαμηλό ρυθμό εναλλαγής προσήμου σε
# σχέση με τους έμφωνους ήχους και χαμηλή ενέργεια βραχέος χρόνου. Τέλος,
# παρατηρείται ότι όσο αυξάνεται το μήκος παραθύρου, τόσο μειώνεται η ακρίβεια
# των ενεργειών βραχέος χρόνου.
```

In [37]:

```
# Alternative Solution

sr = 16000
y1,fs = librosa.load('speech_utterance.wav',sr)
```

In [38]:

```
sd.play(y1[:],sr)
```

In [39]:

```
print(np.size(y1))
y.shape
```

```
72609
```

Out[39]:

```
(150001,)
```

In [40]:

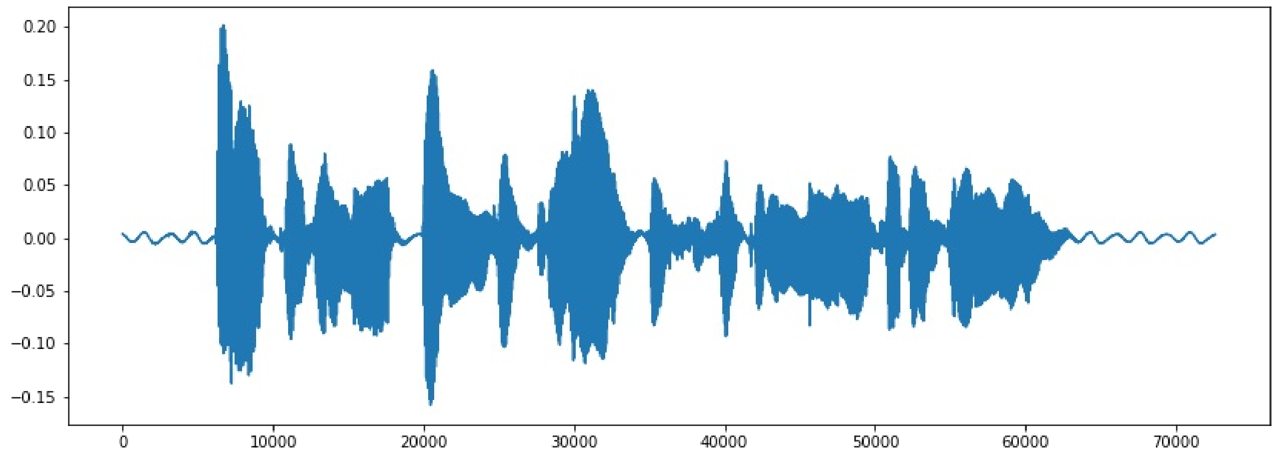
```
librosa.get_duration(y1, fs)
```

Out[40]:

```
4.5380625
```

In [41]:

```
plt.figure(figsize=(14, 5))
plt.plot(y1)
plt.show()
```

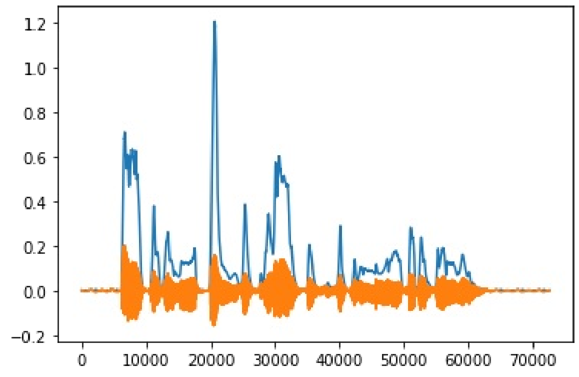


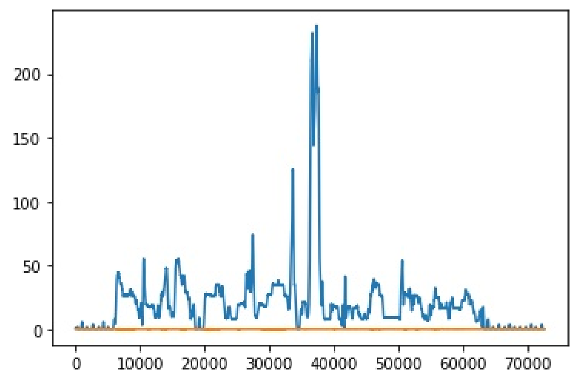
In [42]:

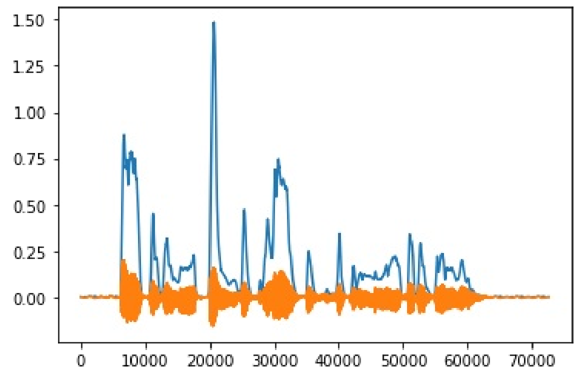
```
def zero_crossing_rate(y,len):
    sgn_y_i_1 = [np.sign(i) for i in y]
    sgn_y_i = np.roll(sgn_y_i_1, 1)
    sgn_y_i[0] = 0
    sub = np.abs(sgn_y_i_1 - sgn_y_i)
    return sp.signal.convolve(sub,np.hamming(len), mode = "same")

def short_time_energy(y,len):
    return sp.signal.convolve(np.abs(y)**2,np.hamming(len), mode = "same")
```

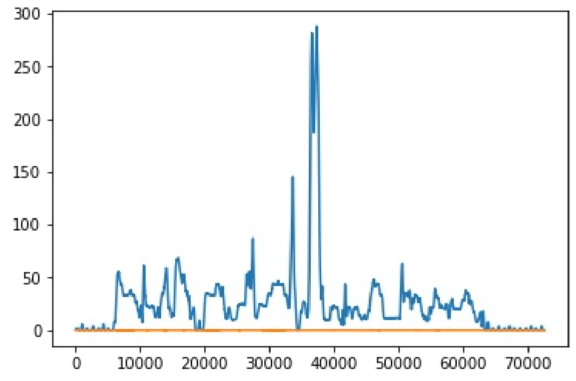
In [43]:

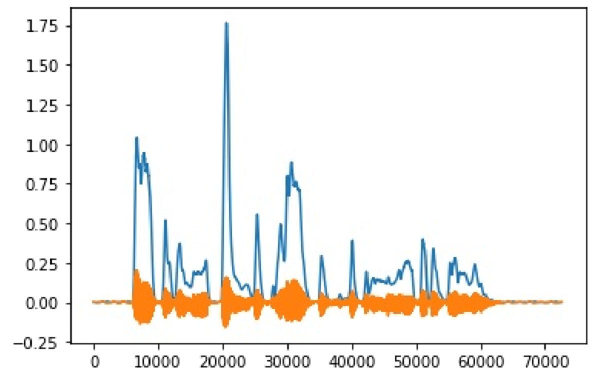
```
hamming_len=np.array((0.02,0.025,0.03))
for i in (hamming_len):
    enrg=short_time_energy(y1,i*16000)
    plt.plot(enrg)
    plt.plot(y1)
    plt.show()
    zr_cr=zero_crossing_rate(y1,i*16000)
    plt.plot(zr_cr)
    plt.plot(y1)
    plt.show()
```

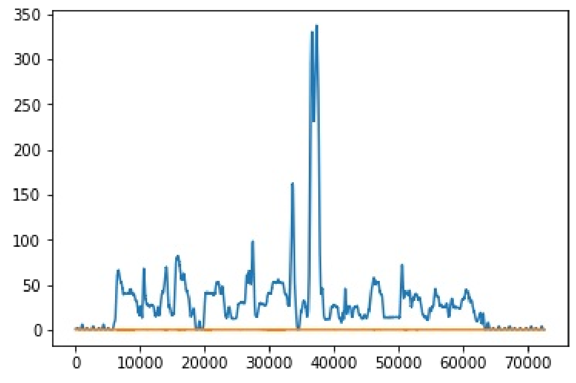












In [44]:

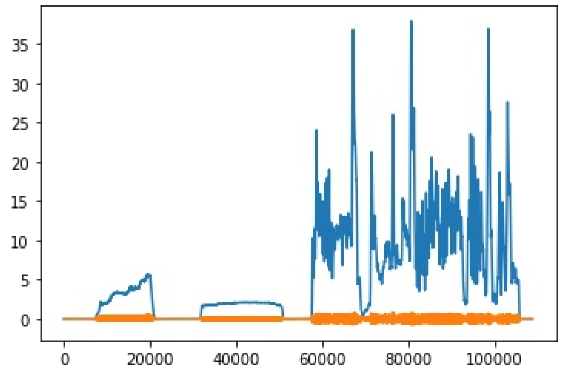
```
y2,sr = librosa.load('music.wav', 16000)
```

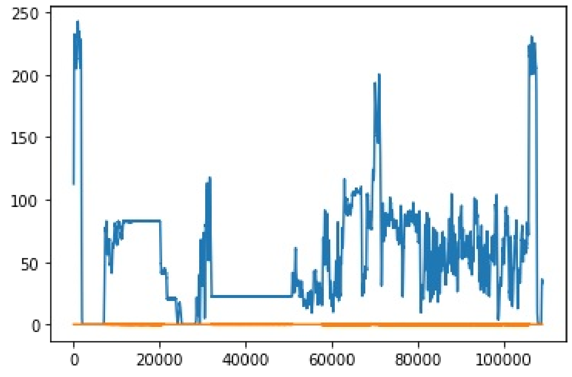
In [45]:

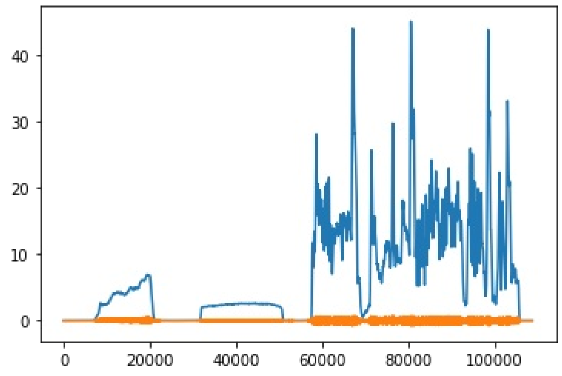
```
sd.play(y2[:],sr)
```

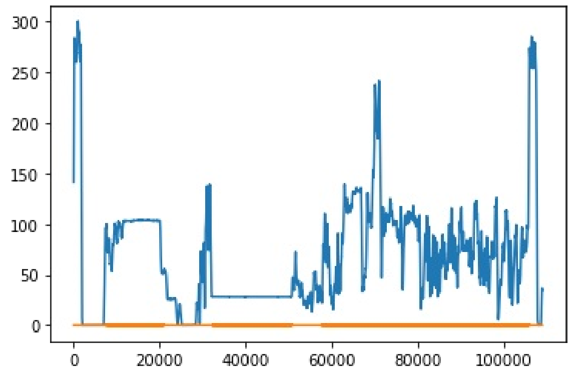
In [46]:

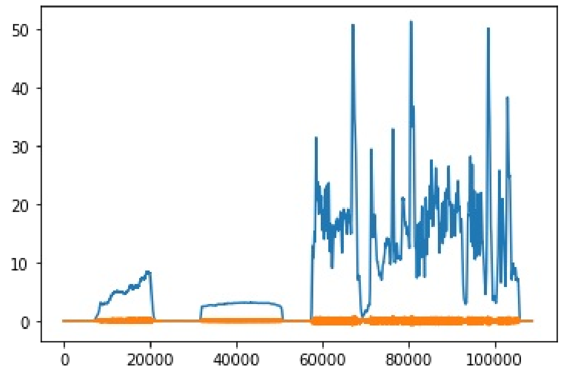
```
for i in (hamming_len):
    enrg=short_time_energy(y2,i*16000)
    plt.plot(enrg)
    plt.plot(y2)
    plt.show()
    zr_cr=zero_crossing_rate(y2,i*16000)
    plt.plot(zr_cr)
    plt.plot(y2)
    plt.show()
```

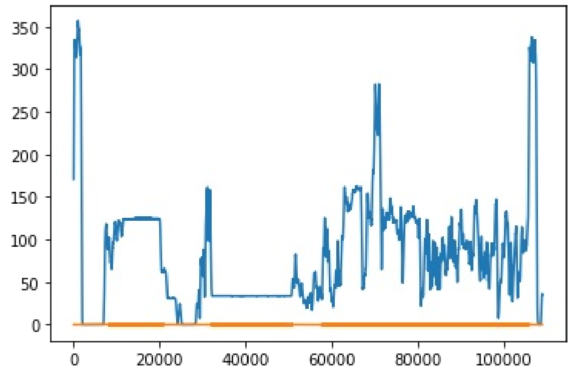












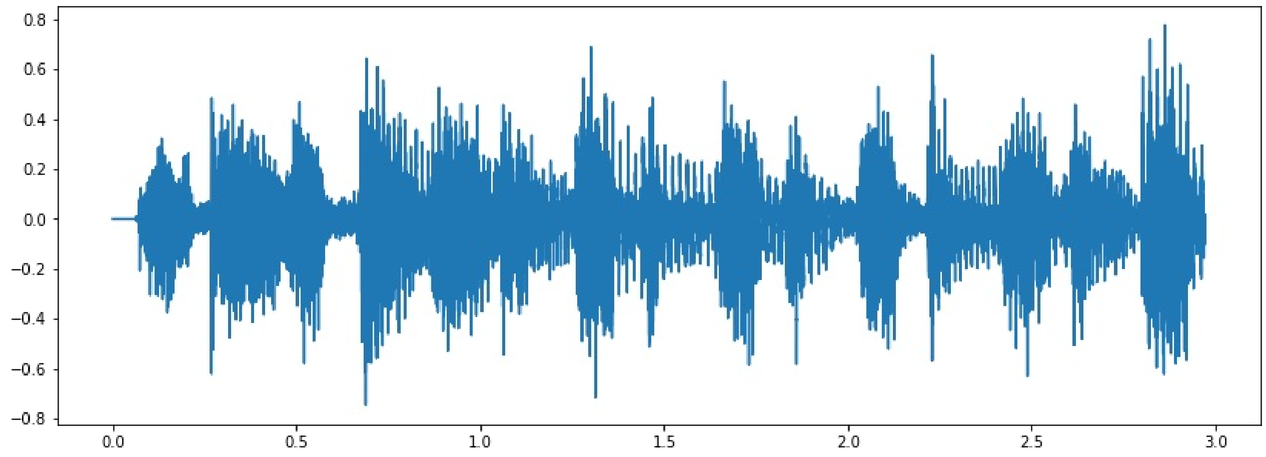
In [47]:

```
# 4.1

sound, sr = librosa.load('salsa_excerpt1.wav')
#play sound
sd.play(sound, sr)

print(len(sound))
print(sr)
L = 2**16
sound_cut = sound[:L]
print(len(sound_cut))
time = len(sound_cut)/sr
print(time)
```

```
661500
22050
65536
2.972154195011338
```

In [48]:

```
t = np.linspace(0.0 , 2**16/22050.0, 2**16)
plt.figure(figsize=(14, 5))
plt.plot(t,sound[:2**16])
plt.show()
```



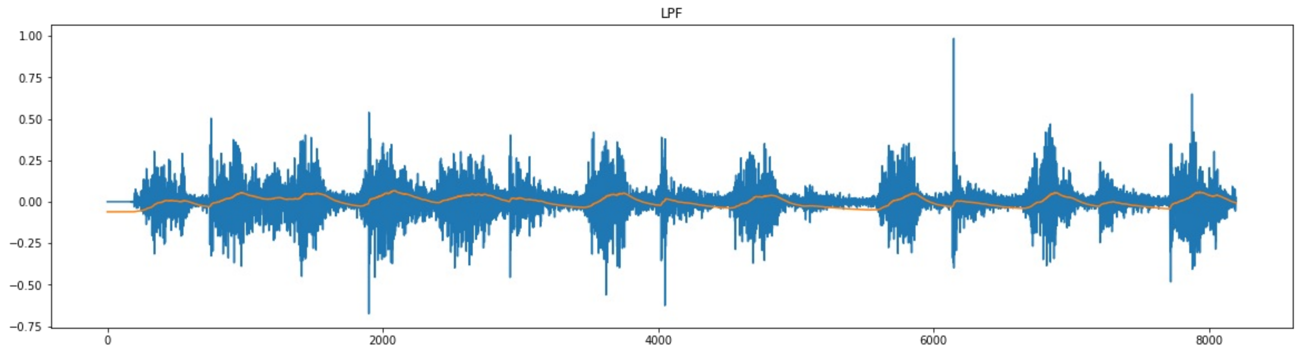
In [49]:

```
# 4.2

y = pywt.wavedec(sound_cut,'db4',level=7)
y = y/np.sqrt(2)

#approximation component
ya = y[0:1]
#detail component
yd = y[1:8]
```

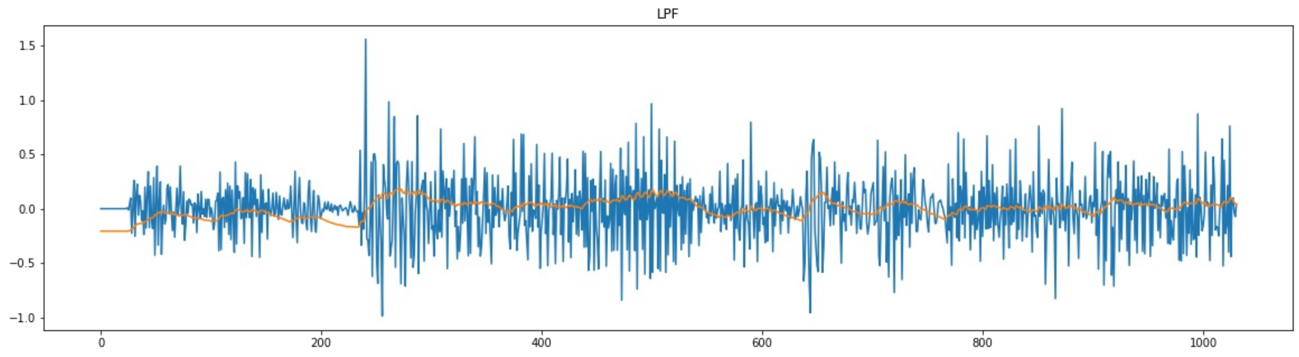
In [50]:

```
# 4.3

z=abs(y)
level=np.empty(8, dtype=int)
b=np.empty(8, dtype=int)
for i in range(1,8):
    level[i]=8-i
level[0]=7
print(level)
print(level[5])

# a0 = 0.001

c=0.005
for i in range(len(level)):
    level[i]=2**level[i]
a=level*0.001
x=np.empty(8, dtype=object)
b=1-a
mean=np.empty(8, dtype=object)
cntrd=np.empty(8, dtype=object)
for i in range(8):
    x[i]=signal.lfilter([a[i]],[1, -b[i]],z[i])
    mean[i]=np.mean(x[i])
    cntrd[i]=x[i]-mean[i]
ya=y[0:1]
yd=y[1:8]
ca=cntrd[0:1]
cd=cntrd[1:8]

t1=np.linspace(0,len(yd[4]), len(yd[4]))
plt.figure(figsize=(20,5))
plt.plot(t1,yd[4])
plt.plot(t1,cd[4])
plt.title('LPF')
plt.show()

t2=np.linspace(0,len(yd[1]), len(yd[1]))
plt.figure(figsize=(20,5))
plt.plot(t2,yd[1])
plt.plot(t2,cd[1])
plt.title('LPF')
plt.show()
```

```
[7 7 6 5 4 3 2 1]
3
```



LPF
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2000
4000
6000
8000

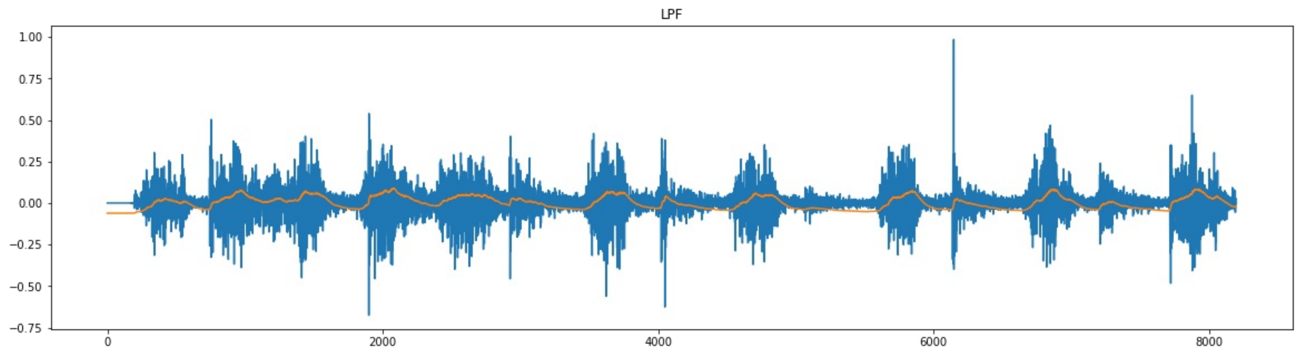


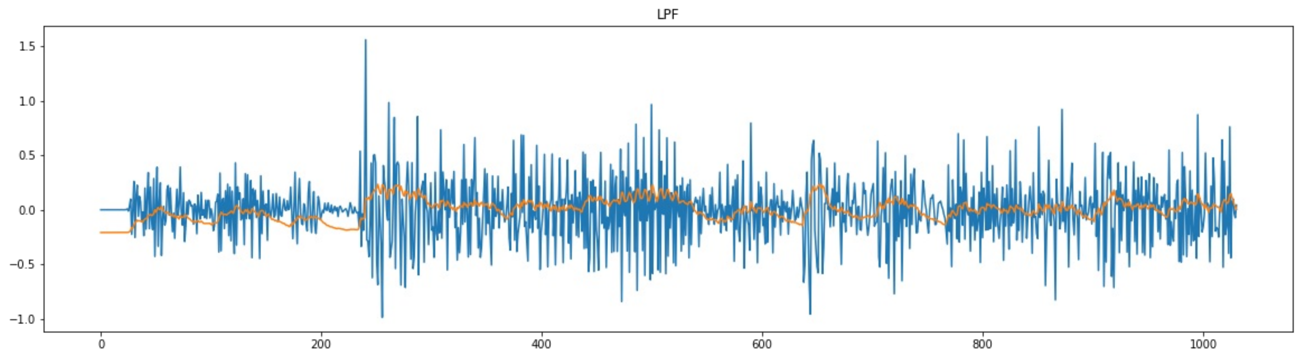
LPF
1.5
1.0
0.5
0.0
−0.5
−1.0
0
200
400
600
800
1000

In [51]:

```
z=abs(y)
level=np.empty(8, dtype=int)
b=np.empty(8, dtype=int)
for i in range(1,8):
    level[i]=8-i
level[0]=7
print(level)
print(level[5])

# a0 = 0.002

c=0.005
for i in range(len(level)):
    level[i]=2**level[i]
a=level*0.002
x=np.empty(8, dtype=object)
b=1-a
mean=np.empty(8, dtype=object)
cntrd=np.empty(8, dtype=object)
for i in range(8):
    x[i]=signal.lfilter([a[i]],[1, -b[i]],z[i])
    mean[i]=np.mean(x[i])
    cntrd[i]=x[i]-mean[i]
ya=y[0:1]
yd=y[1:8]
ca=cntrd[0:1]
cd=cntrd[1:8]

t1=np.linspace(0,len(yd[4]), len(yd[4]))
plt.figure(figsize=(20,5))
plt.plot(t1,yd[4])
plt.plot(t1,cd[4])
plt.title('LPF')
plt.show()

t2=np.linspace(0,len(yd[1]), len(yd[1]))
plt.figure(figsize=(20,5))
plt.plot(t2,yd[1])
plt.plot(t2,cd[1])
plt.title('LPF')
plt.show()
```

```
[7 7 6 5 4 3 2 1]
3
```

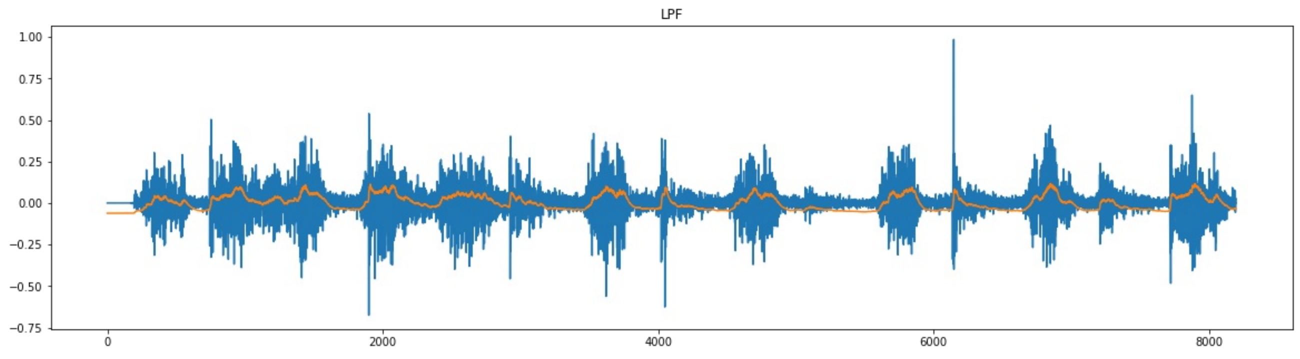


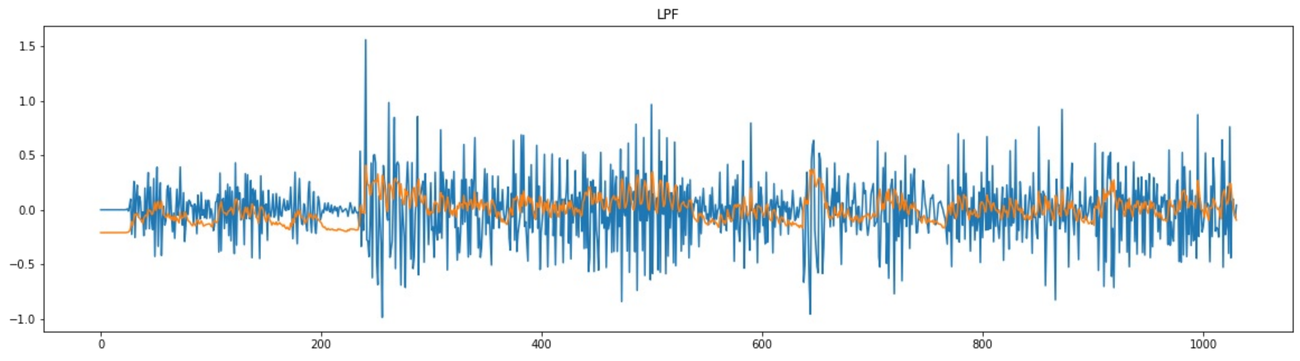


In [52]:

```
z=abs(y)
level=np.empty(8, dtype=int)
b=np.empty(8, dtype=int)
for i in range(1,8):
    level[i]=8-i
level[0]=7
print(level)
print(level[5])

# a0 = 0.005

c=0.005
for i in range(len(level)):
    level[i]=2**level[i]
a=level*0.005
x=np.empty(8, dtype=object)
b=1-a
mean=np.empty(8, dtype=object)
cntrd=np.empty(8, dtype=object)
for i in range(8):
    x[i]=signal.lfilter([a[i]],[1, -b[i]],z[i])
    mean[i]=np.mean(x[i])
    cntrd[i]=x[i]-mean[i]
ya=y[0:1]
yd=y[1:8]
ca=cntrd[0:1]
cd=cntrd[1:8]

t1=np.linspace(0,len(yd[4]), len(yd[4]))
plt.figure(figsize=(20,5))
plt.plot(t1,yd[4])
plt.plot(t1,cd[4])
plt.title('LPF')
plt.show()

t2=np.linspace(0,len(yd[1]), len(yd[1]))
plt.figure(figsize=(20,5))
plt.plot(t2,yd[1])
plt.plot(t2,cd[1])
plt.title('LPF')
plt.show()
```

```
[7 7 6 5 4 3 2 1]
3
```



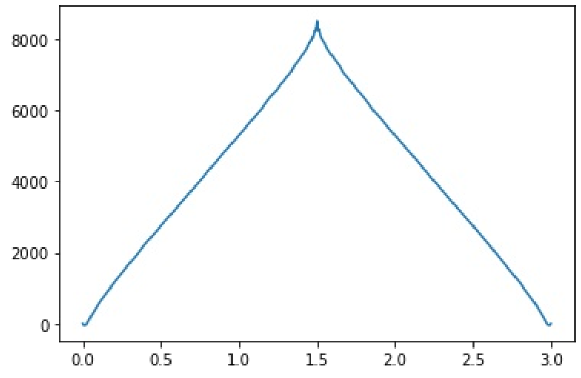


In [53]:

```
# 4.5

sum = 0
k = np.arange(32771)
for i in range(8):

    temp=np.interp(k, np.arange(len(cntrd[i])), cntrd[i])
    sum+=temp
cor=np.correlate(sum,sum,'full')
t=np.linspace(0,3,len(cor))
plt.plot(t,cor)
```

Out[53]:

```
[<matplotlib.lines.Line2D at 0x1b12c9a8448>]
```



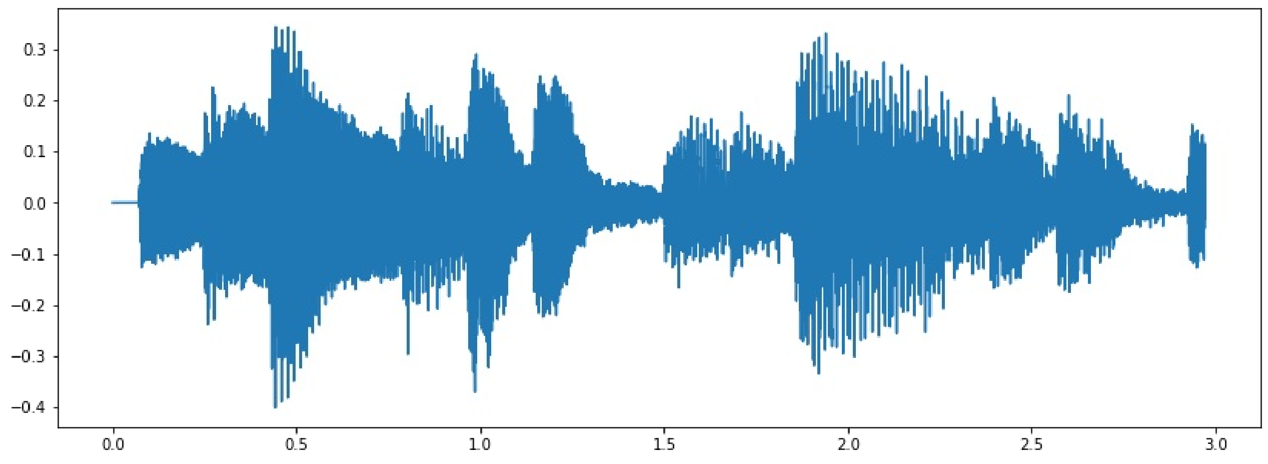
In [54]:

```
# 4.6

# salsa2
sound, sr = librosa.load('salsa_excerpt2.wav')
#play sound
sd.play(sound, sr)

print(len(sound))
print(sr)
L = 2**16
sound_cut = sound[:L]
print(len(sound_cut))
time = len(sound_cut)/sr
print(time)
```

```
661500
22050
65536
2.972154195011338
```

In [55]:

```
t = np.linspace(0.0 , 2**16/22050.0, 2**16)
plt.figure(figsize=(14, 5))
plt.plot(t,sound[:2**16])
plt.show()
```



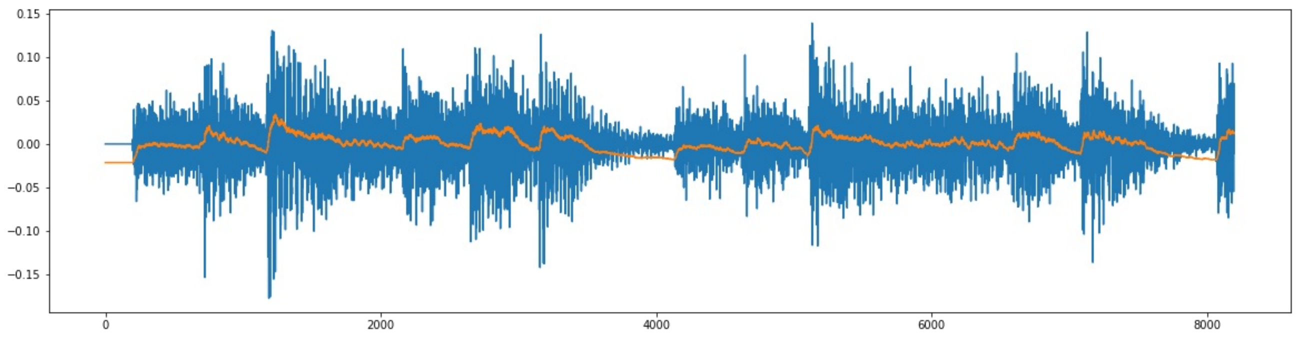
In [56]:

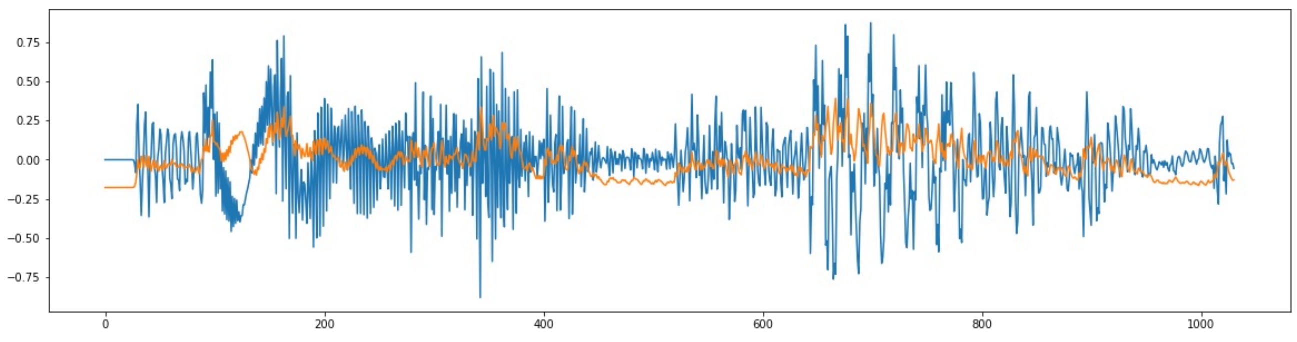
```
y = pywt.wavedec(sound_cut,'db4',level=7)
y = y/np.sqrt(2)

#approximation component
ya = y[0:1]
#detail component
yd = y[1:8]
```

In [57]:

```
z=abs(y)
level=np.empty(8, dtype=int)
b=np.empty(8, dtype=int)
for i in range(1,8):
    level[i]=8-i
level[0]=7
print(level)
print(level[5])

c=0.005
for i in range(len(level)):
    level[i]=2**level[i]
a=level*0.005
x=np.empty(8, dtype=object)
b=1-a
mean=np.empty(8, dtype=object)
cntrd=np.empty(8, dtype=object)
for i in range(8):
    x[i]=signal.lfilter([a[i]],[1, -b[i]],z[i])
    mean[i]=np.mean(x[i])
    cntrd[i]=x[i]-mean[i]
ya=y[0:1]
yd=y[1:8]
ca=cntrd[0:1]
cd=cntrd[1:8]

t1=np.linspace(0,len(yd[4]), len(yd[4]))
plt.figure(figsize=(20,5))
plt.plot(t1,yd[4])
plt.plot(t1,cd[4])
plt.show()

t2=np.linspace(0,len(yd[1]), len(yd[1]))
plt.figure(figsize=(20,5))
plt.plot(t2,yd[1])
plt.plot(t2,cd[1])
plt.show()
```

```
[7 7 6 5 4 3 2 1]
3
```



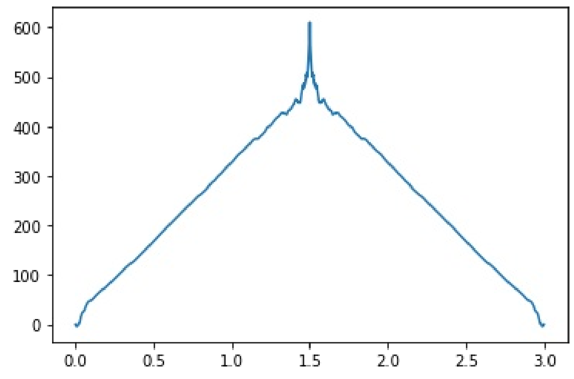


In [58]:

```
sum = 0
k = np.arange(32771)
for i in range(8):

    temp=np.interp(k, np.arange(len(cntrd[i])), cntrd[i])
    sum+=temp
cor=np.correlate(sum,sum,'full')
t=np.linspace(0,3,len(cor))
plt.plot(t,cor)
```

Out[58]:

```
[<matplotlib.lines.Line2D at 0x1b12c9255c8>]
```



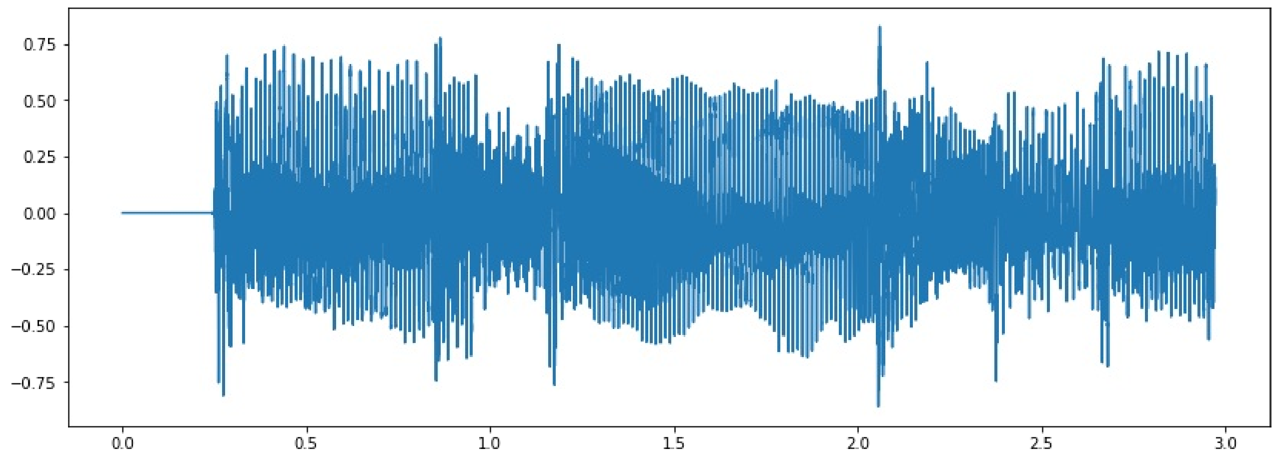
In [59]:

```
# rumba
sound, sr = librosa.load('rumba_excerpt.wav')
#play sound
sd.play(sound, sr)

print(len(sound))
print(sr)
L = 2**16
sound_cut = sound[:L]
print(len(sound_cut))
time = len(sound_cut)/sr
print(time)
```

```
663088
22050
65536
2.972154195011338
```

In [60]:

```
t = np.linspace(0.0 , 2**16/22050.0, 2**16)
plt.figure(figsize=(14, 5))
plt.plot(t,sound[:2**16])
plt.show()
```



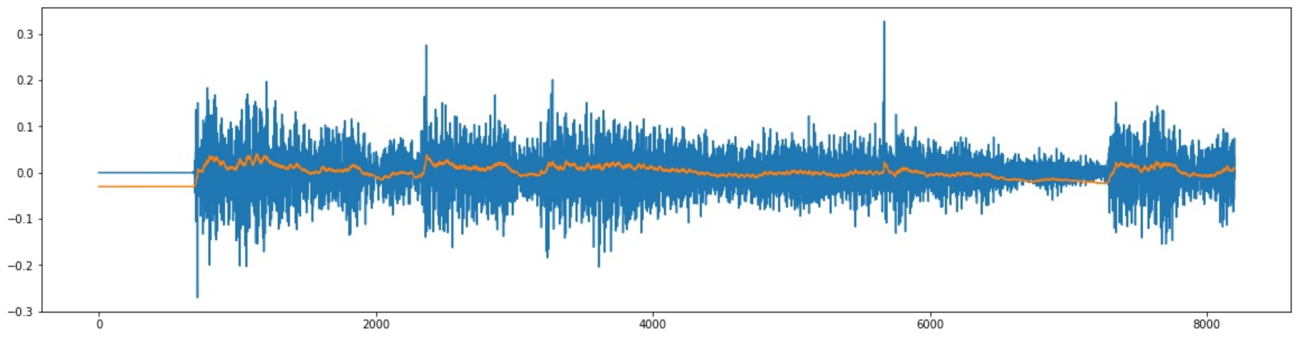
In [61]:

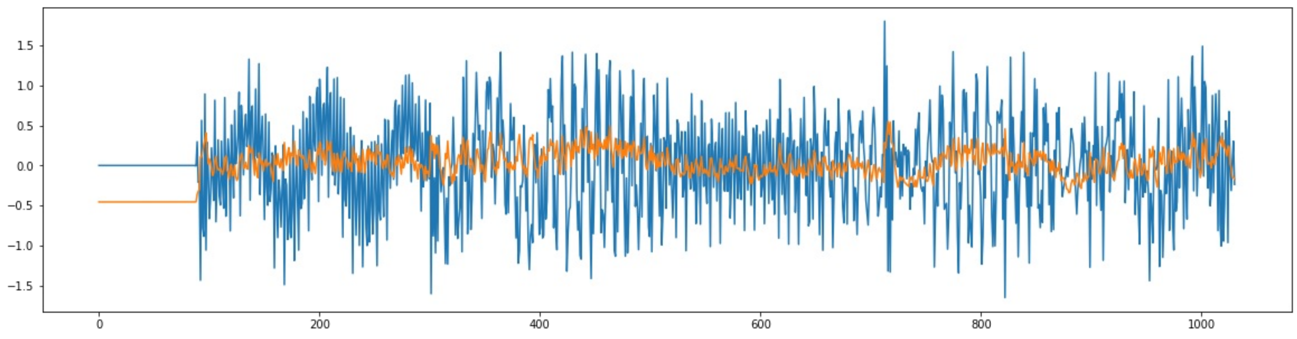
```
y = pywt.wavedec(sound_cut,'db4',level=7)
y = y/np.sqrt(2)

#approximation component
ya = y[0:1]
#detail component
yd = y[1:8]
```

In [62]:

```
z=abs(y)
level=np.empty(8, dtype=int)
b=np.empty(8, dtype=int)
for i in range(1,8):
    level[i]=8-i
level[0]=7
print(level)
print(level[5])

c=0.005
for i in range(len(level)):
    level[i]=2**level[i]
a=level*0.005
x=np.empty(8, dtype=object)
b=1-a
mean=np.empty(8, dtype=object)
cntrd=np.empty(8, dtype=object)
for i in range(8):
    x[i]=signal.lfilter([a[i]],[1, -b[i]],z[i])
    mean[i]=np.mean(x[i])
    cntrd[i]=x[i]-mean[i]
ya=y[0:1]
yd=y[1:8]
ca=cntrd[0:1]
cd=cntrd[1:8]

t1=np.linspace(0,len(yd[4]), len(yd[4]))
plt.figure(figsize=(20,5))
plt.plot(t1,yd[4])
plt.plot(t1,cd[4])
plt.show()

t2=np.linspace(0,len(yd[1]), len(yd[1]))
plt.figure(figsize=(20,5))
plt.plot(t2,yd[1])
plt.plot(t2,cd[1])
plt.show()
```

```
[7 7 6 5 4 3 2 1]
3
```



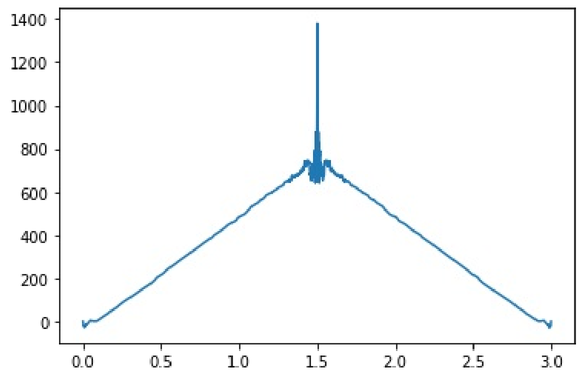


In [63]:

```
sum = 0
k = np.arange(32771)
for i in range(8):

    temp=np.interp(k, np.arange(len(cntrd[i])), cntrd[i])
    sum+=temp
cor=np.correlate(sum,sum,'full')
t=np.linspace(0,3,len(cor))
plt.plot(t,cor)
```

Out[63]:

```
[<matplotlib.lines.Line2D at 0x1b120b68a08>]
```



In [ ]: