On importe les bibliotheques dont on va se servir

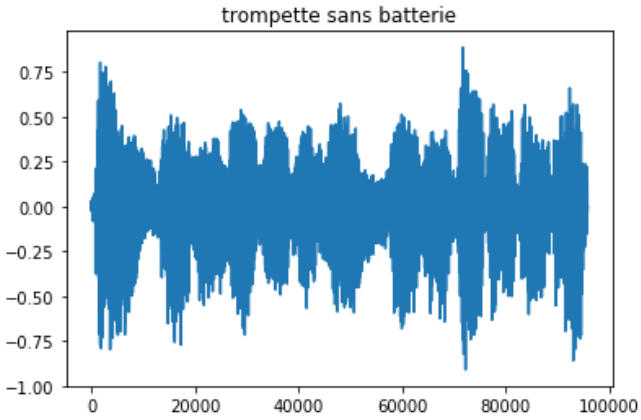
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wavfile
```

On importe les signaux

In [2]:
```
fs, tromp = wavfile.read('trompEssen.wav')
N=tromp.size

#on normalise le signal
m=np.max(np.abs(tromp))
tromp=tromp/(1.1*m)
```

On trace le signal.

In [3]:
```
plt.figure(1)
plt.plot(tromp)
plt.title('trompette sans batterie')
plt.show()
```

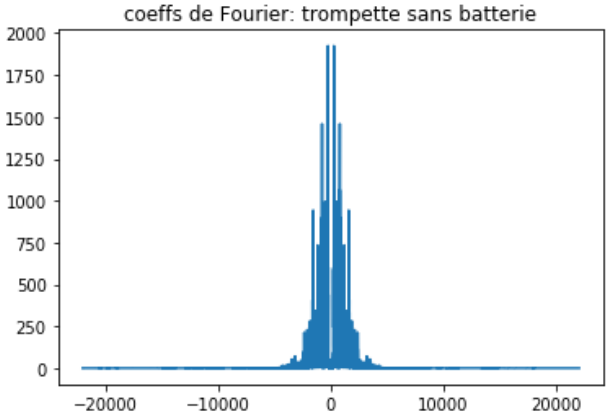


On calcule la transformee discrète du signal

In [4]:
```
trompchap=np.fft.fft(tromp)
```

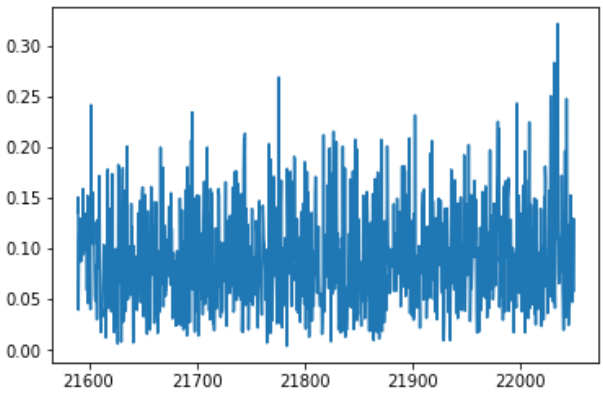
On regarde le resultat. La transformee est complexe ! Il faut donc visualiser le module ou la partie réelle des coefficients. De plus on va normaliser l'axe des fréquences pour pouvoir travailler sur les fréquences physiques. En particulier la fréquence d'échantillonnage f_s nous permet de calculer la fréquence maximale présente dans le signal qui est $f_s/2$.

In [5]:
```
freq=np.arange(-N//2,N-N//2,step=1)*fs/N

plt.figure(2)
plt.plot(freq,np.abs(np.fft.fftshift(trompchap)))
plt.title('coeffs de Fourier: trompette sans batterie')
plt.show()
```



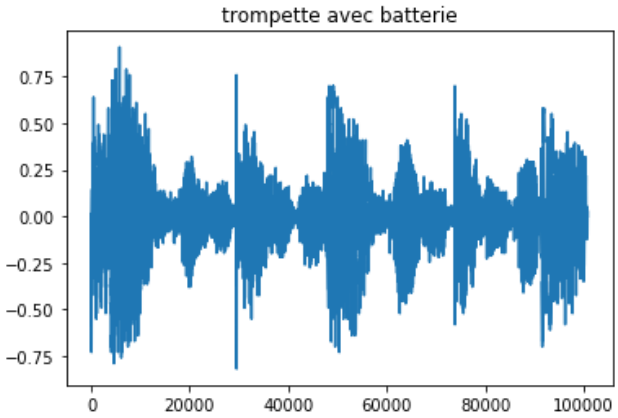
On zoome sur les hautes fréquences pour voir ce qui se passe.

In [6]:
```
tc=np.abs(np.fft.fftshift(trompchap))
nf=tc.size

plt.figure(3)
plt.plot(freq[nf-1000:nf],tc[nf-1000:nf])
plt.show()
```



On effectue les mêmes opérations pour le signal avec la batterie

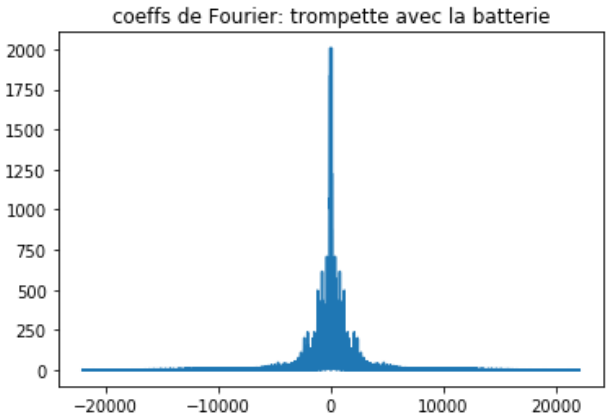
In [7]:
```
fs, trompb = wavfile.read('trompbattEssen.wav')
N=trompb.size

m=np.max(np.abs(trompb))
trompb=trompb/(1.1*m)

plt.figure(4)
plt.plot(trompb)
plt.title('trompette avec batterie')
plt.show()
```



On calcule la représentation fréquentielle du signal

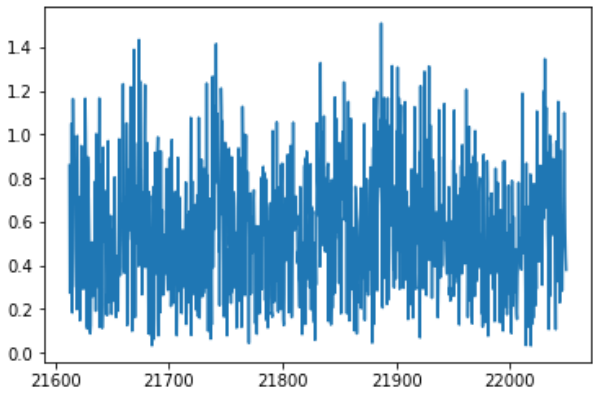
In [8]:
```
trompbchap=np.fft.fft(trompb)
```

In [9]:
```
freq=np.arange(-N//2,N-N//2,step=1)*fs/N

plt.figure(5)
plt.plot(freq,np.abs(np.fft.fftshift(trompbchap)))
plt.title('coeffs de Fourier: trompette avec la batterie')
plt.show()
```



On fait un zoom sur les hautes frequences.

In [10]:
```
tbc=np.abs(np.fft.fftshift(trompbchap))
nf=tbc.size

plt.figure(6)
plt.plot(freq[nf-1000:nf],tbc[nf-1000:nf])
plt.show()
```



On remarque que les hautes fréquences ont une amplitude beaucoup plus forte que dans le signal sans batterie.

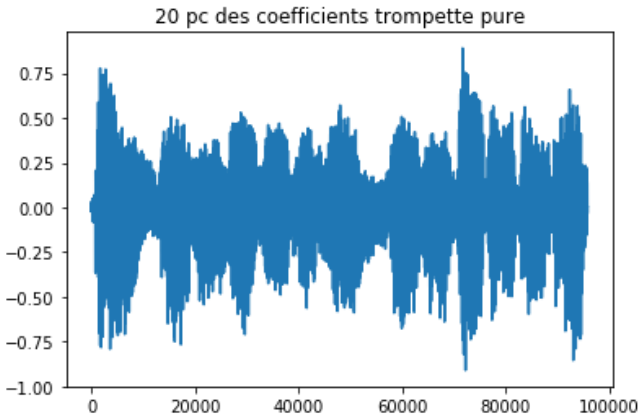
On ne retient maintenant que les fréquences e^n, telles qu'on mette à zéro toutes les fréquences au dessus de $\omega_c = 4000$ Hz et on calcule donc $x_M = \frac{1}{N} \sum_{n=-M}^{M} \langle x, e^n \rangle e^n = \frac{1}{N} \sum_{n=-M}^{M} \hat{x}_n e^n$

On fabrique une fonction qui nous permettra de répéter facilement cette opération.

In [11]:
```
def filtrage(omegac,y,fs):
    ychap=np.fft.fft(y)
    N=ychap.size
    M=int(np.floor(N*omegac/fs))
    mask=np.ones(N)
    coord=np.arange(M+1,N-M,1)
    mask[coord]=0
    yfiltchap=mask*ychap
    yfilt=np.fft.ifft(yfiltchap)
    yfilt=yfilt.real
    m=np.max(np.abs(yfilt))
    yfilt=yfilt/(1.1*m)
    return yfilt
```

Notre opération revient à conserver 20% des coefficients du signal sur la base de Fourier, et à mettre les autres à zéro.

In [12]:
```
omegac=4000

trompfilt=filtrage(omegac,tromp,fs)
plt.figure(7)
plt.plot(trompfilt)
plt.title('20 pc des coefficients trompette pure')
plt.show()

wavfile.write('trompfilt_4000.wav',fs,trompfilt)
```



Examinons l'effet du filtrage sur la partie avec la batterie

In [13]:
```
omegac=4000

trompbfilt=filtrage(omegac,trompb,fs)
plt.figure(8)
plt.plot(trompbfilt)
plt.title('20 pc des coefficients trompette avec la batterie')
plt.show()

wavfile.write('trompbfilt_4000.wav',fs,trompbfilt)
```