# Table of Contents

# Airbus Helicopter Anormaly Detection (Yang WANG)

Dedication:

First of all, I would like to dedicate this work to **Louis Charles BREGUET**, designer, builder, pioneer and inventor of modern helicopter.

# Challenge Large Scale Machine Learning


- Authors:
- Pavlo Mozharovskyi (pavlo.mozharovskyi@telecom-paris.fr), Stephan Clémençon, Jayant Sen Gupta


# Functional anomaly detection

**Anomaly detection** (or **outlier detection**) comprises the **machine learning** methods aimed at identification of observations that exhibit suspicious behaviour and are very likely to cause a problem. In the **unsupervised learning** framework, no label indicating whether a training observation is anomalous or not is available. Hence, anomalies should be identified in an automatic way by learning the *normal* behavior, that of the vast majority of the observations, and considering those differing significantly from it as *abnormal*. Logically, anomalies are rare in the data and thus fall in *low density* regions: anomaly detection thus boils down to identifying the *tail* of the distribution.

With the ubiquitous deployment of sensors monitoring nearly continuously the health of complex infrastructures, **anomaly detection** can now rely on measurements sampled at a very high frequency, providing a very rich representation of the phenomenon under surveillance. In order to exploit fully the collected information, the observations cannot be treated as multivariate data anymore and a functional analysis approach is required.

A (very) short list of **literature** regarding functional anomaly detection can include:


- J.O. Ramsay, Silverman, B.W. (2005): Functional Data Analysis. Springer-Verlag, New-York.
- Ferraty, F., Vieu, P. (2006): Nonparametric Functional Data Analysis: Theory and Practice. Springer Science & Business Media.
- Chandola, V., Banerjee, A., Kumar, V. (2009): Anomaly detection: A survey. ACM Computing Surveys (CSUR) 41(3), 1-58.
- Hubert, M., Rousseeuw, P.J., Segaert, P. (2015): Multivariate functional outlier detection. Statistical Methods & Applications 24(2), 177-202.
- Wang, J.L., Chiou, J.M., Müller, H.G. (2016): Functional data analysis. Annual Review of Statistics and Its Application 3, 257-295.


# The properties of the dataset:

The data set is provided by the Airbus and consistst of the measures of the accelerometer of helicopters during 1 minute at frequency 1024 Hertz, which yields time series measured at in total 60 * 1024 = 61440 equidistant time points.

- Training data: The training set consists of one file, **airbus_train.csv**. File **airbus_train.csv** contains one observation per row, each observation having 61440 entries, measures with equivalent time distance of 1 / 1024 seconds. There are in total 1677 training observations.
- Test data: The training set consists of one file, **airbus_test.csv**, which has the same structure as file **airbus_train.csv**. There are in total 2511 test observations.
- Remark: The task of the **unsupervised anomaly detection** is difficult, in the sense that, strictly speaking, your only feedback will be your score calculated by the submission website.

# The performance criterion:

You should submit a file that contains in each row anomaly score for the observation in the corresponding row of the file **airbus_test.csv**. For a sample submission please see the codes below. Please note, that your score should provide ordering which allows to identify anomalies, i.e. the higher the value of the score, the **more abnormal** the observation should be considered.

The performance criterion is the **Area Under the Receiver Operating Characteristic** (AUC), see also: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve (https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve)

- Training Data

Training data, input (file **airbus_train.csv**): https://partage.imt.fr/index.php/s/zqrRggLBY8GRc9i (https://partage.imt.fr/index.php/s/zqrRggLBY8GRc9i)

- Test Data

Training data, output (file **airbus_test.csv**): https://partage.imt.fr/index.php/s/WpiqcjMq8ymg8zA (https://partage.imt.fr/index.php/s/WpiqcjMq8ymg8zA)

# Import

In [1]:

```
import pandas as pd
import numpy as np

### Signal ###
from scipy import interpolate
from scipy import signal
from scipy import fftpack, fft

### Visualization ###
import matplotlib.pyplot as plt
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from matplotlib.lines import Line2D
from matplotlib.collections import EllipseCollection
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('retina')

### Machine Learning ###
from sklearn.svm import OneClassSVM
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA,kernel_pca
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import mean_squared_error,mean_absolute_error

### keras ###
import keras
from tensorflow.keras import layers
#from tensorflow import keras
from keras import backend as K
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
from keras.models import Model
from keras.losses import mse, binary_crossentropy
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.layers import Conv1D, GlobalMaxPool1D, Dense, Flatten
from keras.callbacks import ModelCheckpoint
```

```
Using TensorFlow backend.
```

## Load and investigate the data

In [2]:

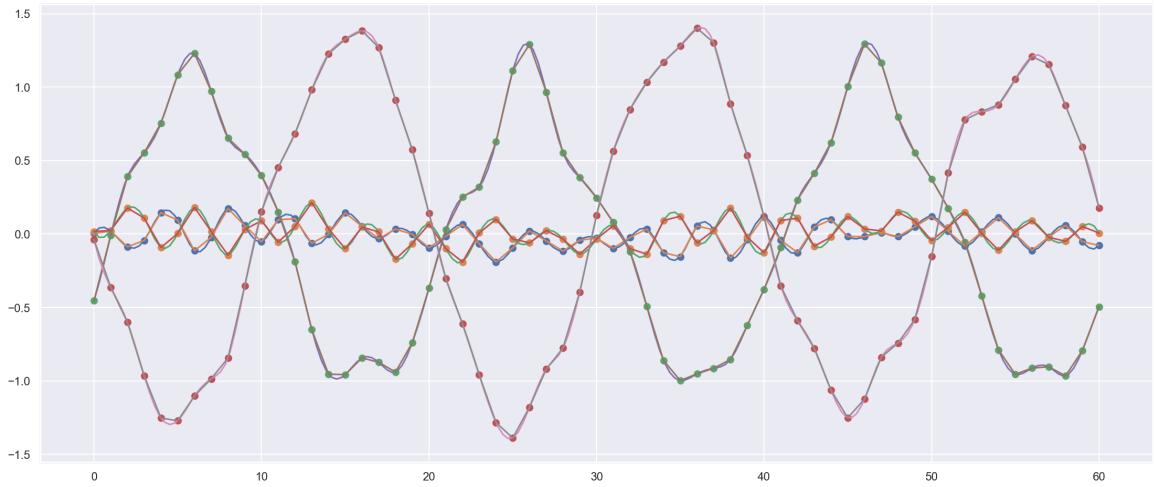
```
xtrain = np.loadtxt('airbus_train.csv', delimiter= ' ')
print(xtrain.shape)
xtest = np.loadtxt('airbus_test.csv', delimiter= ' ')
print(xtest.shape)
```

```
(1677, 61440)
(2511, 61440)
```

## A sample plot

In [147]:

```
# Plot first 614 time points for first 4 observations
plt.figure(figsize=(19, 8))
for i in range(4):
    tck = interpolate.splrep(range(61), xtrain[i+8, :61], s=0)
    xnew = np.linspace(0, 60, 200)  # np.arange(0, 2*np.pi, np.pi/50)
    ynew = interpolate.splev(xnew, tck, der=0)
    # plt.scatter(xnew,ynew)
    plt.plot(xnew, ynew)
    plt.scatter(range(61), xtrain[i+8, :61])
    plt.plot(range(61), xtrain[i+8, :61])
plt.show()
```




# Some physics before start

## What can be the anormalies?

### Aircrafte Flutter

Flutter is a **dynamic instability of an elastic structure in a fluid flow, caused by positive feedback between the body's deflection and the force exerted by the fluid flow**. In a linear system, "**flutter point**" is the point at which the structure is undergoing simple harmonic motion—zero net damping—and so any further decrease in **net damping** will result in a **self-oscillation** and eventual failure. "Net damping" can be understood as the sum of the structure's natural positive damping and the negative damping of the aerodynamic force. Flutter can be classified into two types: **hard flutter**, in which the net damping decreases very suddenly, very close to the flutter point; and **soft flutter**, in which the net damping decreases gradually. https://www.youtube.com/watch?v=qpJBvQXQC2M&t=59s (https://www.youtube.com/watch?v=qpJBvQXQC2M&t=59s)

https://www.youtube.com/watch?v=MEhVk57ydhw (https://www.youtube.com/watch?v=MEhVk57ydhw)

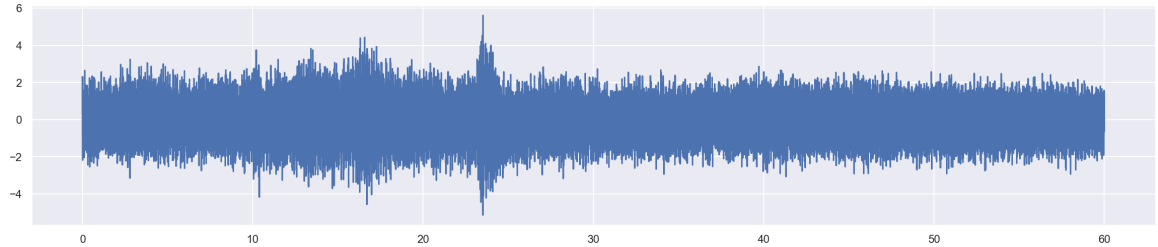
https://www.youtube.com/watch?v=0FeXjhUEXIc (https://www.youtube.com/watch?v=0FeXjhUEXIc)

## What are the possible approaches?

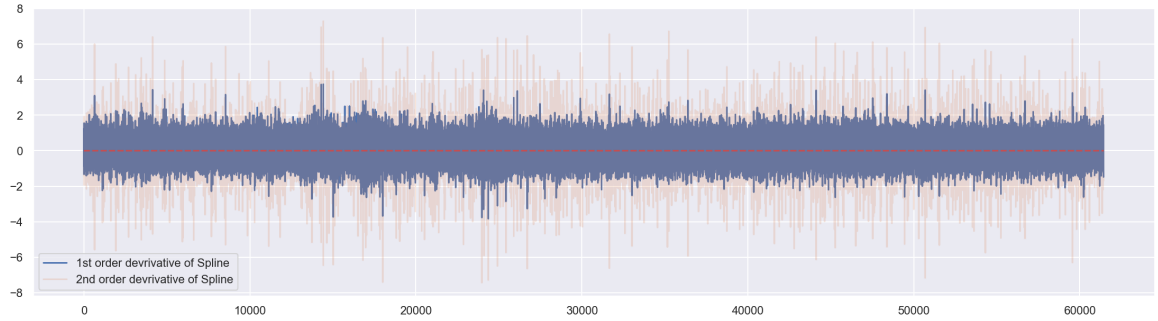
### Raw signal time series observation

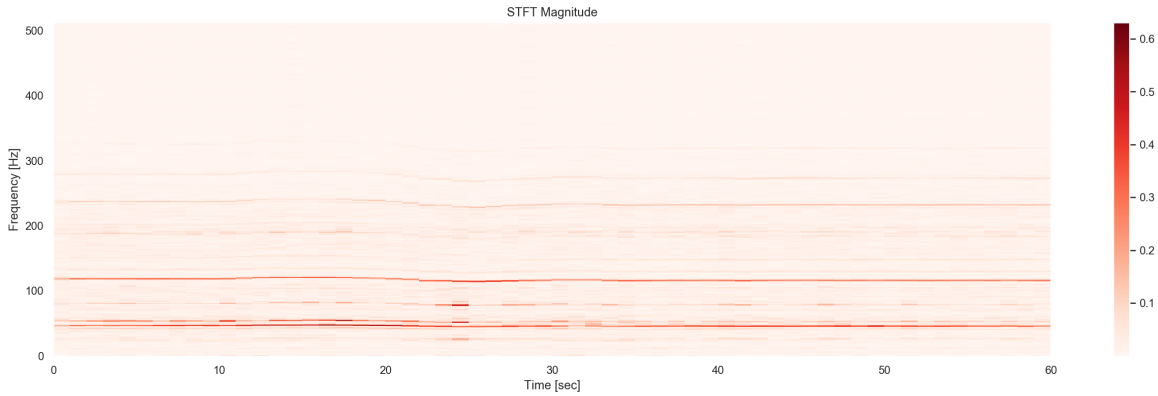
In [1420]:

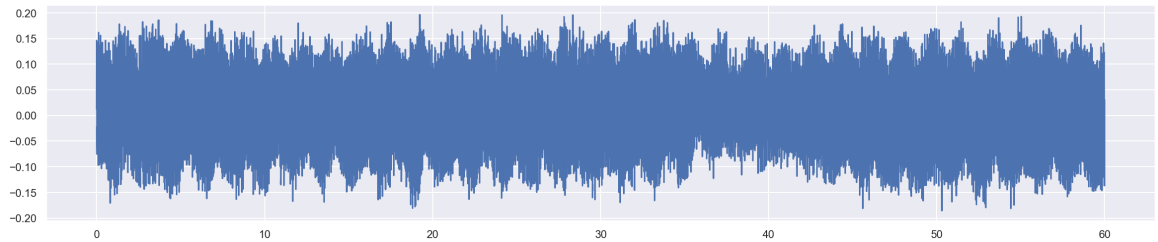
```
for i in range(20):
    k=10
    plot_wave(t,xtest[k+i])
    plot_derive(xtest[k+i])
    #plot_spectrum(xtrain[k+i])
    f,Zxx = plot_STFT(xtest[k+i])
    #average frequence
    #plt.plot((f@np.abs(Zxx))/np.sum(np.abs(Zxx),axis=0))
```

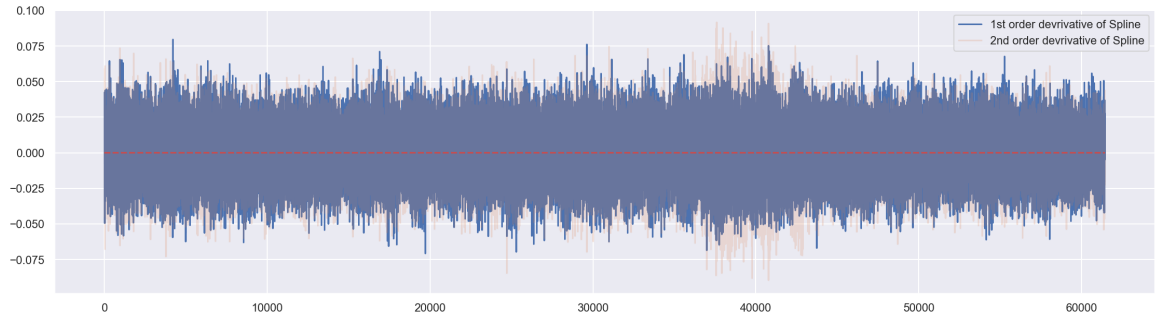


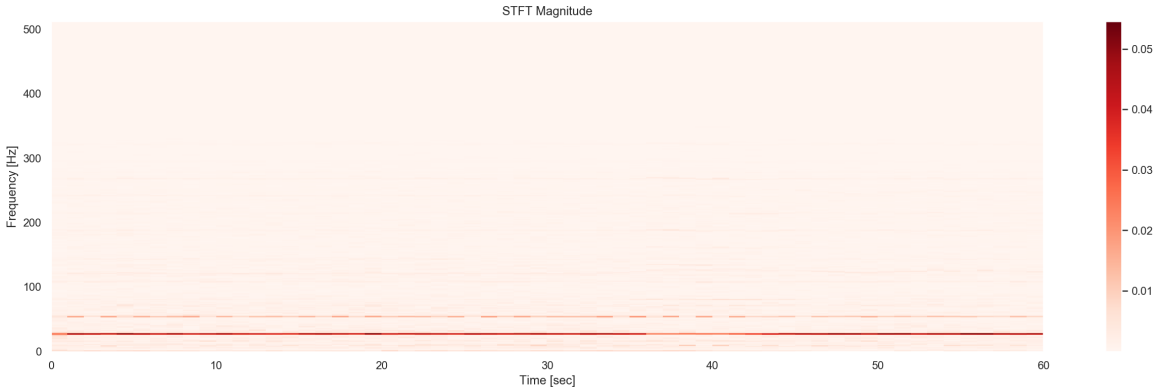

```
//anaconda3/lib/python3.7/site-packages/IPython/core/pylabtools.py:128: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

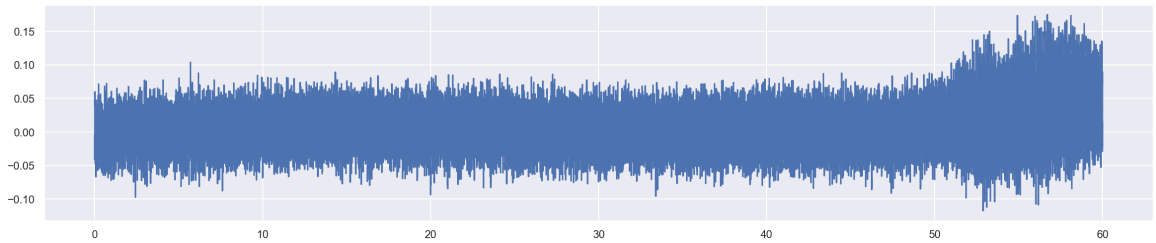


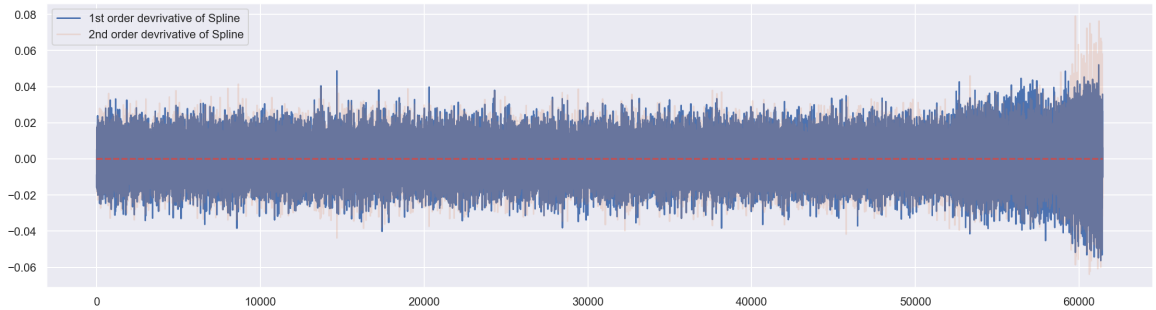



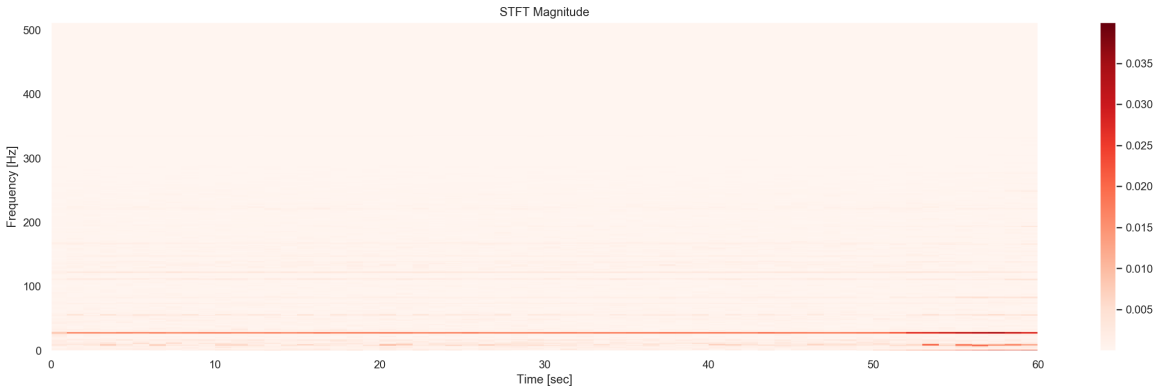



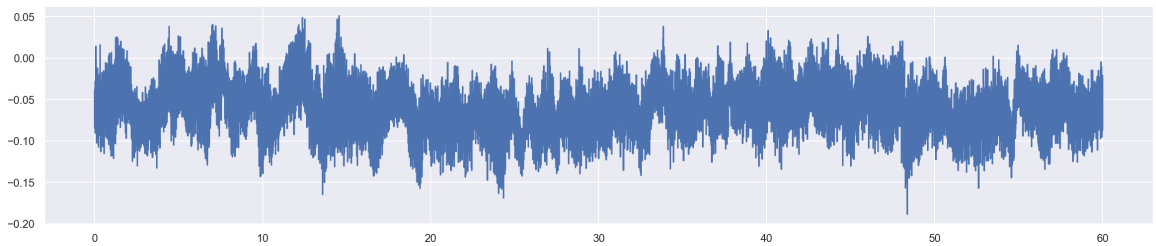



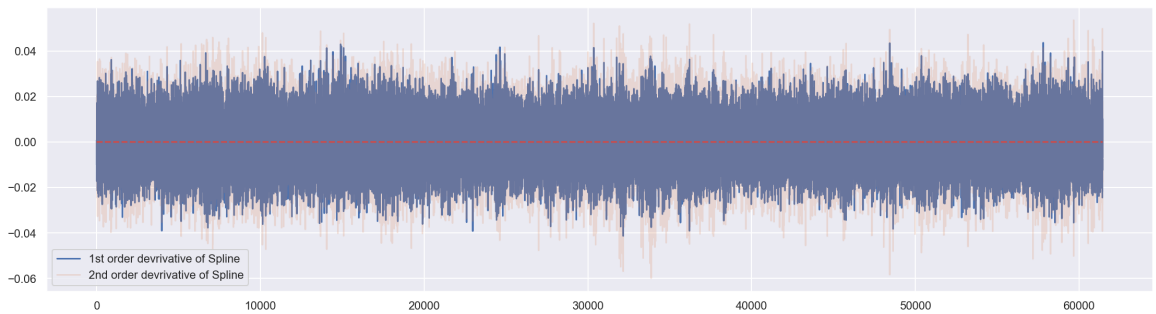

STFT Magnitude
Frequency [Hz]
Time [sec]

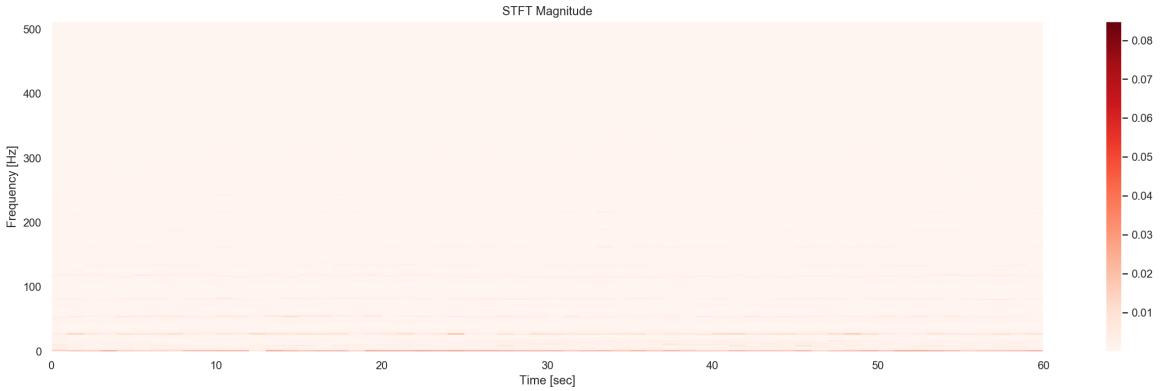


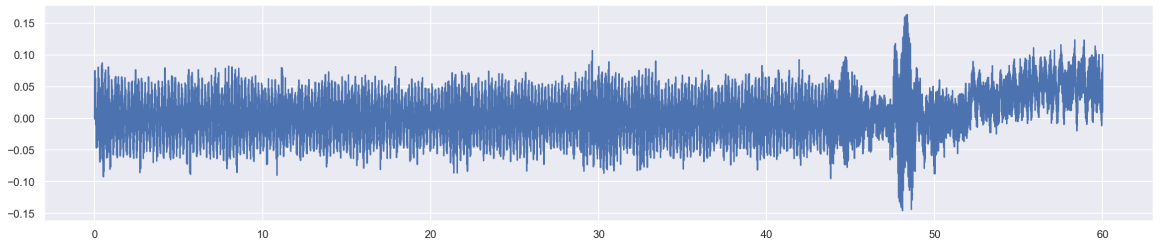

1st order devrivative of Spline
2nd order devrivative of Spline

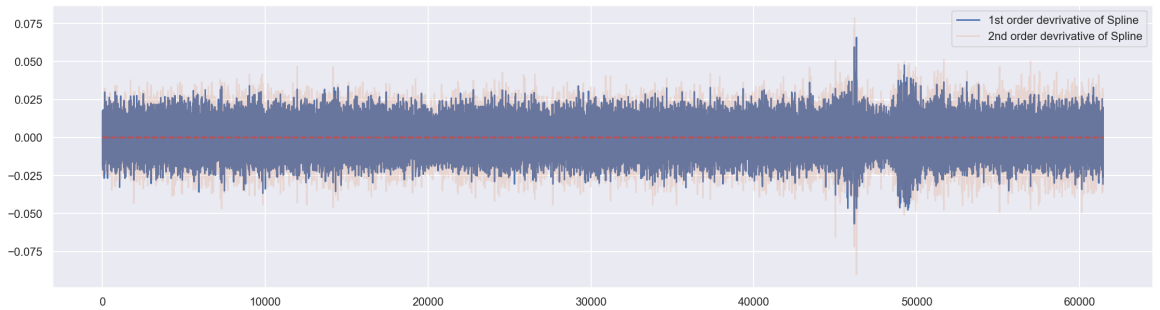

STFT Magnitude
Frequency [Hz]
Time [sec]

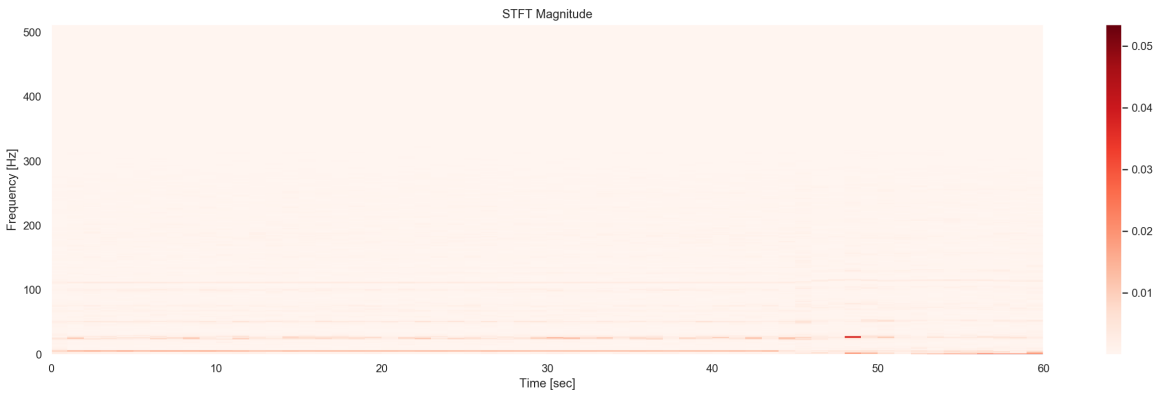


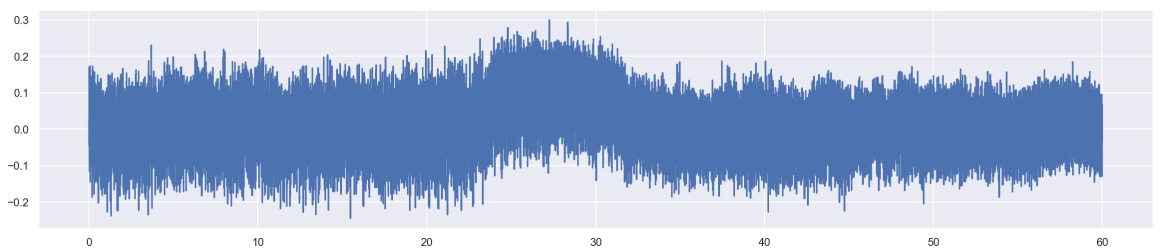

1st order devrivative of Spline
2nd order devrivative of Spline

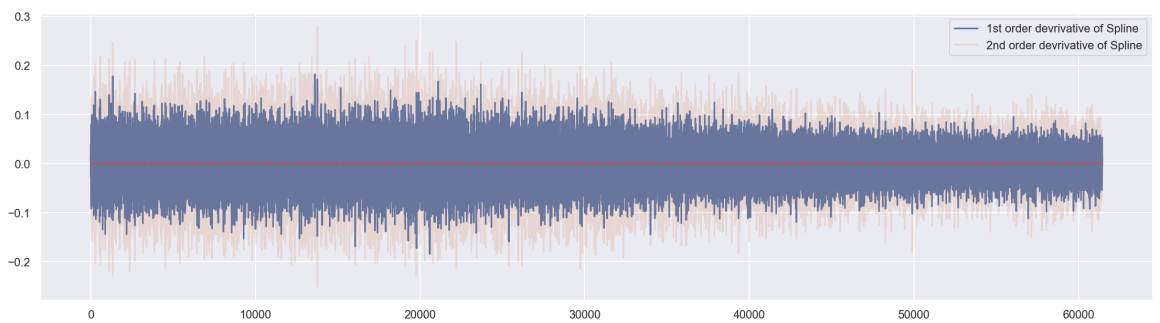

STFT Magnitude
Frequency [Hz]
Time [sec]

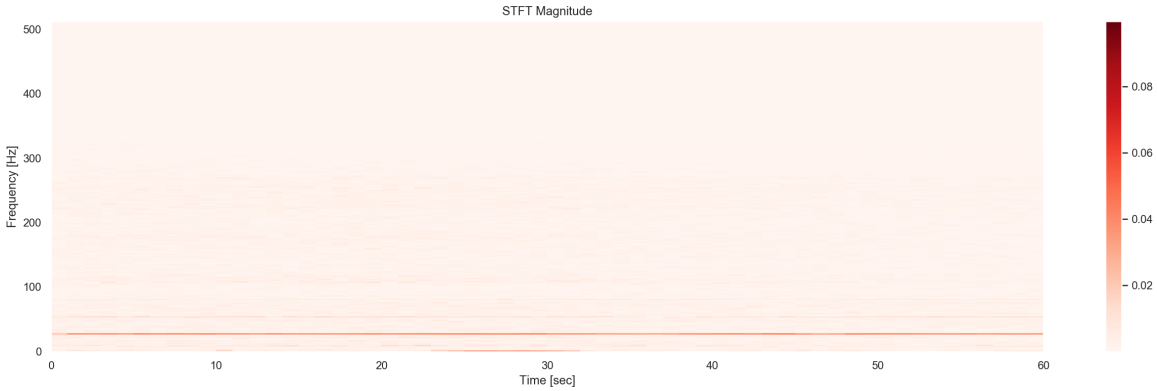


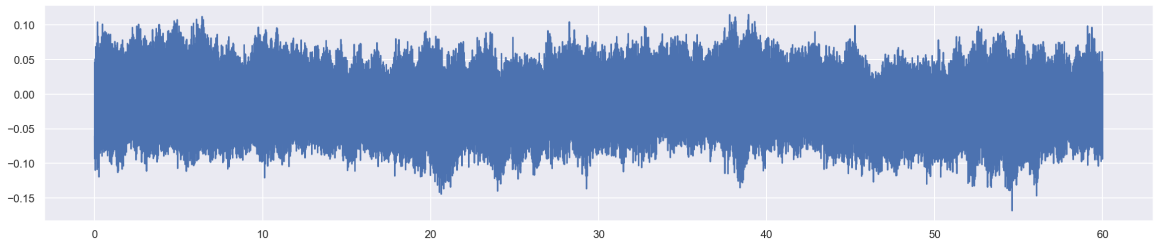

1st order devrivative of Spline
2nd order devrivative of Spline

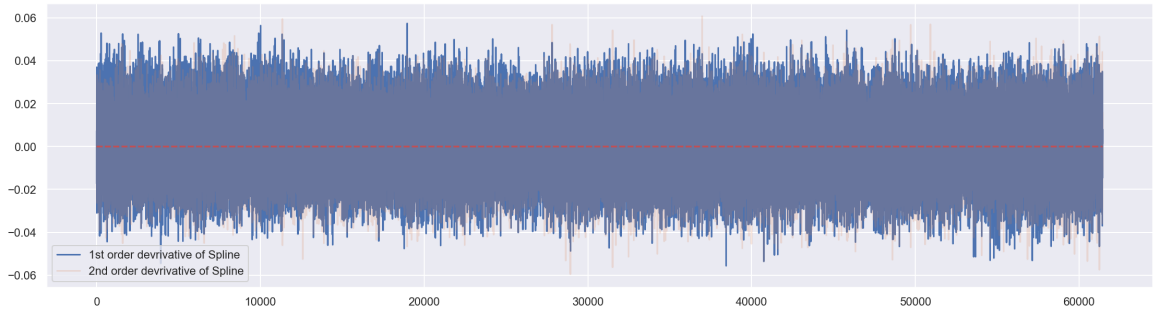

STFT Magnitude
Frequency [Hz]
Time [sec]

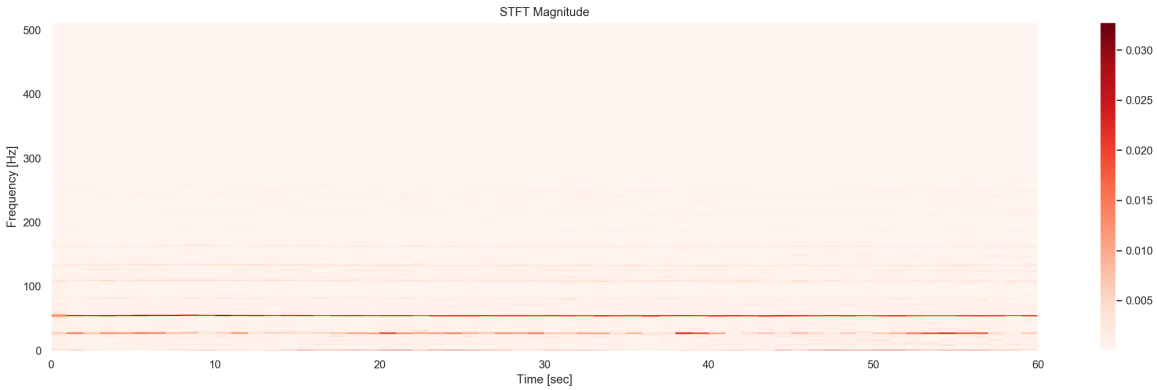


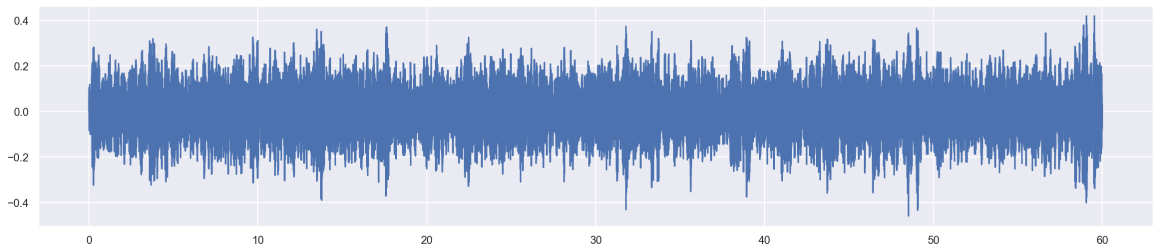

1st order devrivative of Spline
2nd order devrivative of Spline

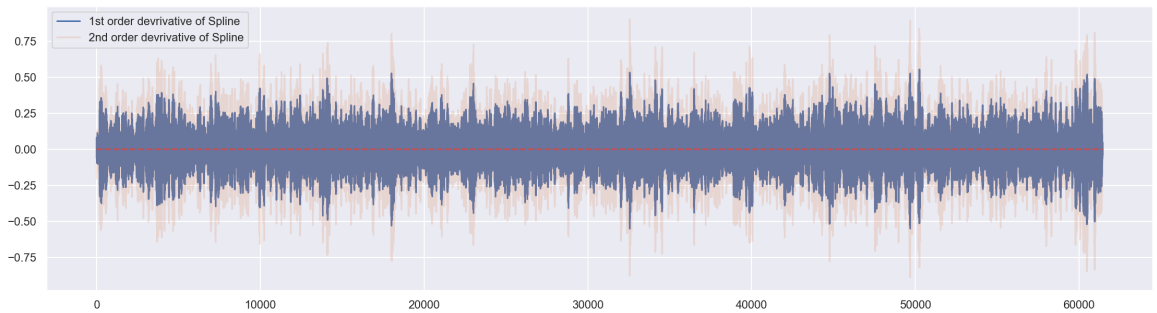

STFT Magnitude
Frequency [Hz]
Time [sec]

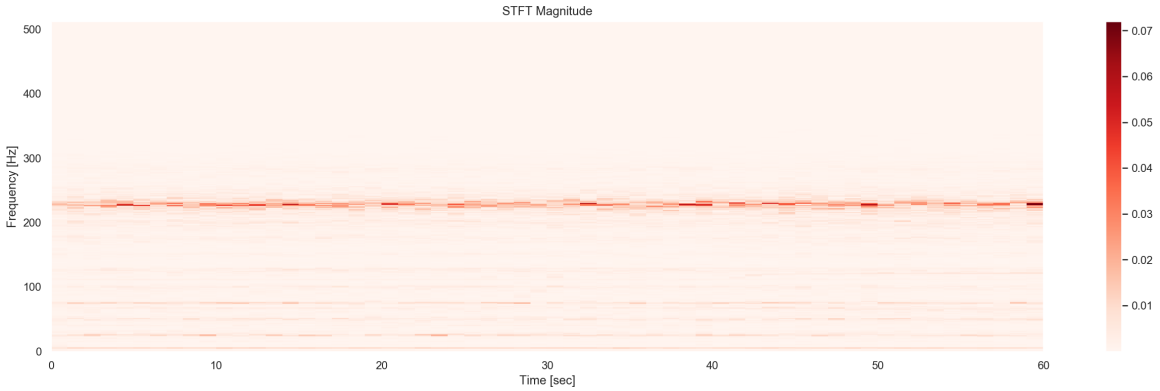


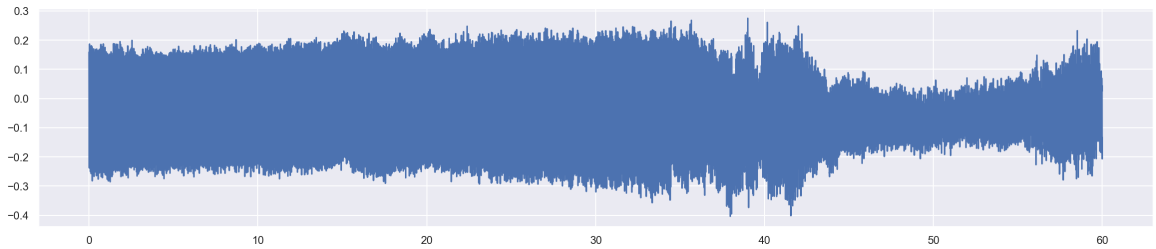

1st order devrivative of Spline
2nd order devrivative of Spline

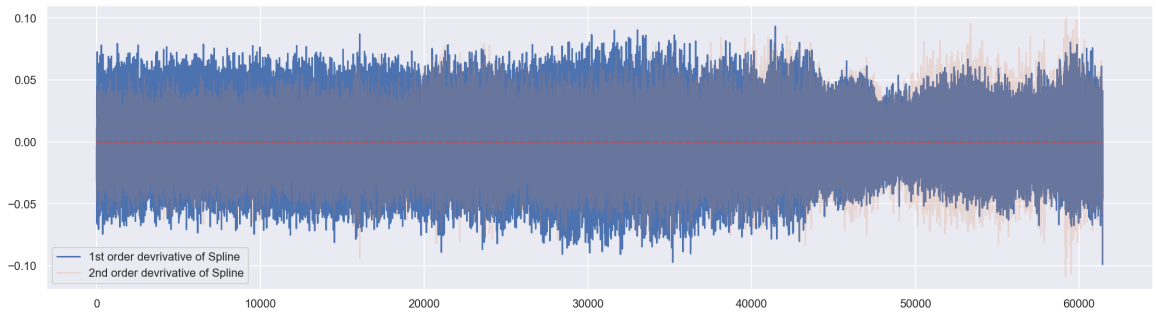

STFT Magnitude
Frequency [Hz]
Time [sec]

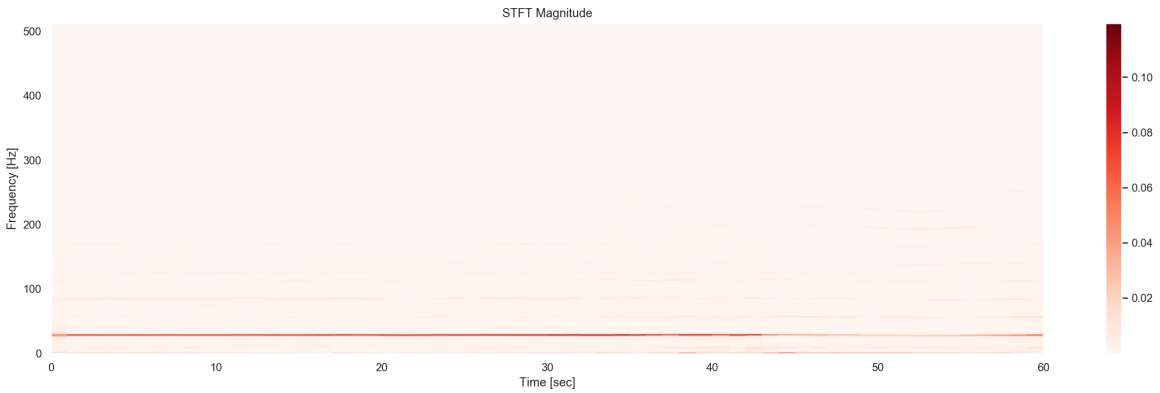


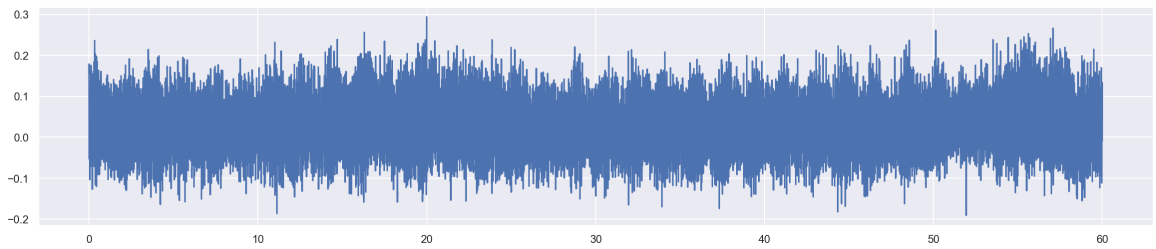

1st order devrivative of Spline
2nd order devrivative of Spline

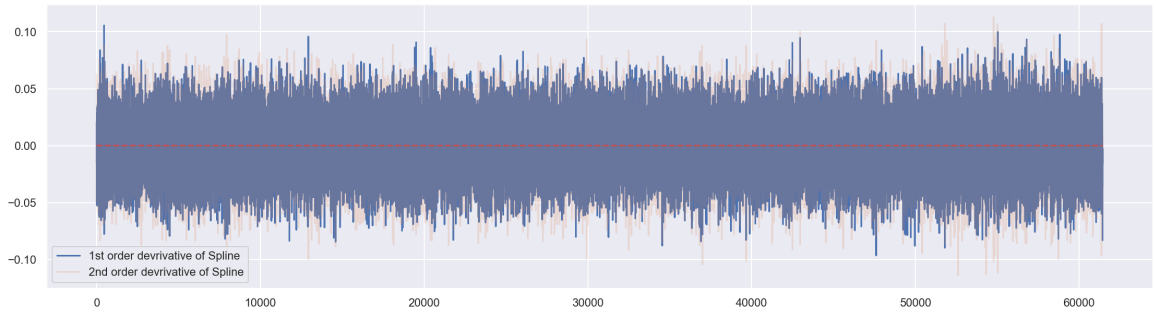

STFT Magnitude
Frequency [Hz]
Time [sec]

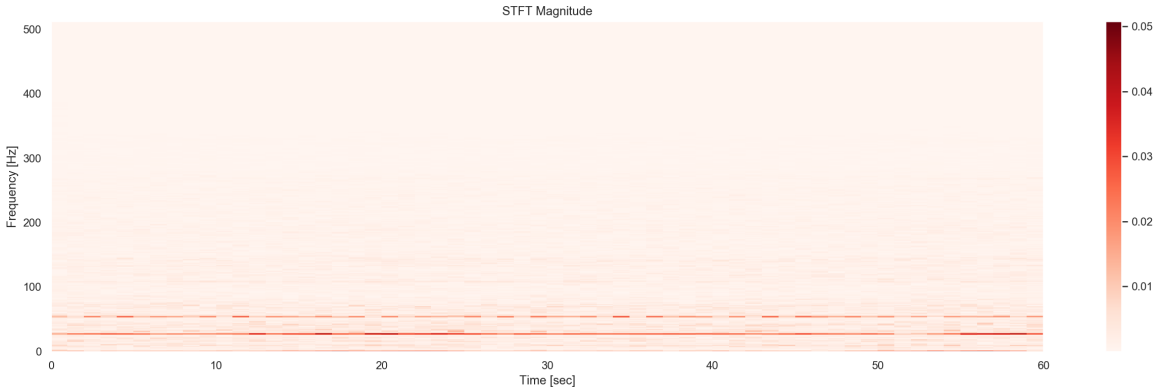


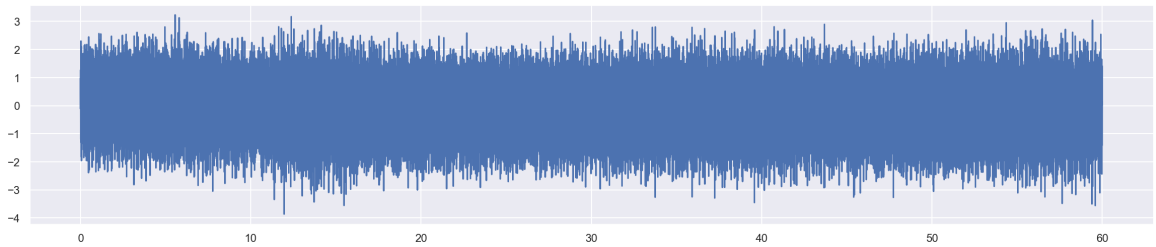

1st order devrivative of Spline
2nd order devrivative of Spline

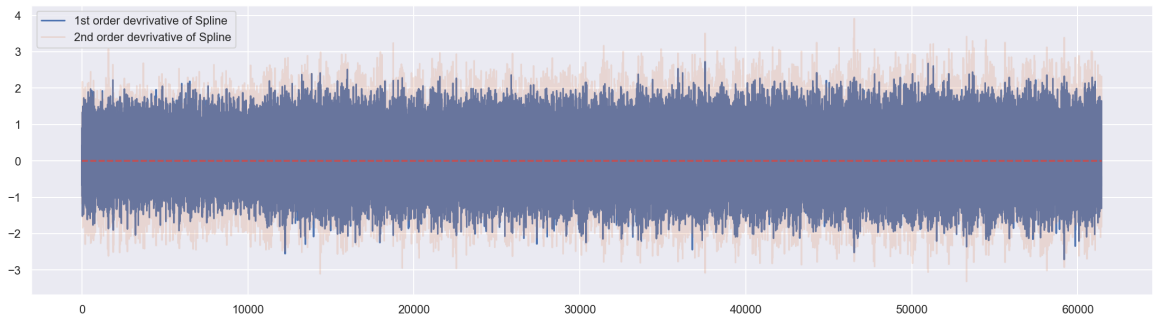

STFT Magnitude
Frequency [Hz]
Time [sec]

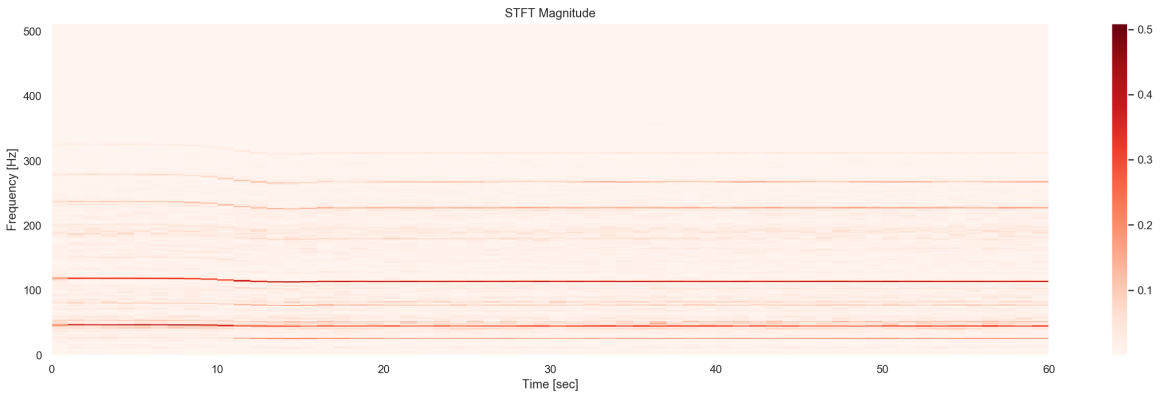


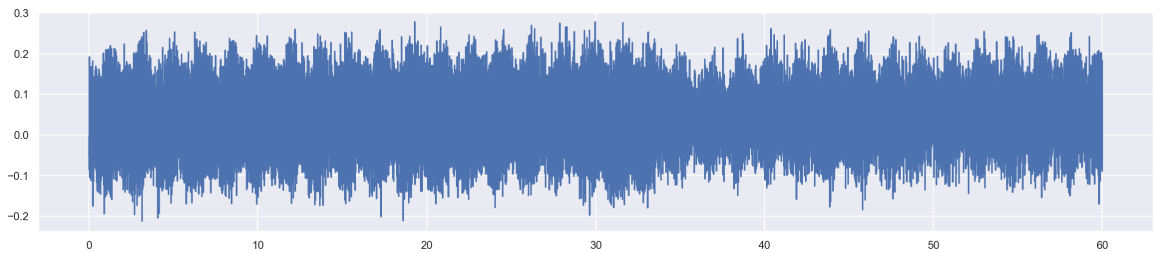

1st order devrivative of Spline
2nd order devrivative of Spline

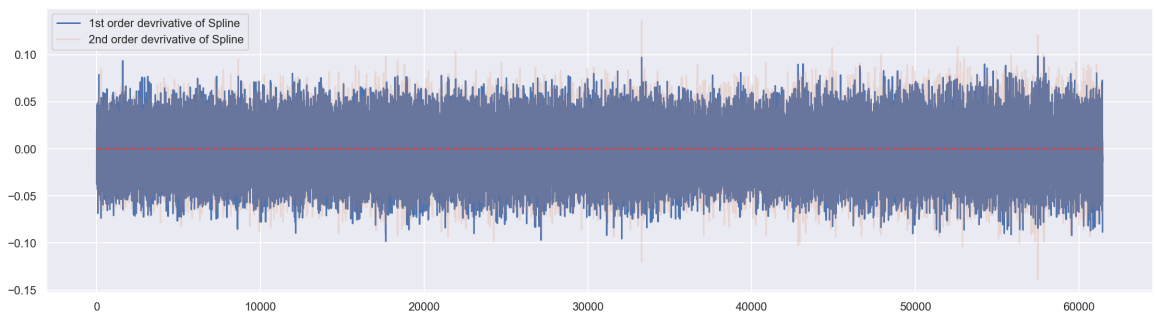

STFT Magnitude
Frequency [Hz]
Time [sec]

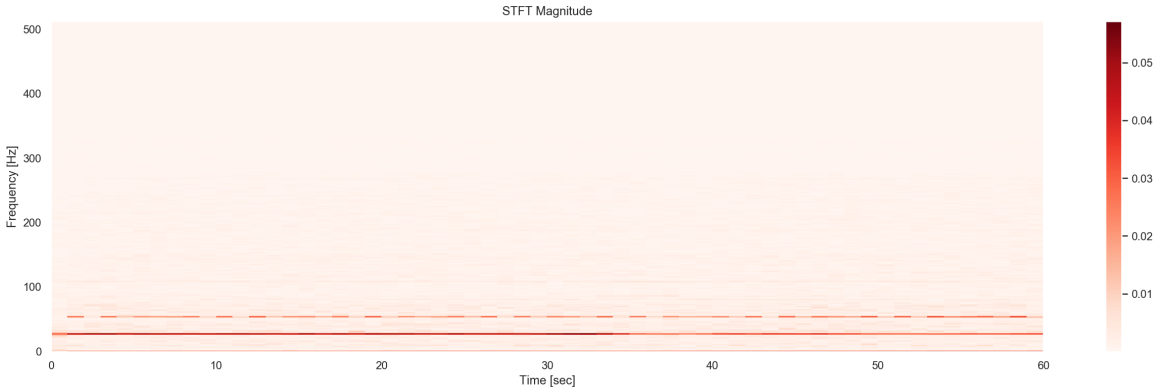


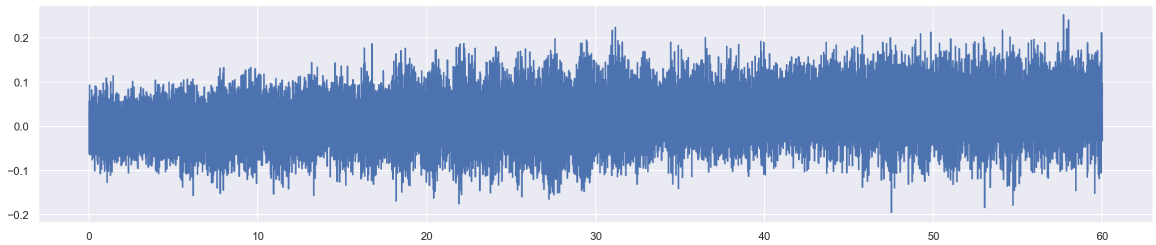

1st order devrivative of Spline
2nd order devrivative of Spline

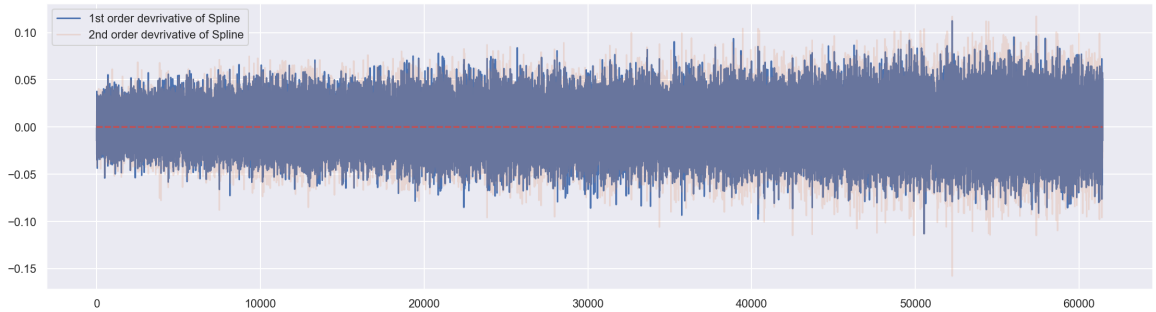

STFT Magnitude
Frequency [Hz]
Time [sec]

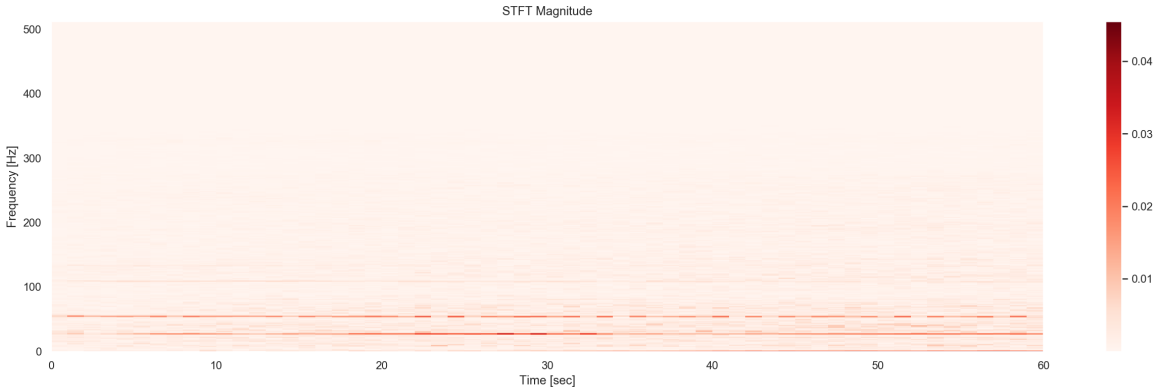


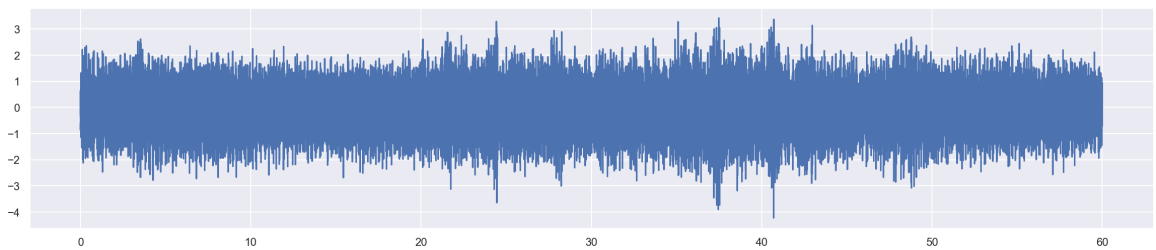

1st order devrivative of Spline
2nd order devrivative of Spline

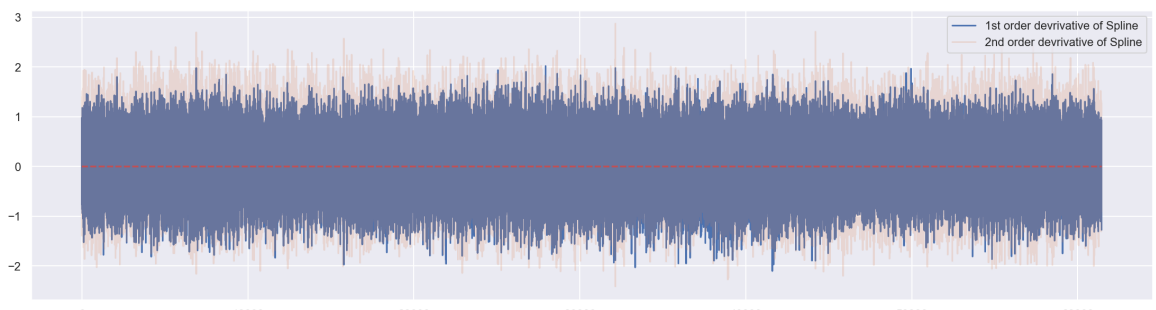

STFT Magnitude
Frequency [Hz]
Time [sec]

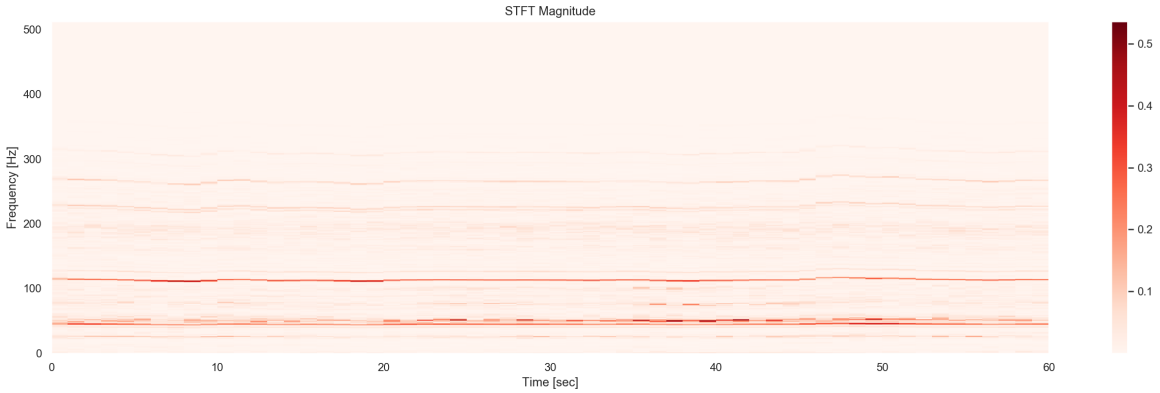


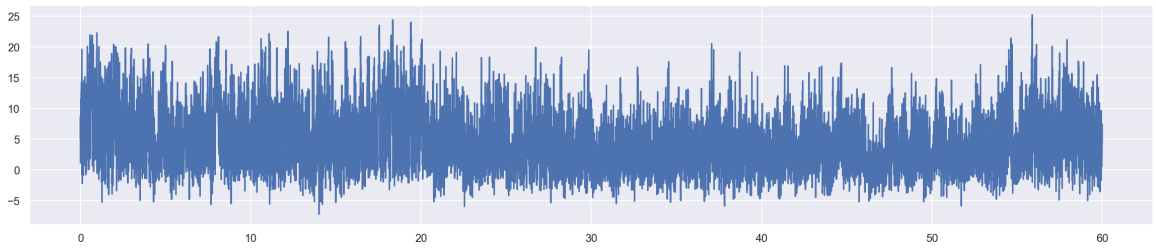

1st order devrivative of Spline
2nd order devrivative of Spline

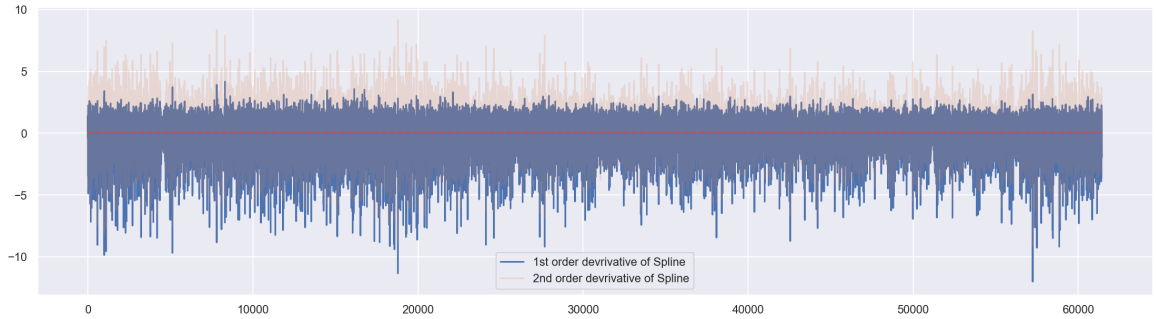

STFT Magnitude
Frequency [Hz]
Time [sec]

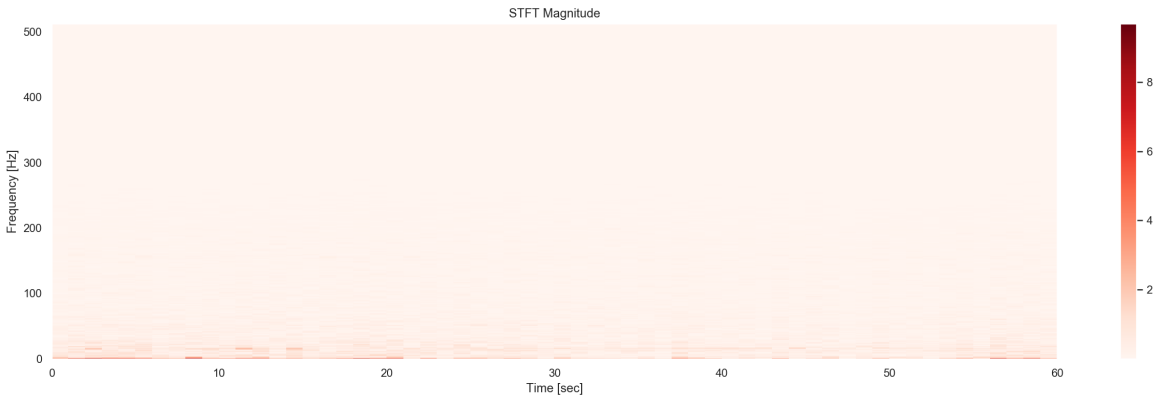


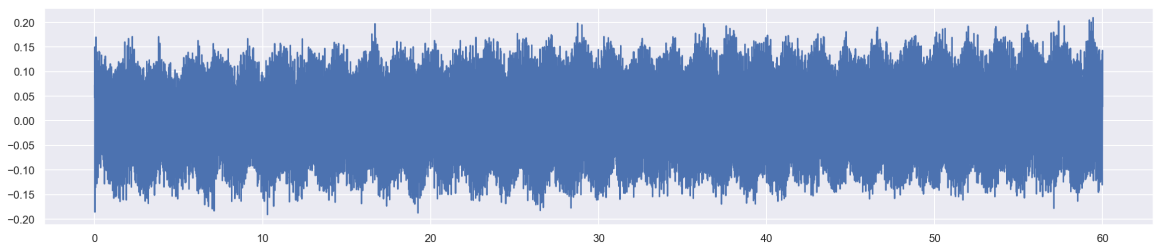

1st order devrivative of Spline
2nd order devrivative of Spline

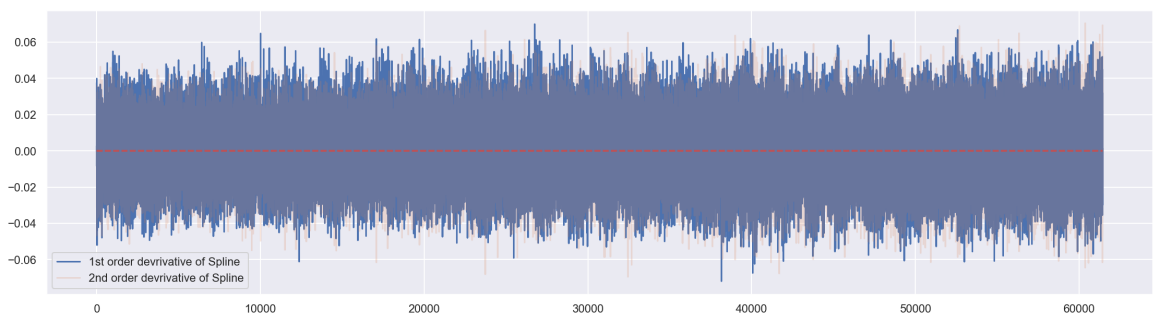

STFT Magnitude
Frequency [Hz]
Time [sec]

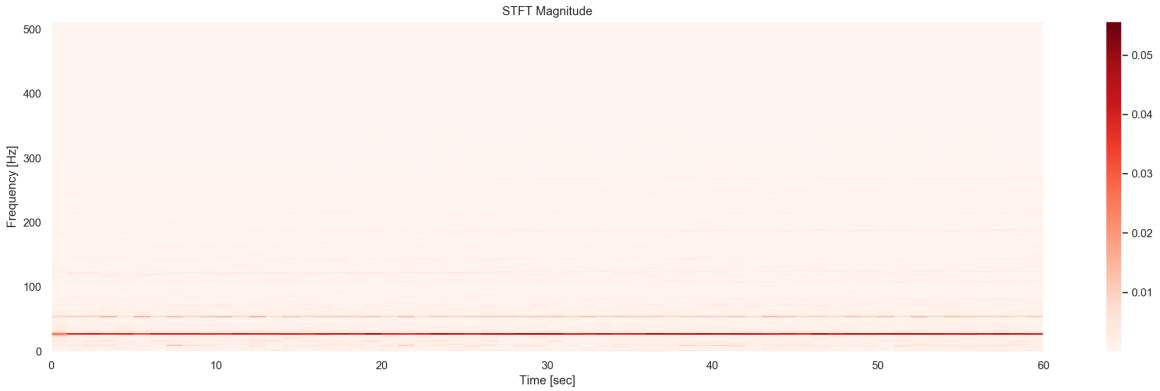


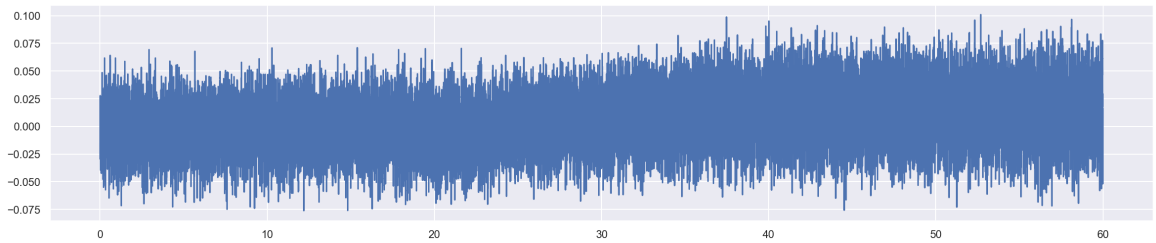

1st order devrivative of Spline
2nd order devrivative of Spline

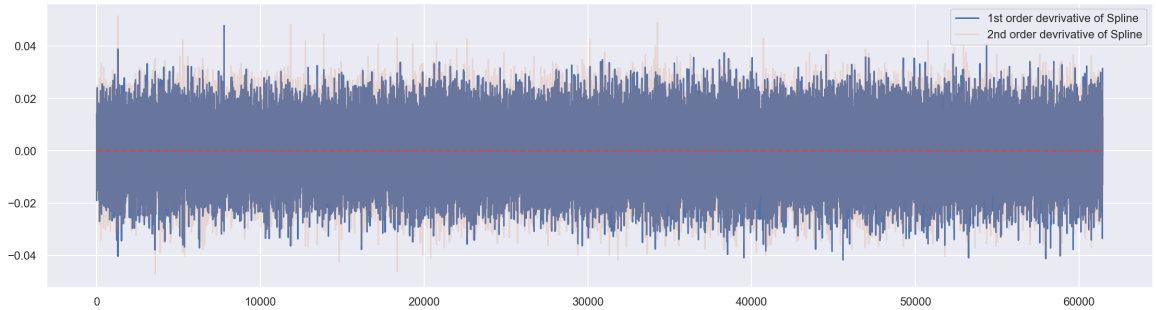

STFT Magnitude
Frequency [Hz]
Time [sec]

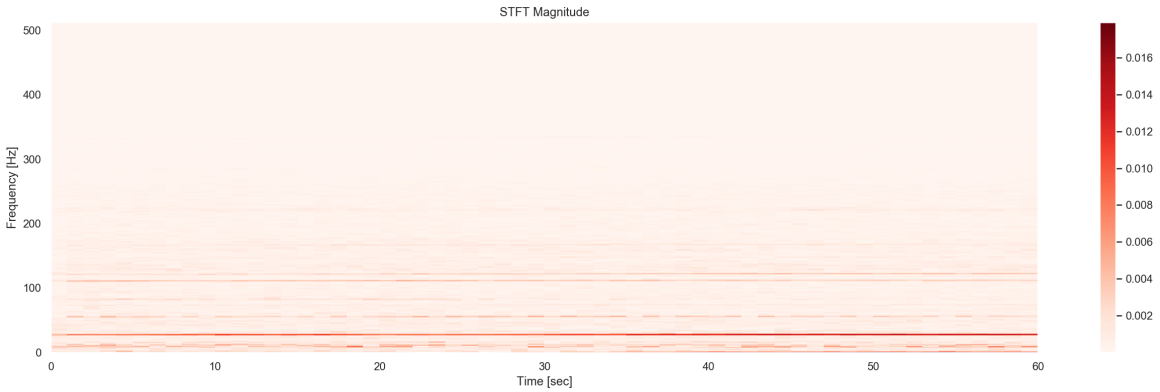


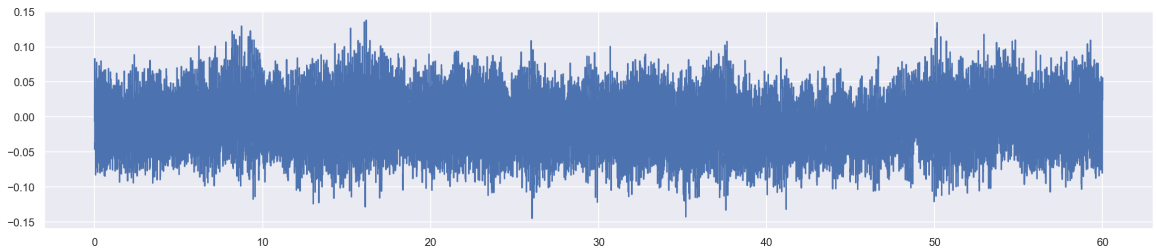

1st order devrivative of Spline
2nd order devrivative of Spline

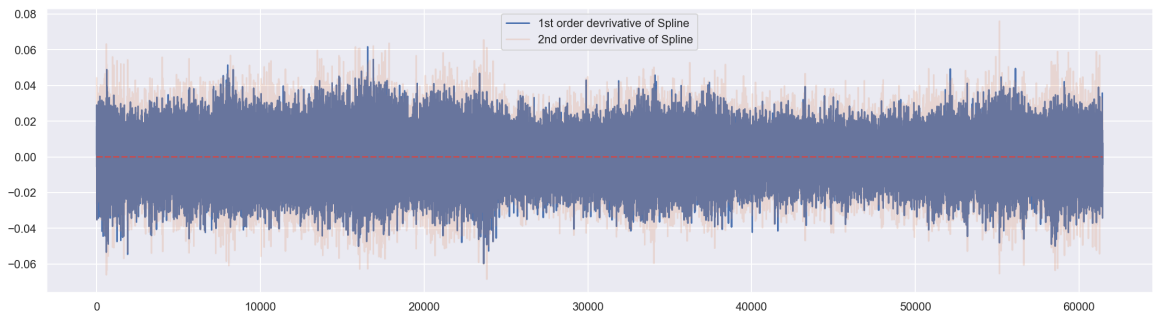

STFT Magnitude
Frequency [Hz]
Time [sec]

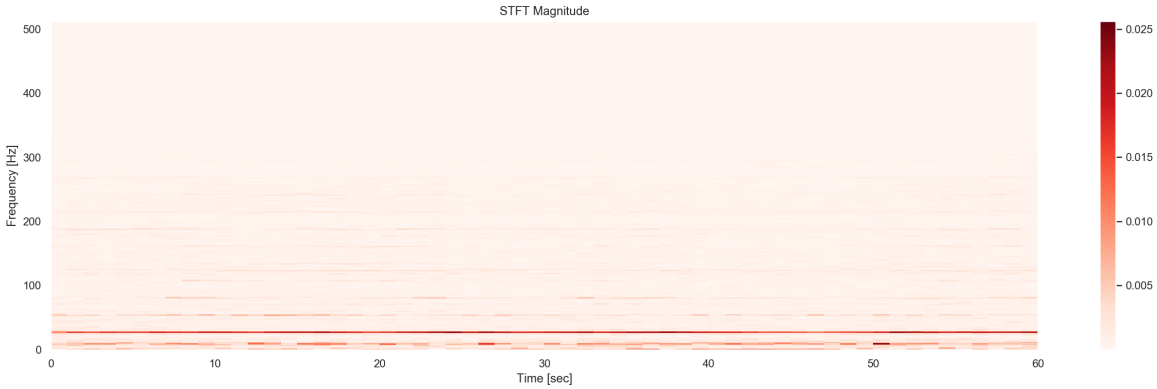


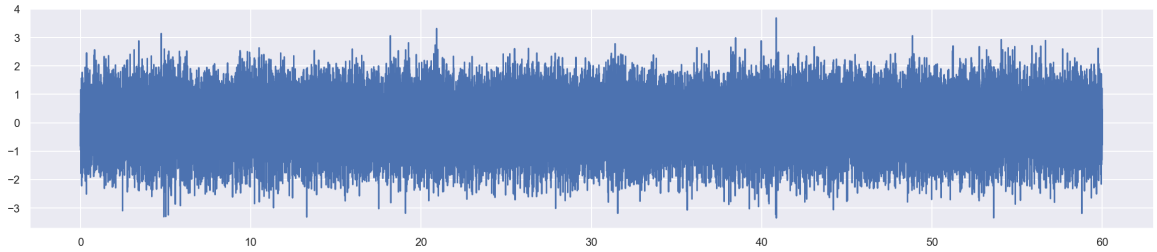

1st order devrivative of Spline
2nd order devrivative of Spline

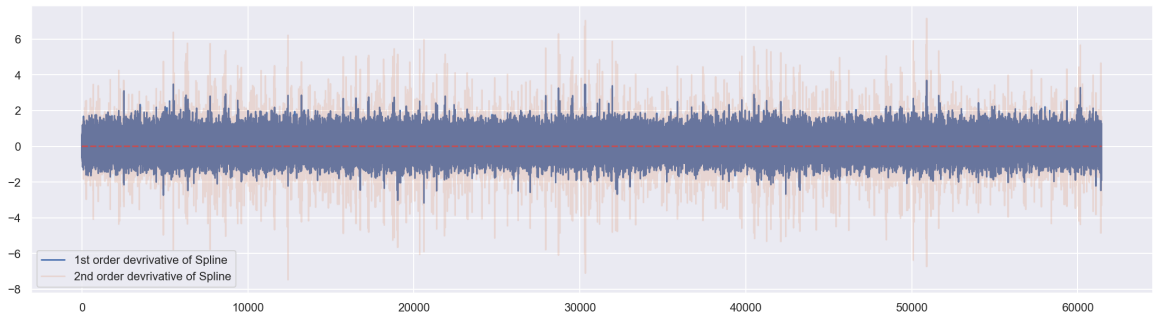

STFT Magnitude
Frequency [Hz]
Time [sec]

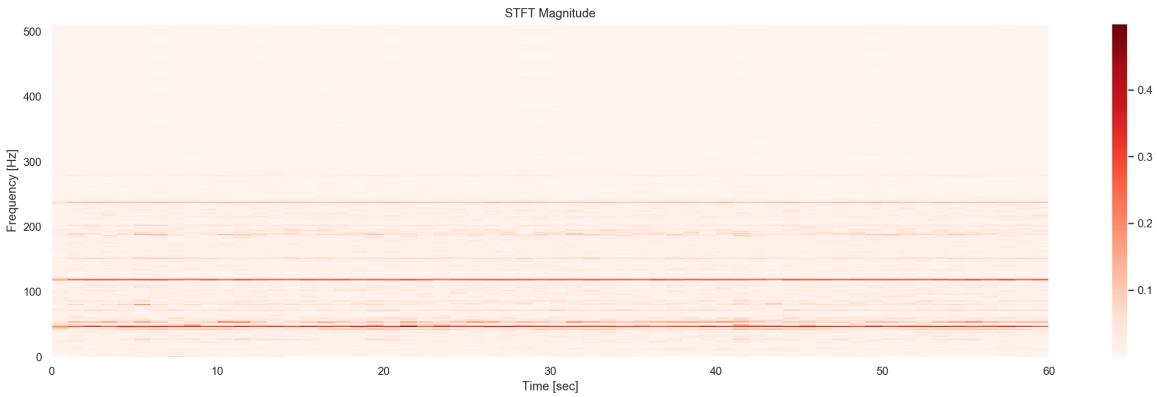


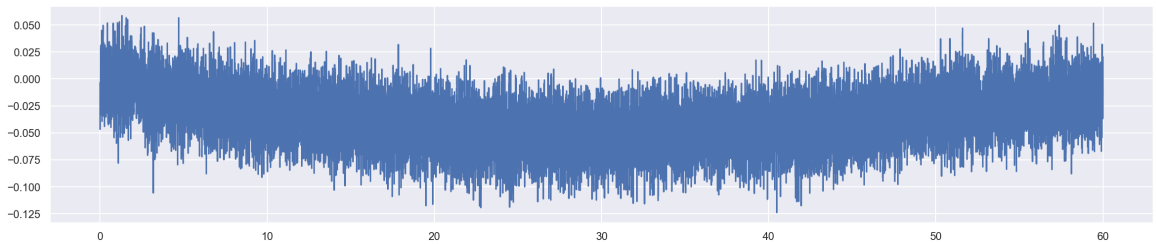

1st order devrivative of Spline
2nd order devrivative of Spline

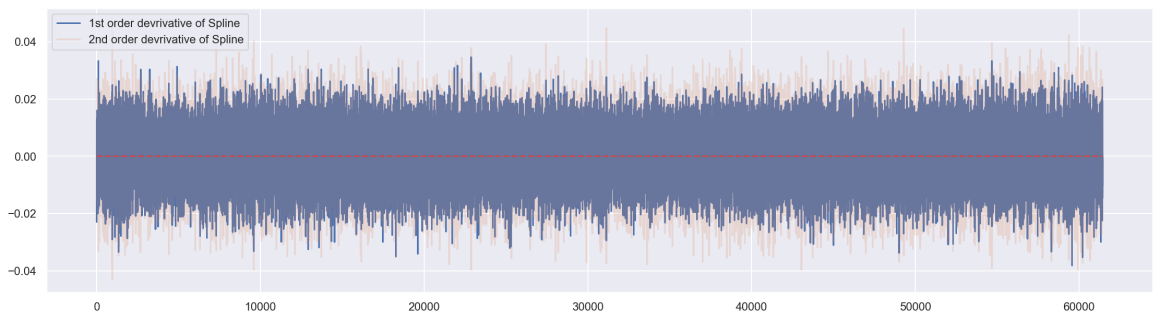

STFT Magnitude
Frequency [Hz]
Time [sec]

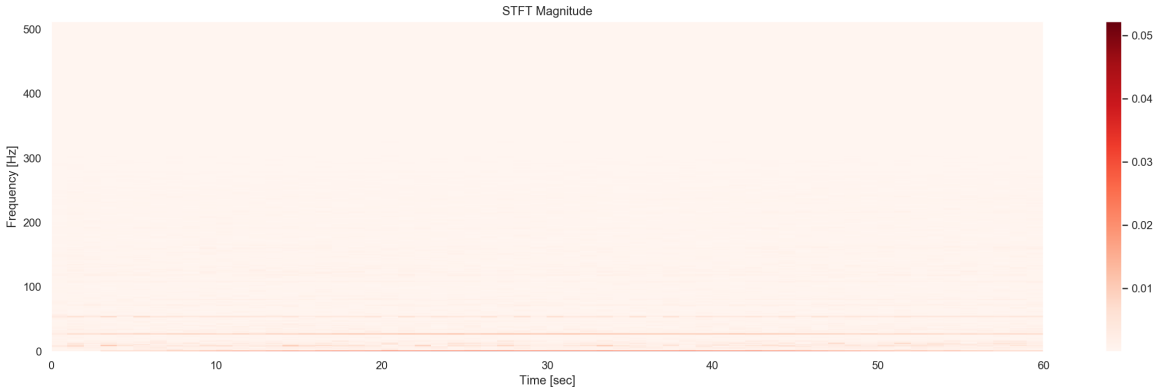



1st order devrivative of Spline
2nd order devrivative of Spline


STFT Magnitude
Frequency [Hz]
Time [sec]




1st order devrivative of Spline
2nd order devrivative of Spline


STFT Magnitude
Frequency [Hz]
500
400
300
200
100
0
0
10
20
30
40
50
60
Time [sec]
0.05
0.04
0.03
0.02
0.01

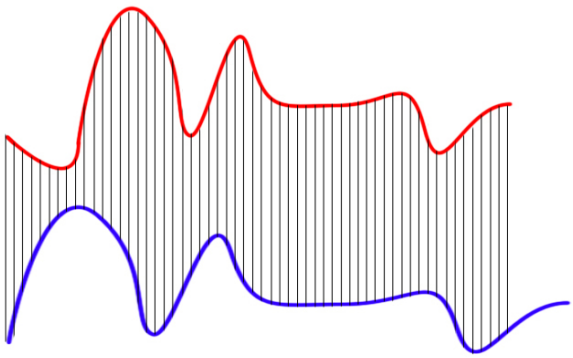
## Time Series Clustering

## Similarity

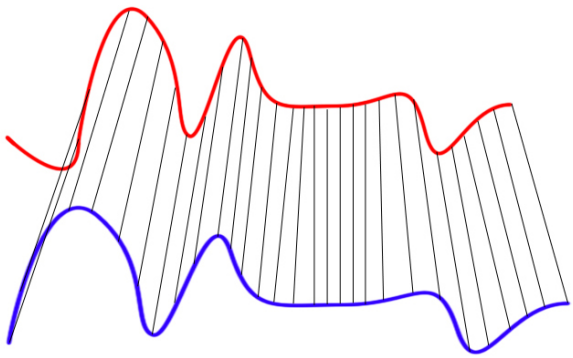
The objective of time series comparison methods is to produce a distance metric between two input time series. The similarity or dissimilarity of two-time series is typically calculated by converting the data into vectors and calculating the Euclidean distance between those points in vector space.

## DTW (Dynamic Time Warping)

In time series analysis, dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. For instance, similarities in walking could be detected using DTW, even if one person was walking faster than the other, or if there were accelerations and decelerations during the course of an observation.



Euclidean Matching



Dynamic Time Warping Matching

Clearly these two series follow the same pattern, but the blue curve is longer than the red. If we apply the one-to-one match, shown in the top, the mapping is not perfectly synced up and the tail of the blue curve is being left out. DTW overcomes the issue by developing a one-to-many match so that the troughs and peaks with the same pattern are perfectly matched, and there is no left out for both curves(shown in the bottom top).

In [142]:

```
def dtw(s, t, window):
    n, m = len(s), len(t)
    w = np.max([window, abs(n-m)])
    dtw_matrix = np.zeros((n+1, m+1))

    for i in range(n+1):
        for j in range(m+1):
            dtw_matrix[i, j] = np.inf
    dtw_matrix[0, 0] = 0

    for i in range(1, n+1):
        for j in range(np.max([1, i-w]), np.min([m, i+w])+1):
            dtw_matrix[i, j] = 0

    for i in range(1, n+1):
        for j in range(np.max([1, i-w]), np.min([m, i+w])+1):
            cost = abs(s[i-1] - t[j-1])
            # take last min from a square box
            last_min = np.min(
                [dtw_matrix[i-1, j], dtw_matrix[i, j-1], dtw_matrix[i-1, j-1]])
            dtw_matrix[i, j] = cost + last_min
    return dtw_matrix
```

Python implementation of FastDTW, which is an approximate Dynamic Time Warping (DTW) algorithm that provides optimal or near-optimal alignments with an O(N) time and memory complexity.

In [144]:

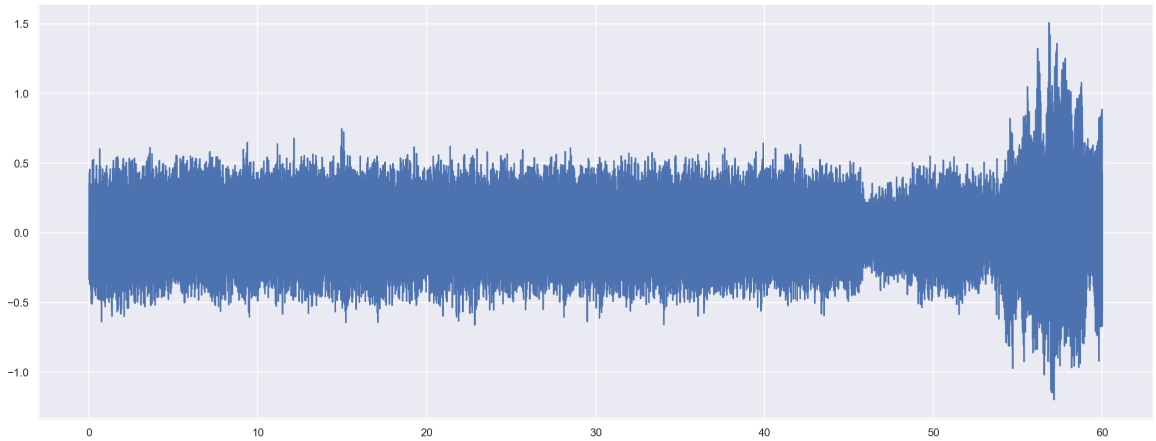
```
from fastdtw import fastdtw
from scipy.spatial.distance import euclidean

x=xtrain[1]
y=xtrain[2]

distance, path = fastdtw(x, y, dist=euclidean)

print(distance)
print(path[:20])
```

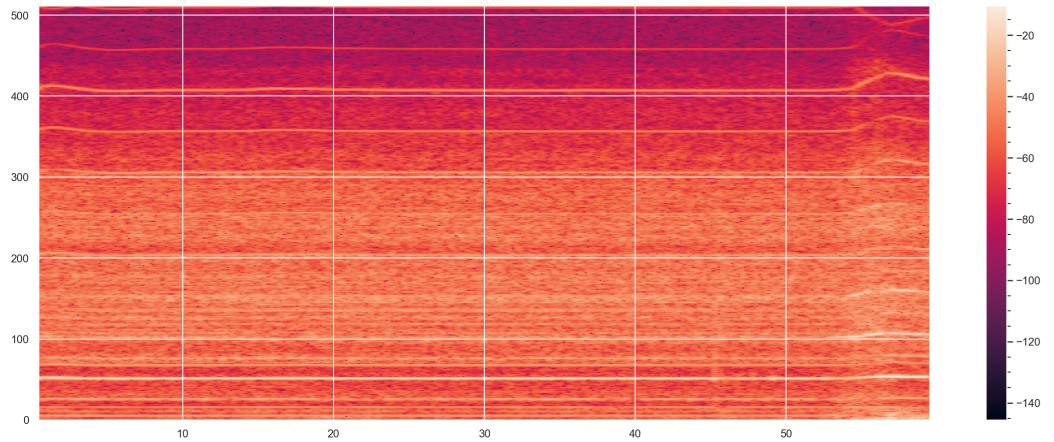
```
18174.91575999992
[(0, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8,
1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (1
6, 2), (17, 3), (18, 3), (19, 4)]
```

## Periodogram-based distance

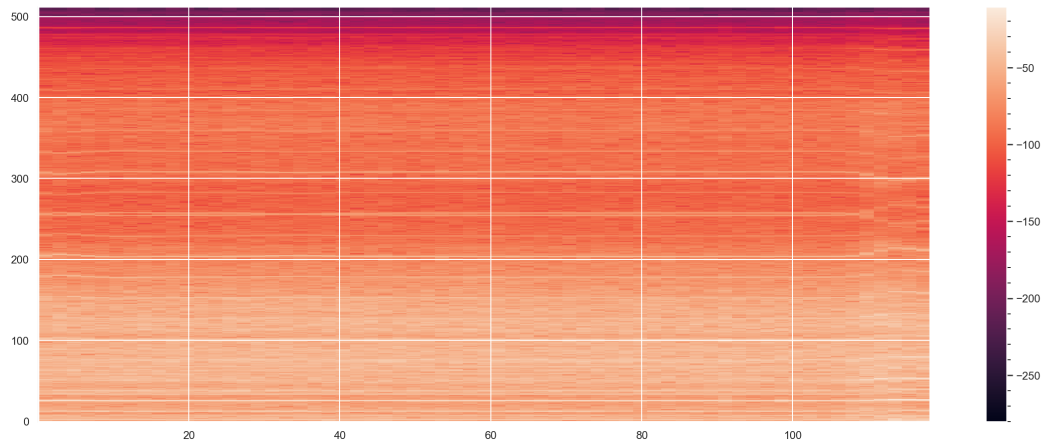
In [158]:

```
def draw_spectrum(x, dt=1/1024):
    dt = 1/1024
    t = np.arange(0.0, 60.0, dt)
    fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, figsize=(20, 26))

    ax1.plot(t, x)
    # raw signal
    NFFT = 1024  # the length of the windowing segments
    Fs1 = int(1.0 / dt)  # the sampling frequency
    spectrum1, freqs1, bins1, im1 = ax2.specgram(
        x, NFFT=NFFT, Fs=Fs1, noverlap=900, mode='magnitude')  # mode : {'defaul
t', 'psd', 'magnitude', 'angle', 'phase'}
    cbar = fig.colorbar(im1, ax=ax2)
    cbar.minorticks_on()

    # the interpolated data
    i = 2
    tck = interpolate.splrep(range(len(x)), x, s=0)
    xnew = np.arange(0, 61440, 1/i)
    x2 = interpolate.splev(xnew, tck, der=0)
    NFFT = 1024*i  # the length of the windowing segments
    Fs2 = int(1.0 * 1 / dt)  # the sampling frequency

    spectrum2, freqs2, bins2, im2 = ax3.specgram(
        x2, NFFT=NFFT, Fs=Fs2, noverlap=128, mode='magnitude')
    cbar2 = fig.colorbar(im2, ax=ax3)
    cbar2.minorticks_on()

    return spectrum1, freqs1, spectrum2, freqs2


spectrum1, freqs1, spectrum2, freqs2 = draw_spectrum(xtrain[6])
```


1.5
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
60


500
400
300
200
100
0
10
20
30
40
50


−20
−40
−60
−80
−100
−120
−140


500
400
300
200
100
0
20
40
60
80
100


−50
−100
−150
−200
−250

In [152]:

```
pd.DataFrame(np.log2(spectrum1).T).describe()
```

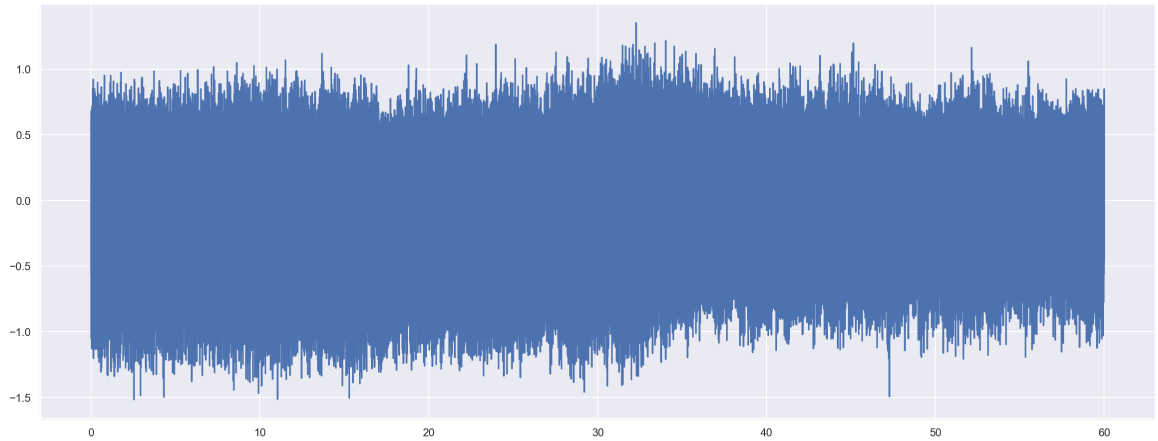
Out[152]:

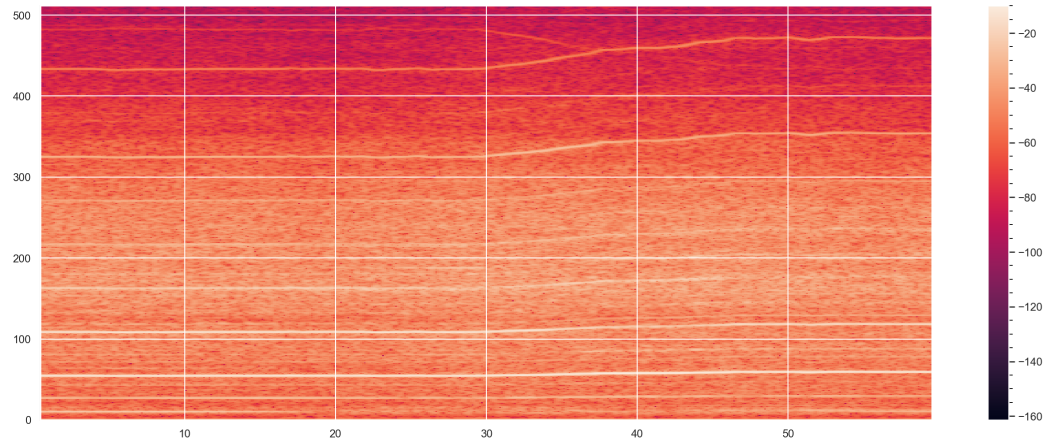
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **count** | 119.000000 | 119.000000 | 119.000000 | 119.000000 | 119.000000 | 119.000000 | 119.000000 |
| **mean** | -10.099309 | -9.229992 | -8.310577 | -8.269961 | -6.969925 | -6.036250 | -6.859622 |
| **std** | 1.637636 | 1.690498 | 1.530460 | 1.530745 | 0.588311 | 0.479357 | 0.537620 |
| **min** | -14.861751 | -13.424947 | -10.896369 | -12.536328 | -8.280608 | -9.664204 | -10.546948 |
| **25%** | -10.906036 | -10.015844 | -9.158851 | -9.129029 | -7.344481 | -6.133416 | -7.037854 |
| **50%** | -10.157486 | -9.345526 | -8.579695 | -8.360844 | -7.008360 | -6.026244 | -6.896915 |
| **75%** | -9.439787 | -8.724738 | -7.822520 | -7.675591 | -6.673912 | -5.878141 | -6.739346 |
| **max** | -5.171843 | -3.466379 | -2.784594 | -3.422106 | -4.773168 | -4.078747 | -3.896027 |

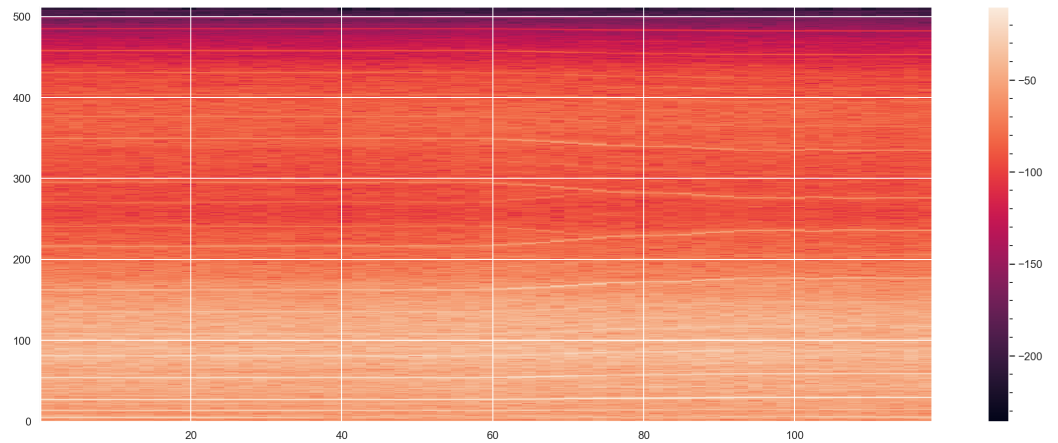
8 rows × 513 columns

In [159]:

```
spectrum1, freqs1, spectrum2, freqs2 = draw_spectrum(xtrain[157])
```


1.0
0.5
0.0
−0.5
−1.0
−1.5
0
10
20
30
40
50
60


500
400
300
200
100
0
10
20
30
40
50
−20
−40
−60
−80
−100
−120
−140
−160


500
400
300
200
100
0
20
40
60
80
100
−50
−100
−150
−200

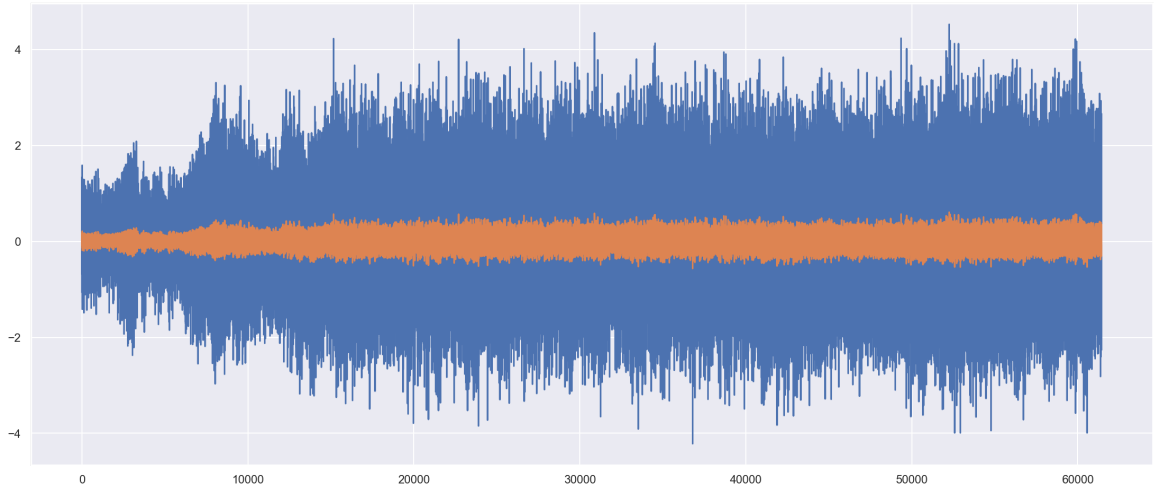
# First order/seconde order derivative and other feature engenieering

In [145]:

```
# first order derivative and seconde order derivative
yder1 = interpolate.splev(xnew, tck, der=1)
yder2 = interpolate.splev(xnew, tck, der=2)
plt.figure(figsize=(19, 8))

# plot Derivative of spline
plt.plot(xnew, yder1, label='Cubic Spline devrivative')
plt.plot(xnew, yder2, label='Cubic Spline devrivative')

# spline
plt.scatter(xnew, ynew)
plt.plot(range(length), xtrain[i, start:end], label='raw data')

# plot axe y=0
plt.plot(xnew, np.zeros(len(xnew)), 'r--')
#plt.axis([-0.05, 6.33, -1.05, 1.05])
plt.title('Derivative estimation from spline')
plt.legend()
plt.show()
```

In [148]:

```
plt.figure(figsize=(19,8))
i=1
plt.plot(range(61440),xtrain[i+8,:61440]/abs(xtrain[i+8,:61440]).mean())
plt.plot(range(61440),xtrain[i+8,:61440])
plt.show()
```




## Build fonctionnal space

The Nyquist–Shannon sampling theorem is a theorem in the field of digital signal processing which serves as a fundamental bridge between continuous-time signals and discrete-time signals. It establishes a sufficient condition for a sample rate that permits a discrete sequence of samples to capture all the information from a continuous-time signal of finite bandwidth.

If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.

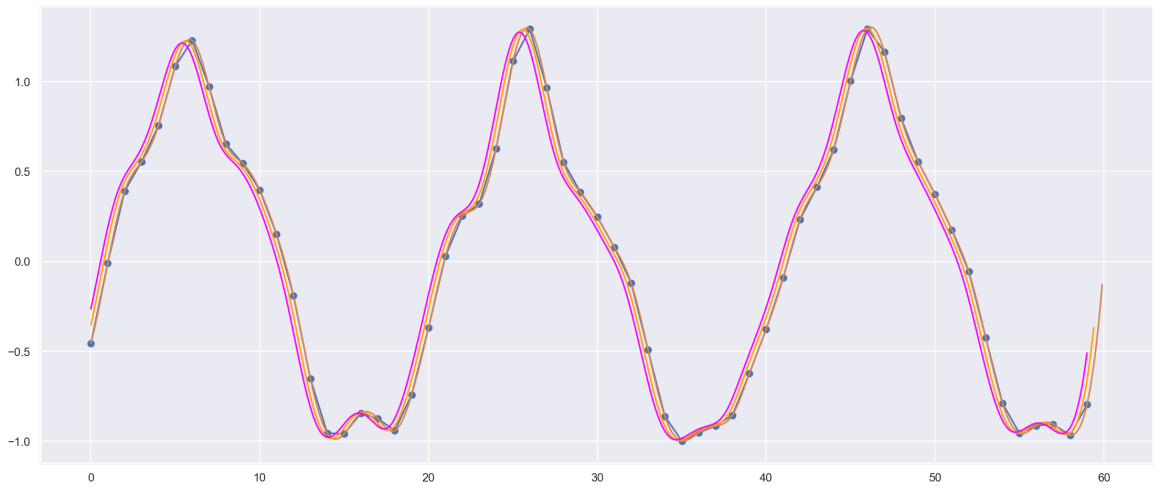
A sufficient sample-rate is therefore anything larger than $2B$ samples per second. Equivalently, for a given sample rate $f_s$, perfect reconstruction is guaranteed possible for a bandlimit $B < f_s/2$.

## Interpolation - Spline

In mathematics, a spline is a special function defined piecewise by polynomials. In interpolating problems, spline interpolation is often preferred to polynomial interpolation because it yields similar results, even when using low degree polynomials, while avoiding Runge's phenomenon for higher degrees.

In [113]:

```
i = 10
k = 1
start = 0
end = 60
length = end-start
ynew = []

def rolling_window(a, window):
    shape = a.shape[:-1] + (a.shape[-1] - window + 1, window)
    strides = a.strides + (a.strides[-1],)
    return np.lib.stride_tricks.as_strided(a, shape=shape, strides=strides)

plt.figure(figsize=(19, 8))
for n in range(k):
    # spline interpolation
    tck = interpolate.splrep(range(start, end), xtrain[i+n, start:end], s=0)
    xnew = np.arange(start, end, 1/10)
    ynew = interpolate.splev(xnew, tck, der=0)

    # raw data
    plt.scatter(range(start, end), xtrain[i+n, start:end])
    plt.plot(range(start, end), xtrain[i+n, start:end])

    # spline
    # plt.scatter(xnew,ynew)
    plt.plot(xnew, ynew)

    # SMA pandas rolling
    #rolling_mean = ynew.rolling(window=20).mean()
    #rolling_mean2 = ynew.rolling(window=50).mean()

    # SMA numpy
    rolling_mean1 = np.mean(rolling_window(ynew, 6), -1)
    rolling_mean2 = np.mean(rolling_window(ynew, 10), -1)

    plt.plot(xnew[:len(rolling_mean1)], rolling_mean1,
             label='AMD 20 Day SMA', color='orange')
    plt.plot(xnew[:len(rolling_mean2)], rolling_mean2,
             label='AMD 50 Day SMA', color='magenta')
```




## 1st , 2nd order derivative

In [116]:

```
yder1 = interpolate.splev(xnew, tck, der=1)
yder2 = interpolate.splev(xnew, tck, der=2)

plt.figure(figsize=(19, 7))
# spline
plt.scatter(xnew, ynew)
plt.plot(xnew, ynew, label='Cubic Spline interpolation ')

# plot Derivative of spline
plt.plot(xnew, yder1, label='1st order devrivative of Spline')
plt.plot(xnew, np.zeros(len(xnew)), 'r--')

# plot Derivative of spline
plt.plot(xnew, yder2, label='2nd order devrivative of Spline')
plt.plot(xnew, np.zeros(len(xnew)), 'r--')
plt.legend()
```

Out[116]:

```
<matplotlib.legend.Legend at 0x1707b7f98>
```

In [106]:

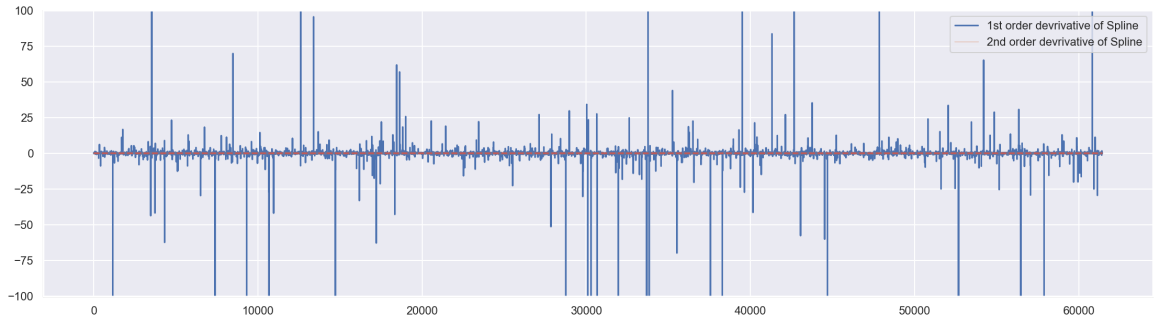
```
i=10
k=1

def plot_derive(x):
    #spline interpolation
    tck = interpolate.splrep(range(61440),x, s=0)
        #xnew = np.linspace(start,end,length*10)
    xnew = np.arange(0,61440,1/10)
    ynew = interpolate.splev(xnew, tck, der=0)

    yder1 = interpolate.splev(xnew, tck, der=1)
    yder2 = interpolate.splev(xnew, tck, der=2)

    plt.figure(figsize=(19,5))
    #plt.scatter(xnew,ynew)
    #plt.plot(xnew,ynew,label='Cubic Spline interpolation ')

    #plot Derivative of spline
    plt.plot(xnew, yder1,label='1st order devrivative of Spline')
    # plot axe y=0
    plt.plot(xnew,np.zeros(len(xnew)),'r--')


    #plot Derivative of spline
    plt.plot(xnew, yder2,label='2nd order devrivative of Spline',alpha=0.2)
    # plot axe y=0
    plt.plot(xnew,np.zeros(len(xnew)),'r--')
    plt.legend()
    plt.show()

plot_derive(xtrain[i])
```

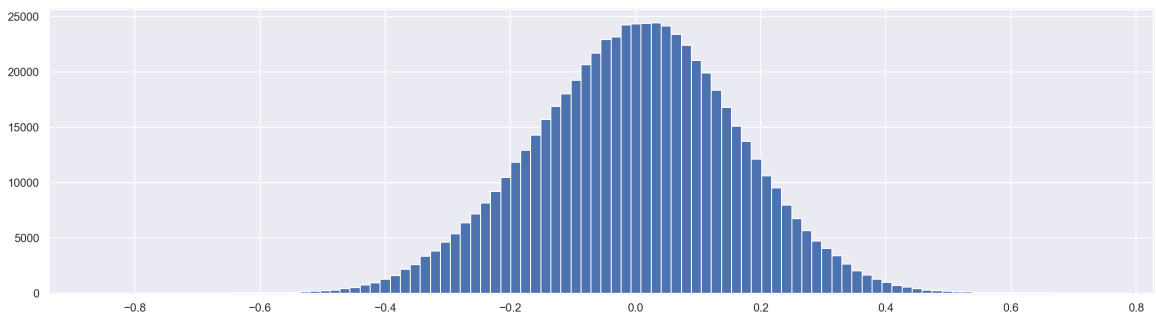
```
//anaconda3/lib/python3.7/site-packages/IPython/core/pylabtools.py:1
28: UserWarning: Creating legend with loc="best" can be slow with la
rge amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

In [110]:

```
def plot_derive(x):
    #spline interpolation
    tck = interpolate.splrep(range(61440),x, s=0)
        #xnew = np.linspace(start,end,length*10)
    xnew = np.arange(0,61440,1/10)
    ynew = interpolate.splev(xnew, tck, der=0)
    yder1 = interpolate.splev(xnew, tck, der=1)
    yder2 = interpolate.splev(xnew, tck, der=2)

    plt.figure(figsize=(19,5))
    plt.ylim(-100, 100)
    #plt.scatter(xnew,ynew)
    #plt.plot(xnew,ynew,label='Cubic Spline interpolation ')

    #plot Derivative of spline
    plt.plot(xnew, yder1/ynew/100,label='1st order devrivative of Spline')
    plt.plot(xnew,np.zeros(len(xnew)),'r--')

    #plot Derivative of spline
    plt.plot(xnew, yder2,label='2nd order devrivative of Spline',alpha=0.2)
    plt.legend()

    plt.figure(figsize=(19,5))
    plt.hist(yder2, bins=100)
    plt.show()

for i in range(1):
    plot_derive(xtrain[1+i])
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: Ru
ntimeWarning: divide by zero encountered in true_divide
//anaconda3/lib/python3.7/site-packages/IPython/core/pylabtools.py:1
28: UserWarning: Creating legend with loc="best" can be slow with la
rge amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



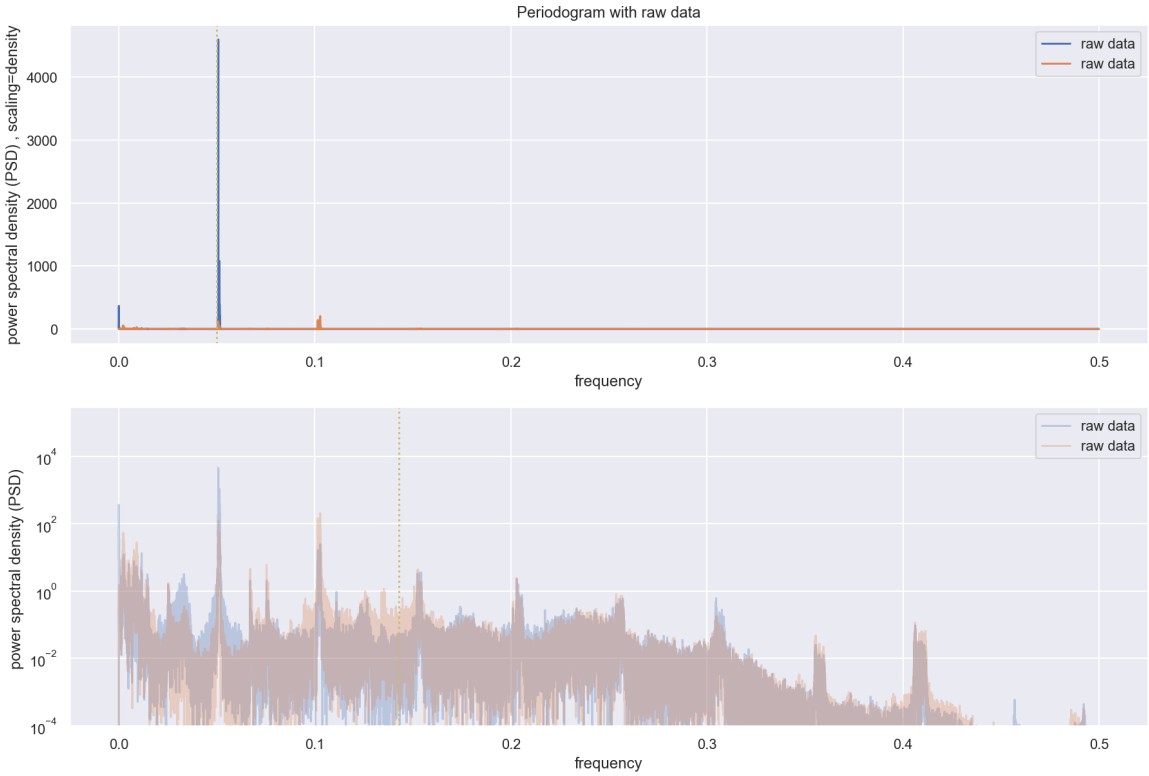
# Frequency Domaine

Frequency domain refers to the analysis of mathematical functions or signals with respect to frequency, rather than time. Put simply, a time-domain graph shows how a signal changes over time, whereas a frequency-domain graph shows how much of the signal lies within each given frequency band over a range of frequencies.

## Perodogram

In [222]:

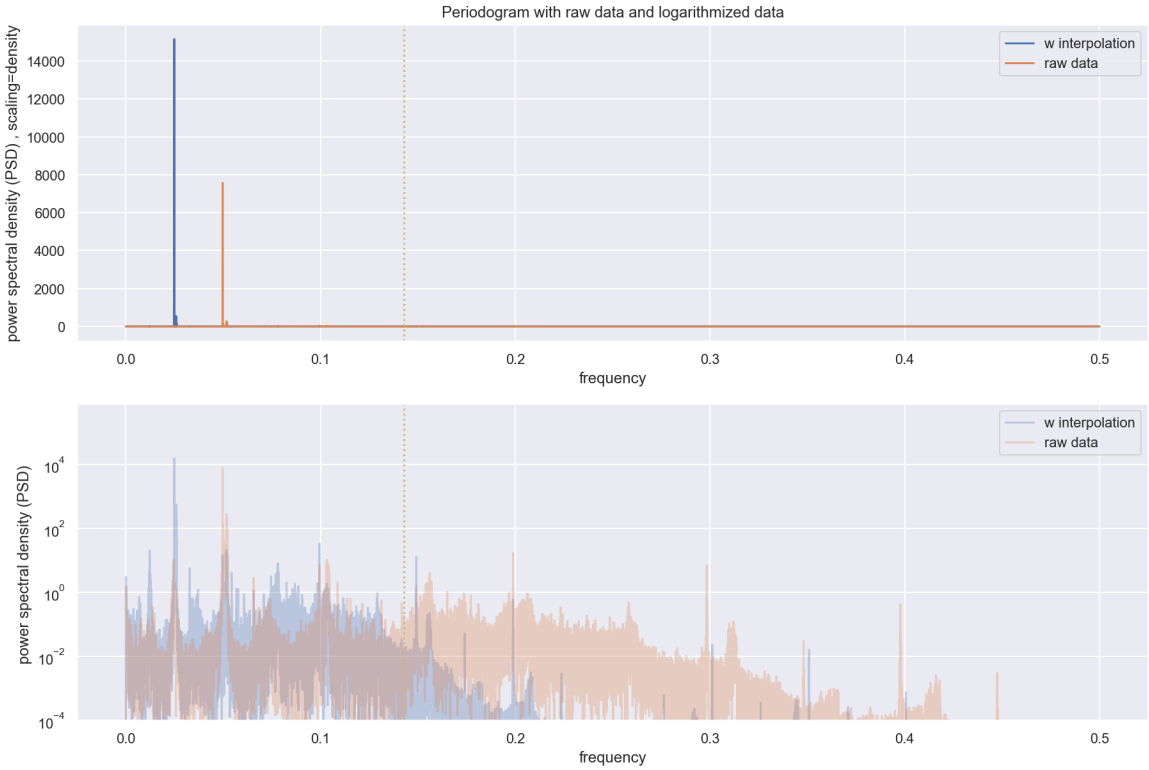
```
# plot Pedrodogram
i = 20
k = 2
ynew = []
start = 0
end = 61440
sns.set()
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)

for n in range(k):
    # f : ndarray : Array of sample frequencies.
    # Pxx : ndarray :Power spectral density or power spectrum of x.
    f, Pxx_den = signal.periodogram(xtrain[i+n, start:end], scaling='density')
    plt.plot(f, Pxx_den, label='raw data')
plt.xlabel('frequency')
plt.axvline(x=0.05, c='y', linestyle=':')
plt.ylabel('power spectral density (PSD) , scaling=density')
plt.title('Periodogram with raw data')
plt.legend()

plt.subplot(2, 1, 2)
for n in range(k):
    f, Pxx_den = signal.periodogram(xtrain[i+n, start:end], scaling='density')
    # This is just a thin wrapper around plot which additionally changes the y-a
xis to log scaling.
    plt.semilogy(f, Pxx_den, alpha=0.3, label='raw data')
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.semilogy.html
plt.ylim(10e-5,)
plt.axvline(x=1/7, c='y', linestyle=':')
plt.xlabel('frequency')
plt.ylabel('power spectral density (PSD) ')
plt.legend()
plt.show()
```


Periodogram with raw data
power spectral density (PSD) , scaling=density
raw data
raw data
4000
3000
2000
1000
0
0.0
0.1
0.2
0.3
0.4
0.5
frequency


power spectral density (PSD)
raw data
raw data
$10^{4}$
$10^{2}$
$10^{0}$
$10^{-2}$
$10^{-4}$
0.0
0.1
0.2
0.3
0.4
0.5
frequency

In [214]:

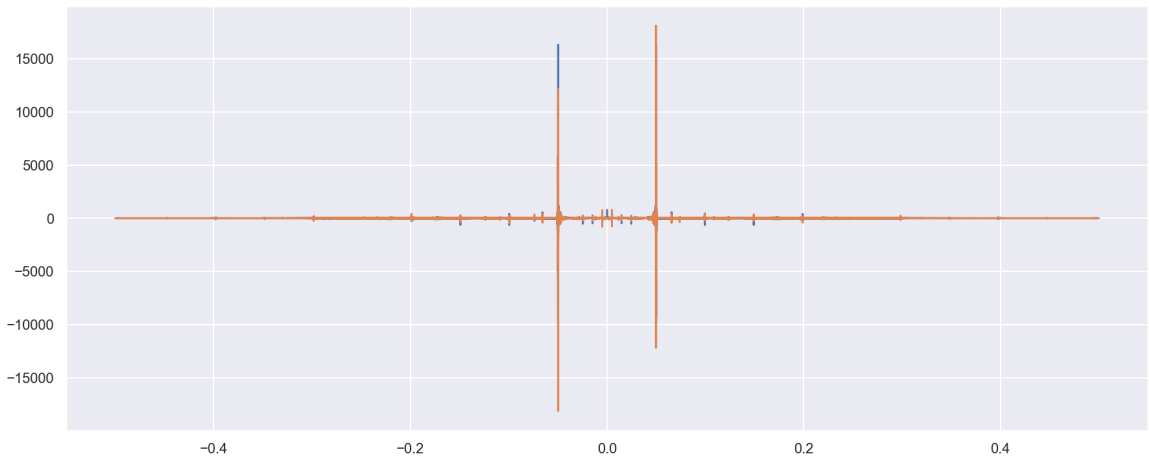
```
# plot Pedrodogram for interpolated data
i = 10
k = 1
ynew = []
start=0
end=61440

for n in range(k):
    # spline interpolation
    tck = interpolate.splrep(range(61440), xtrain[i+n, ], s=0)
    xnew = np.arange(start, end, 1/2)
    y = interpolate.splev(xnew, tck, der=0)
    ynew.append(y)

sns.set()
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)

for n in range(k):
    # f : ndarray : Array of sample frequencies.
    # Pxx : ndarray :Power spectral density or power spectrum of x.
    f, Pxx_den = signal.periodogram(ynew[n], scaling='density')
    plt.plot(f, Pxx_den,label='w interpolation')
    f, Pxx_den = signal.periodogram(xtrain[i+n, start:end], scaling='density')
    plt.plot(f, Pxx_den,label='raw data')

plt.xlabel('frequency')
plt.axvline(x=1/7, c='y', linestyle=':')

plt.ylabel('power spectral density (PSD) , scaling=density')
plt.title('Periodogram with raw data ')
plt.legend()

plt.subplot(2, 1, 2)

for n in range(k):
    f, Pxx_den = signal.periodogram(ynew[n], scaling='density')
    # This is just a thin wrapper around plot which additionally changes the y-a
xis to log scaling.
    plt.semilogy(f, Pxx_den, alpha=0.3,label='w interpolation')

    f, Pxx_den = signal.periodogram(xtrain[i+n, start:end], scaling='density')
    # This is just a thin wrapper around plot which additionally changes the y-a
xis to log scaling.
    plt.semilogy(f, Pxx_den, alpha=0.3,label='raw data')
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.semilogy.html
plt.ylim(10e-5,)
plt.axvline(x=1/7, c='y', linestyle=':')
plt.xlabel('frequency')
plt.ylabel('power spectral density (PSD) ')
plt.legend()
plt.show()
```




# FFT Fast Fourier Transformation

In [68]:

```
i=2
start=0
end=61440
length=end-start
sp = np.fft.fft(xtrain[i,start:end])
freq = np.fft.fftfreq(length)
sns.set()
plt.figure(figsize=(15, 6))
plt.plot(freq, sp.real, freq, sp.imag)
plt.show()
```



The DTFT, X(e jΩ), is periodic. One period extends from f = 0 to fs, where fs is the sampling frequency.

The FFT contains information between 0 and fs, however, we know that the sampling frequency 5 must be at least twice the highest frequency component. Therefore, the signal's spectrum should be entirly below fs 2 , the Nyquist frequency.

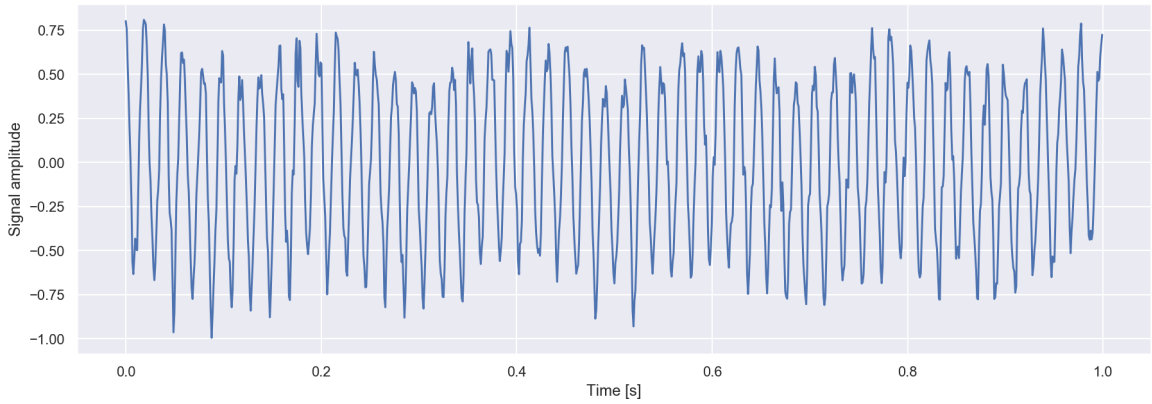
In [554]:

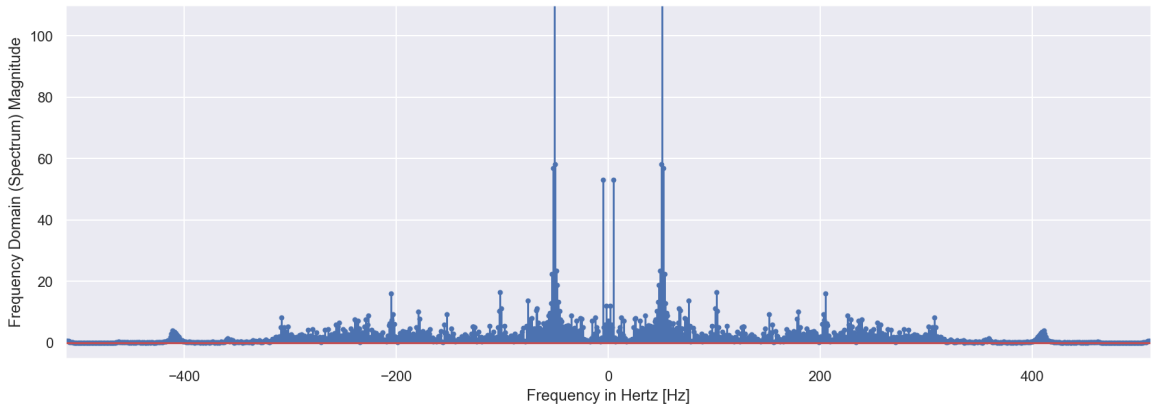
```
f = 1  # Frequency, in cycles per second, or Hertz
f_s = 1024  # Sampling rate, or number of measurements per second
i=5
start=0
end=1024
length=end-start

#t = np.linspace(0, 60, 60*f_s, endpoint=False)
t = np.linspace(0, 1, 1*f_s, endpoint=False)
x = xtrain[i,start:end]

fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(t, x)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Signal amplitude')

X = fftpack.fft(x)
freqs = fftpack.fftfreq(len(x)) * f_s

fig, ax = plt.subplots(figsize=(15, 5))

ax.stem(freqs, np.abs(X),markerfmt='C0.',basefmt='C3-')
ax.set_xlabel('Frequency in Hertz [Hz]')
ax.set_ylabel('Frequency Domain (Spectrum) Magnitude')
ax.set_xlim(-f_s / 2, f_s / 2)
ax.set_ylim(-5, 110)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: Us
erWarning: In Matplotlib 3.3 individual lines on a stem plot will be
added as a LineCollection instead of individual lines. This signific
antly improves the performance of a stem plot. To remove this warnin
g and switch to the new behaviour, set the "use_line_collection" key
word argument to True.
```

Out[554]:

```
(-5, 110)
```



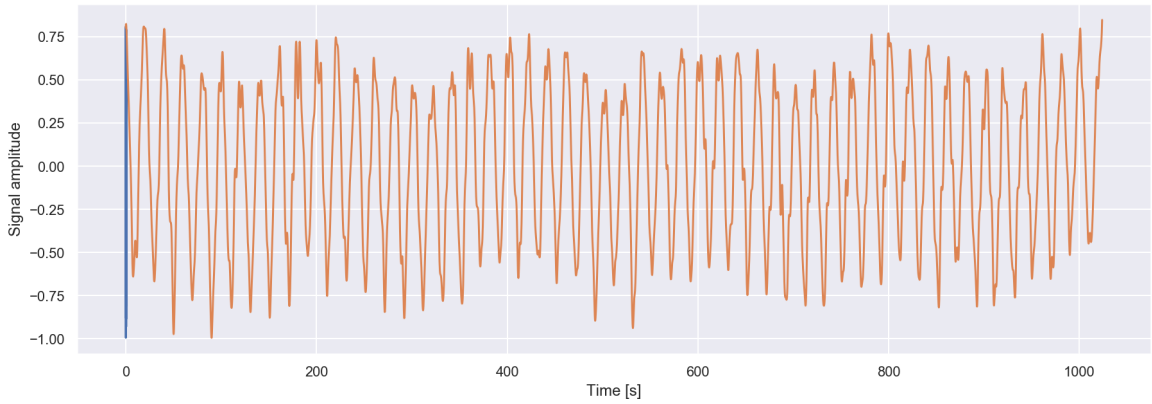
In [670]:

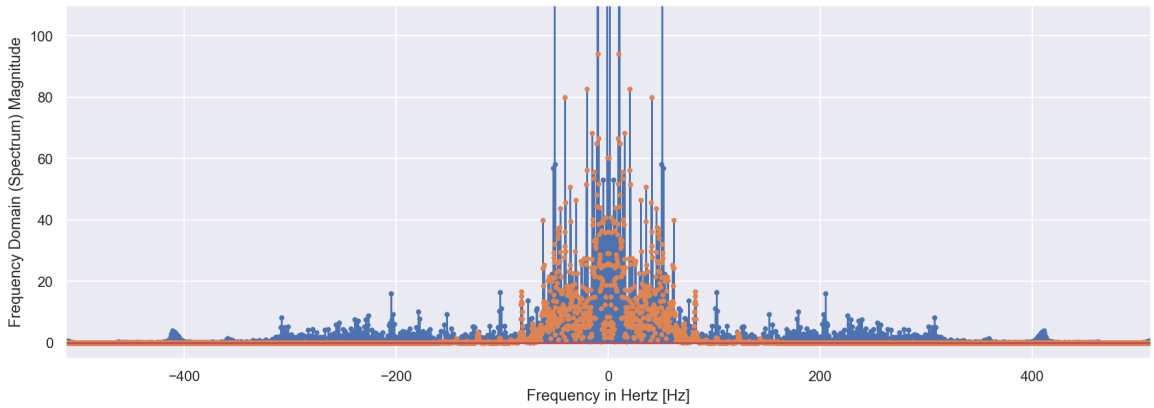
```
f = 1  # Frequency, in cycles per second, or Hertz
f_s = 1024  # Sampling rate, or number of measurements per second

i = 5
start = 0
end = 1024
length = end-start

tck = interpolate.splrep(range(length), xtrain[i+n, start:end], s=0)
xnew = np.arange(start, end, 1/5)
ynew = interpolate.splev(xnew, tck, der=0)

t = np.linspace(0, 1, 1*f_s, endpoint=False)

fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(t, xtrain[i+n, start:end])
ax.plot(xnew, ynew)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Signal amplitude')

x = xtrain[i+n, start:end]
X = fftpack.fft(x)
freqs = fftpack.fftfreq(len(x)) * f_s

fig, ax = plt.subplots(figsize=(15, 5))

ax.stem(freqs, np.abs(X), markerfmt='C0.', basefmt='C3-')

x2 = ynew
X = fftpack.fft(x2)
freqs = fftpack.fftfreq(len(x2)) * f_s
ax.stem(freqs, np.abs(X), markerfmt='C1.', basefmt='C3-')
ax.set_xlabel('Frequency in Hertz [Hz]')
ax.set_ylabel('Frequency Domain (Spectrum) Magnitude')
ax.set_xlim(-f_s / 2, f_s / 2)
ax.set_ylim(-5, 110)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:31: Us
erWarning: In Matplotlib 3.3 individual lines on a stem plot will be
added as a LineCollection instead of individual lines. This signific
antly improves the performance of a stem plot. To remove this warnin
g and switch to the new behaviour, set the "use_line_collection" key
word argument to True.
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:36: Us
erWarning: In Matplotlib 3.3 individual lines on a stem plot will be
added as a LineCollection instead of individual lines. This signific
antly improves the performance of a stem plot. To remove this warnin
g and switch to the new behaviour, set the "use_line_collection" key
word argument to True.
```

Out[670]:

```
(-5, 110)
```




# STFT Short-term Fourier Transformation

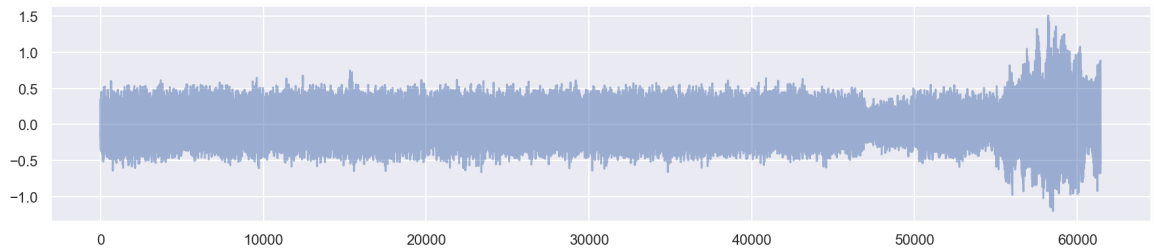
Short time Fourier transform (STFT) is one of the methods of linear time-frequency analysis that can provide localized spectrum in time domain by applying Fourier transform in a localized time window.

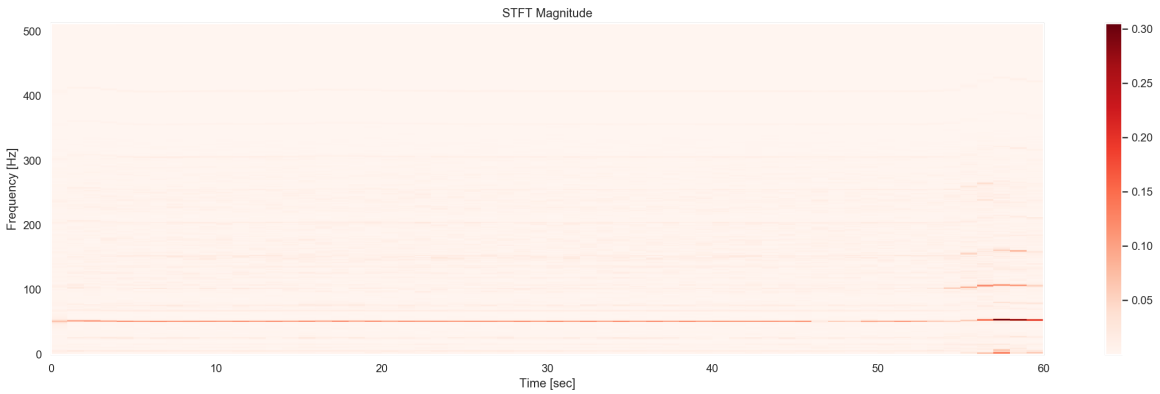
In [ ]:

```
scipy.signal.stft(x, fs=1.0, window='hann', nperseg=256, noverlap=None, nfft=Non
e,
                  detrend=False, return_onesided=True, boundary='zeros', padded=
True, axis=-1)[source]
```

In [134]:

```
# Sampling frequency of the x time series
fs = 1024
i = 6
plt.figure(figsize=(15, 3))
plt.plot(xtrain[i], alpha=0.5)

def cal_STFT(x):
    """f  Array of sample frequencies.
    t  Array of segment times.
    Zxx - STFT of x. By default, the last axis of Zxx corresponds to the segment times."""
    f, t, Zxx = signal.stft(x, fs, nperseg=1024, noverlap=0)  # 100 // 20)
    return f,Zxx

def plot_STFT(x):
    plt.figure(figsize=(22, 6))
    """f  Array of sample frequencies.
    t  Array of segment times.
    Zxx - STFT of x. By default, the last axis of Zxx corresponds to the segment times."""
    f, t, Zxx = signal.stft(x, fs, nperseg=1024, noverlap=0)  # 100 // 20)
    plt.pcolormesh(t, f, np.abs(Zxx), cmap='Reds')  # , vmin=0)
    plt.title('STFT Magnitude')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.colorbar()
    return

plot_STFT(xtrain[i])
#spectrum, freqs = plot_spectrum(xtrain[i])
```



In [135]:

```
# the sum of STFT of all train data
plt.figure(figsize=(22, 8))
ZxxTotal=np.zeros(Zxx.shape)
for i in range(len(xtrain)):
    f, t, Zxx = signal.stft(xtrain[i], fs, nperseg=1024, noverlap=0)  # 100 // 2
0)
    ZxxTotal=ZxxTotal+np.abs(Zxx)
plt.pcolormesh(t, f, np.abs(ZxxTotal), cmap='Reds')  # , vmin=0)
plt.title('the sum of STFT Magnitude of all train data')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()
plt.show()
```


I observe the great difference of global patern of train and test dataset, in the test dataset, a lot of samples are concentrated on 0Hz frenquency. While in the **train dataset we can see a very strong density around 50Hz and 100Hz, and their multiples. This corresponde an hypothese of self-exciting ocillation anormaly**.

In [138]:

```
# the sum of STFT of all test data
plt.figure(figsize=(22, 8))
ZxxTotal=np.zeros(Zxx.shape)
for i in range(len(xtest)):
    f, t, Zxx = signal.stft(xtest[i], fs, nperseg=1024, noverlap=0)  # 100 // 2
0)
    ZxxTotal=ZxxTotal+np.abs(Zxx)
plt.pcolormesh(t, f, np.abs(ZxxTotal), cmap='Reds')  # , vmin=0)
plt.title('the sum of STFT Magnitude of all test data')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()
plt.show()
```

In [1027]:

```
pd.DataFrame(np.abs(Zxx)).T.describe()
```

Out[1027]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **count** | 616.000000 | 616.000000 | 616.000000 | 616.000000 | 616.000000 | 616.000000 | 616.000000 |
| **mean** | 0.032502 | 0.017276 | 0.021068 | 0.020759 | 0.229899 | 0.450074 | 0.221167 |
| **std** | 0.018123 | 0.010109 | 0.011643 | 0.012321 | 0.040605 | 0.060651 | 0.021299 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.026377 | 0.012484 | 0.012453 | 0.011646 | 0.232824 | 0.455731 | 0.217639 |
| **50%** | 0.030782 | 0.016234 | 0.019834 | 0.019252 | 0.240609 | 0.464186 | 0.224038 |
| **75%** | 0.036612 | 0.020613 | 0.028318 | 0.027131 | 0.247539 | 0.471800 | 0.229840 |
| **max** | 0.240832 | 0.123651 | 0.067580 | 0.094502 | 0.279502 | 0.529780 | 0.265941 |

8 rows × 51 columns

# Time–frequency analysis

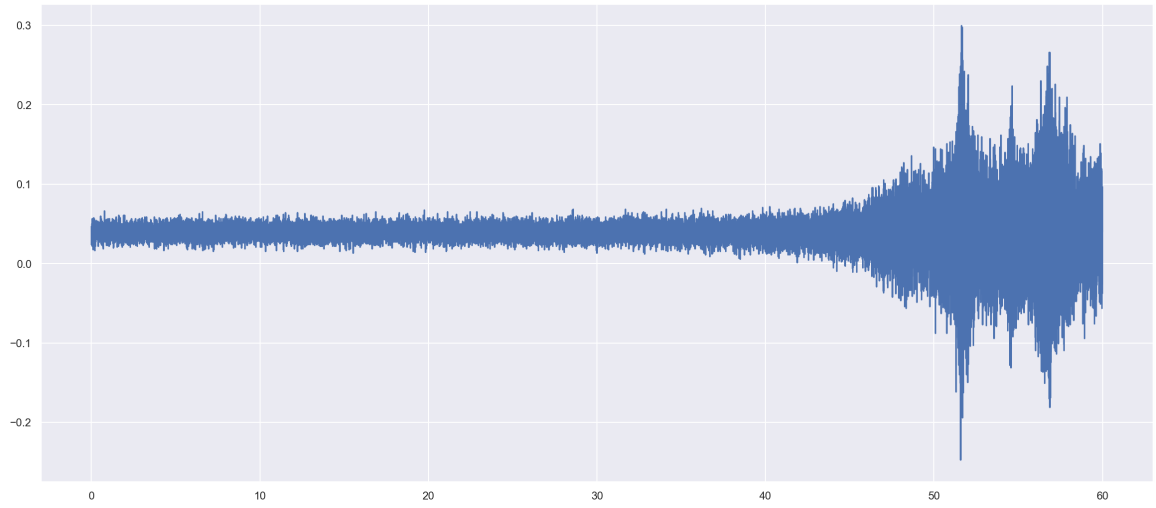
Time–frequency analysis comprises those techniques that study a signal **in both the time and frequency domains simultaneously**, using various **time–frequency representations**. Rather than viewing a 1-dimensional signal (a function, real or complex-valued, whose domain is the real line) and some transform (another function whose domain is the real line, obtained from the original via some transform), time–frequency analysis studies a **two-dimensional signal – a function whose domain is the two-dimensional real plane**, obtained from the signal via a time–frequency transform.

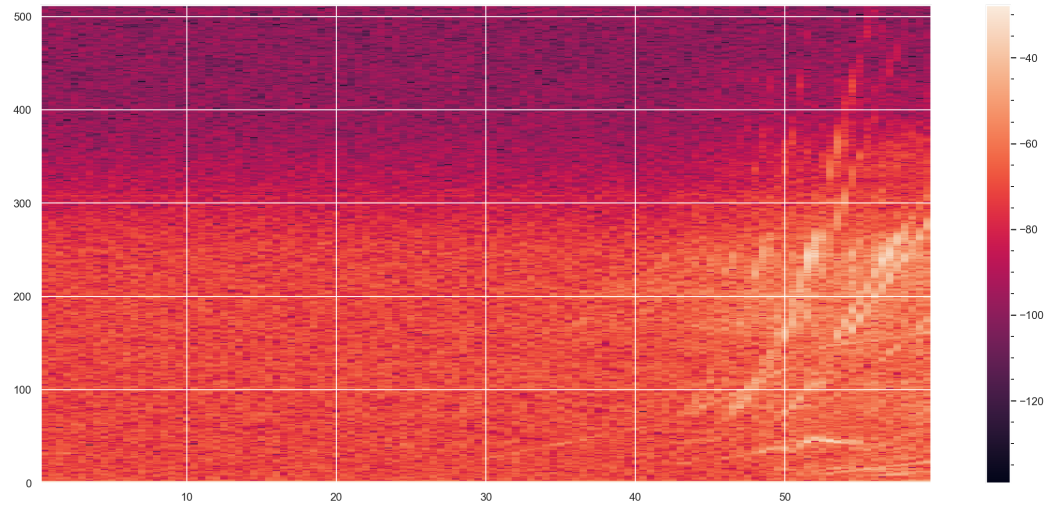
- Why I will focus on Time–frequency analysis?

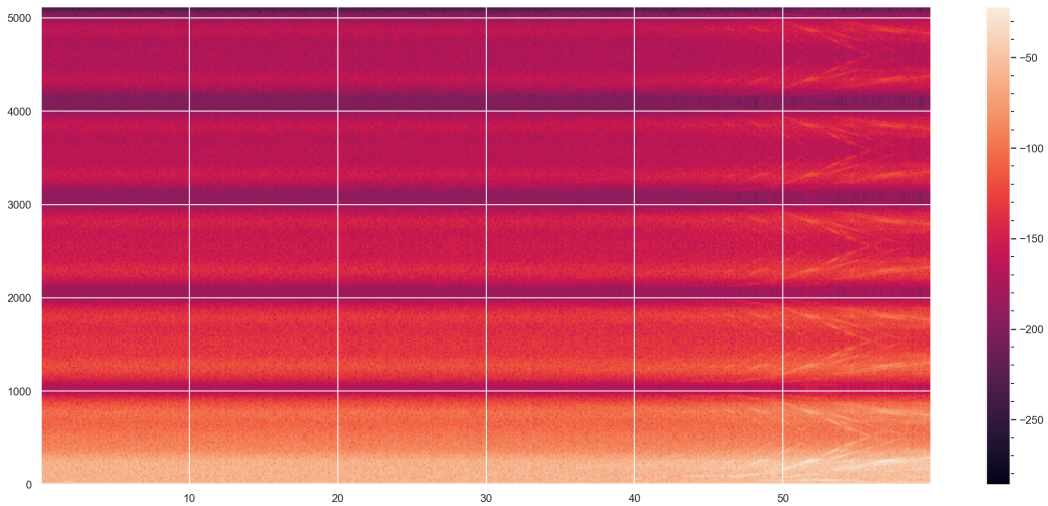
  Because signal frequency characteristics are varying with time. We can see clearly on the graphes below.

In [81]:

```
i = 6
for i in range(1):
    dt = 1/1024
    t = np.arange(0.0, 60.0, dt)
    s1 = xtrain[i, :]

    tck = interpolate.splrep(range(len(xtrain[i, :])), xtrain[i, :], s=0)
    xnew = np.arange(0, 61440, 1/10)
    s2 = interpolate.splev(xnew, tck, der=0)

    # the raw data
    x1 = s1   # the signal
    NFFT = 1024  # the length of the windowing segments
    Fs1 = int(1.0 / dt)  # the sampling frequency

    # the interpolated data
    x2 = s2   # the signal
    NFFT = 1024  # the length of the windowing segments
    Fs2 = int(1.0 * 10 / dt)  # the sampling frequency

    fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, figsize=(20, 30))
    ax1.plot(t, x1)
    #mode : {'default', 'psd', 'magnitude', 'angle', 'phase'}
    spectrum1, freqs1, bins1, im1 = ax2.specgram(
        x1, NFFT=NFFT, Fs=Fs1, noverlap=512, mode='magnitude')
    cbar = fig.colorbar(im1, ax=ax2)
    cbar.minorticks_on()

    spectrum2, freqs2, bins2, im2 = ax3.specgram(
        x2, NFFT=NFFT, Fs=Fs2, noverlap=128, mode='magnitude')
    cbar2 = fig.colorbar(im2, ax=ax3)
    cbar2.minorticks_on()
    # The `specgram` method returns 4 objects. They are:
    # - Pxx: the periodogram
    # - freqs: the frequency vector
    # - bins: the centers of the time bins
    # - im: the matplotlib.image.AxesImage instance representing the data in the
plot
    plt.show()
```


0.3
0.2
0.1
0.0
−0.1
−0.2
0
10
20
30
40
50
60


500
400
300
200
100
0
10
20
30
40
50
−40
−60
−80
−100
−120

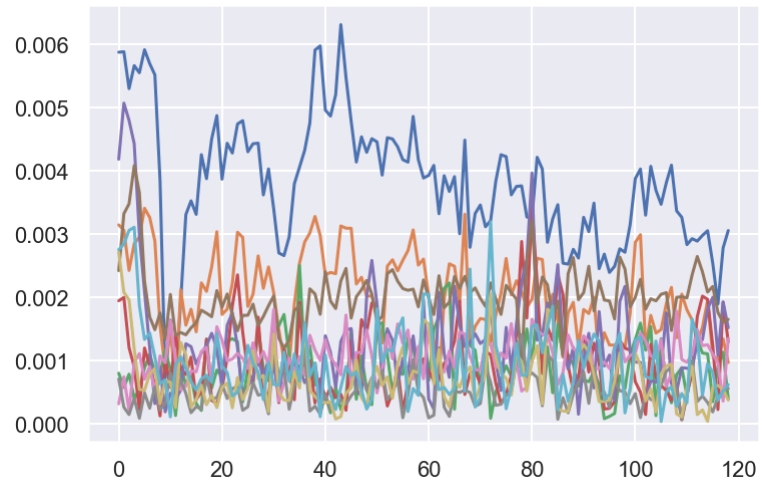

5000
4000
3000
2000
1000
0
10
20
30
40
50
−50
−100
−150
−200
−250

In [652]:

```
for i in range(10):
    plt.plot(spectrum1[i])
spectrum1.shape
```

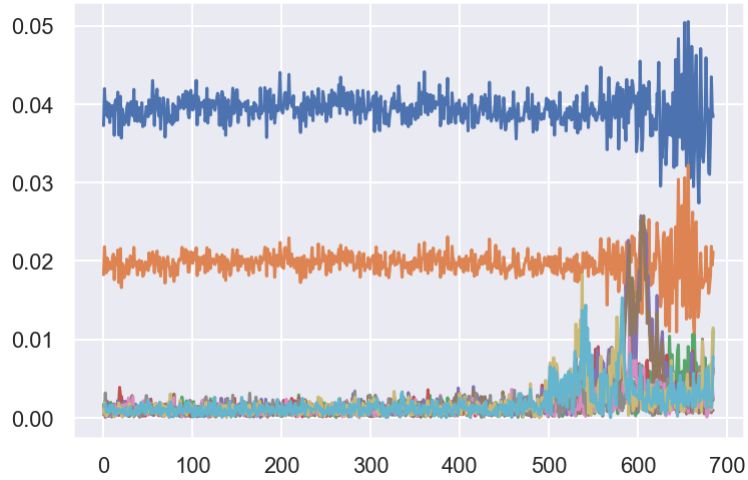
Out[652]:

```
(513, 119)
```



In [140]:

```
spectrum2[1]
for i in range(10):
    plt.plot(spectrum2[i])
```




# Frequency Domaine approach with STFT Matrix based

## 2 D and 3 D STFT Matrix

**Short time Fourier transform (STFT)** is one of the methods of linear time-frequency analysis that can provide localized spectrum in time domain by applying Fourier transform in a localized time window. A sigle STFT is a 2D time-frequency matrix, here we have a 3rd dimention which is samples.

- Why I use STFT instead of FFT?

**Because STFT contains the frequency evolution in time**, FFT only reflect the frequency on the whole time sequence.

Here we can consider STFT as a multivariate time series.

I use weighted average STFT frenquce sequence to transforme the problem into univariate time series using 2D features (time, average freqs) for each data.

The **3 Dimensions STFT Matrx has 3 dimension of (time, frequence , power)**. I treat this multivariate time series as image, using **2D convolutional VAE**.

In [207]:

```
# a STFT spectrum example
plt.figure(figsize=(22, 8))
i=211
f, t, Zxx = signal.stft(xtrain[i], fs, nperseg=1024, noverlap=0)
plt.pcolormesh(t, f, np.abs(np.abs(Zxx)), cmap='Reds')
plt.title('the sum of STFT Magnitude of all train data')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()
plt.show()
```


In [82]:

```
# STFT matrix for neuro networks
def STFT_matrix_3D_array(data, n_samples):
    f, Zxx = cal_STFT(data[0])
    xtrain_STFT_3D = np.zeros(Zxx.shape)
    for i in range(n_samples):
        if i == 0:
            xtrain_STFT_3D = np.abs(Zxx)
        else:
            f, Zxx = cal_STFT(data[i])
            xtrain_STFT_3D = np.dstack([xtrain_STFT_3D, np.abs(Zxx)])
    xtrain_STFT_3D = np.moveaxis(xtrain_STFT_3D, -1, 0)
    return xtrain_STFT_3D


def STFT_matrix_3D_array_1sample(data):
    f, Zxx = cal_STFT(data)
    xtrain_STFT_3D = np.abs(Zxx)
    xtrain_STFT_3D = np.expand_dims(xtrain_STFT_3D, 0)
    return xtrain_STFT_3D


result = STFT_matrix_3D_array_1sample(xtest[2])
result.shape
```

Out[82]:

```
(1, 513, 61)
```

In [87]:

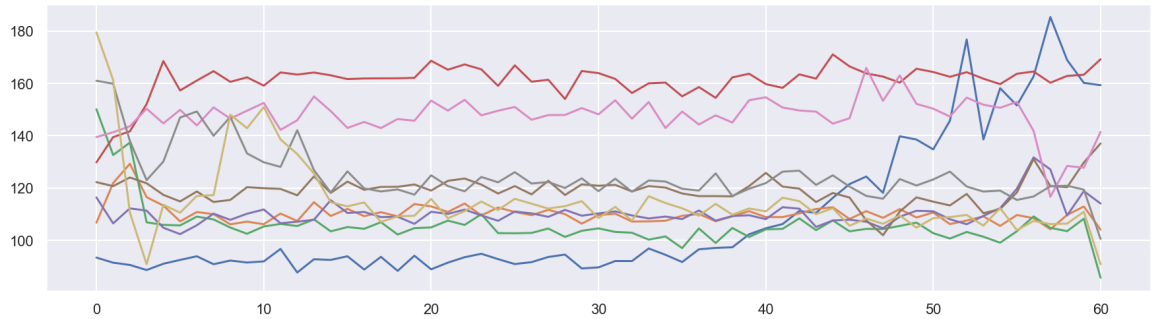
```
# STFT matrix as list
def STFT_matrix_3D_list(data, n_samples):
    xtrain_STFT_3D = []
    for i in range(n_samples):
        f, Zxx = cal_STFT(xtrain[i])
        xtrain_STFT_3D.append(np.abs(Zxx))
    return xtrain_STFT_3D
```

In [91]:

```
# PCA transform only on 2 dimension , can use average frequence sequence
#average frequence
#plt.plot((f@np.abs(Zxx))/np.sum(np.abs(Zxx),axis=0))
xtrain_STFT_3D=STFT_matrix_3D_array(xtrain,200)
f, Zxx = cal_STFT(xtrain[1])
xtrain_STFT_2D_mean=np.zeros
for i,Zxx in enumerate (xtrain_STFT_3D):
    if i==0:
        xtrain_STFT_2D_mean=(f@Zxx)/np.sum(Zxx,axis=0)
    else:
        xtrain_STFT_2D_mean=np.vstack((xtrain_STFT_2D_mean,(f@Zxx)/np.sum(Zxx,ax
is=0)))
```

In [92]:

```
plt.figure(figsize=(15,4))
for i in range(9):
    plt.plot(xtrain_STFT_2D_mean[i])
```

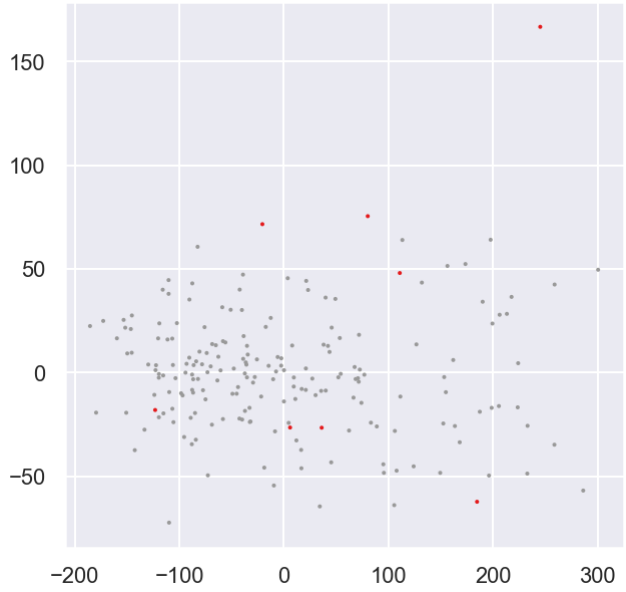


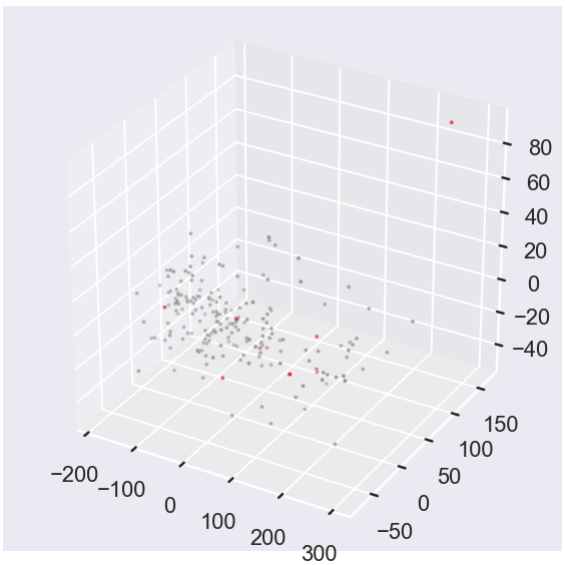

In [98]:

```
def viz_PCA(x, labels):
    pca = PCA(n_components=2).fit(x)
    datapoint = pca.transform(x)
    plt.figure(figsize=(5, 5))
    plt.scatter(datapoint[:, 0], datapoint[:, 1], c=labels, cmap='Set1', s=1)
    plt.show()

def viz_PCA3D(x, labels):
    pca3D = PCA(n_components=3).fit(x)
    datapoint3D = pca3D.transform(x)
    fig = plt.figure(figsize=(5, 5))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(datapoint3D[:, 0], datapoint3D[:, 1],
               datapoint3D[:, 2], c=labels, cmap='Set1', s=1)
    plt.show()
```

In [100]:

```
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=False)
labels = lof1.fit_predict(xtrain_STFT_2D_mean)

#labels=np.where( sscore<np.percentile(sscore,10),1 , 0)
viz_PCA(xtrain_STFT_2D_mean, labels)
viz_PCA3D(xtrain_STFT_2D_mean, labels)
```

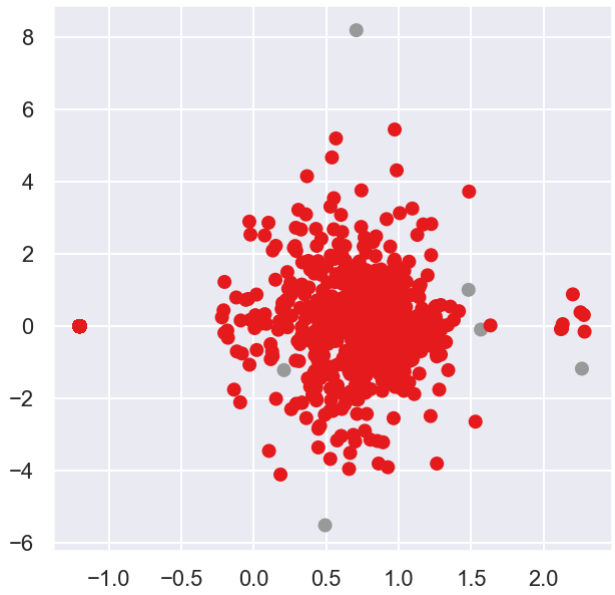


In [1359]:

```
pca2 = PCA(n_components=2, whiten=True)
datapoint = pca2.fit_transform(xtrain_STFT_2D_mean)

lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(datapoint)
sscore = -lof1.score_samples(datapoint)
label = np.where(sscore > np.percentile(sscore, 99), 1, 0)

print('percent of the variance ' + str(sum(pca2.explained_variance_ratio_)))
plt.figure(figsize=(5, 5))
plt.scatter(datapoint[:, 0], datapoint[:, 1], c=label, cmap='Set1')
plt.show()
```

```
percent of the variance 0.9959792068249499
```




```
5
```

# STFT with Univariate Time Series treatment

## LOF - on STFT 61-dimensional space

LOF only take 2 dimension data, I use average frequence for each second to **reduce the dimenstion form 1677*51*361 to 1677*61**.

- Features: STFT (average frequence for T )
- Features dimenstion: 61
- Data: train data
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :0.598095653034**

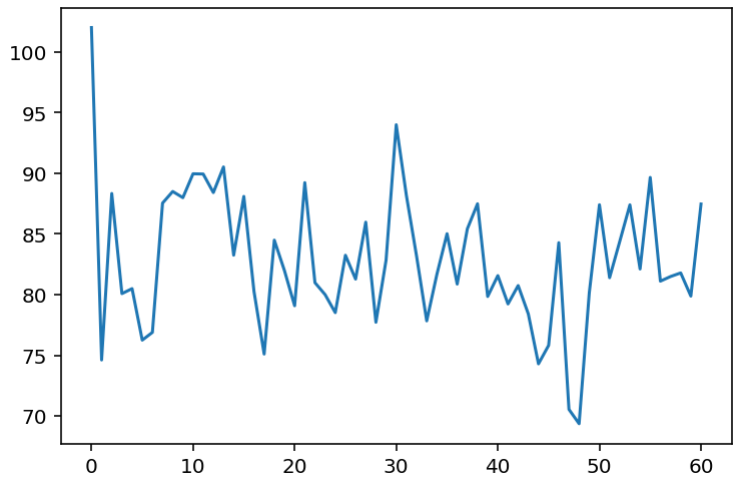
In [96]:

```
plt.plot(xtest_STFT_2D_mean[1])
```

```
1.772134178279798
10892.022004092021
```

Out[96]:

```
[<matplotlib.lines.Line2D at 0x1b0a41320>]
```




In [14]:

```
# STFT matrix
xtrain_STFT = []
for i in range(len(xtrain)):
    f, Zxx = cal_STFT(xtrain[i])
    xtrain_STFT.append(np.abs(Zxx))

xtest_STFT = []
for i in range(len(xtest)):
    f, Zxx = cal_STFT(xtest[i])
    xtest_STFT.append(np.abs(Zxx))
print(len(xtrain_STFT),xtrain_STFT[1].shape, len(xtest_STFT),xtest_STFT[1].shape
)
```

```
1677 (513, 61) 2511 (513, 61)
```

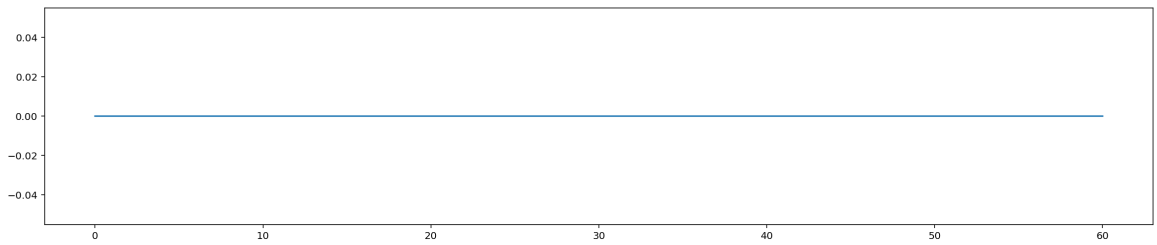
In [71]:

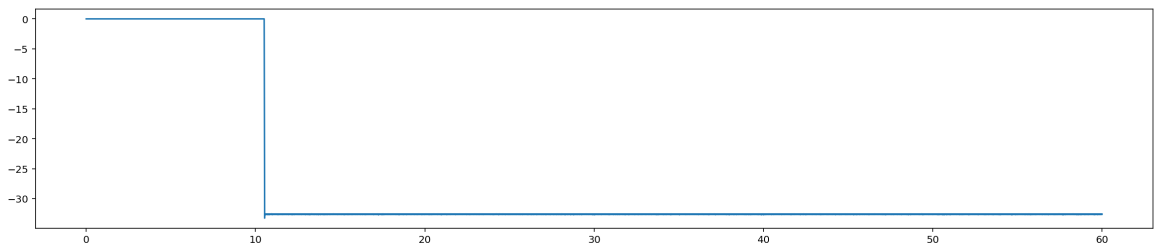
```
# LOF only take 2 dimension data,
# we can use average frequence for each second to reduce the dimenstion form 167
7*513*61 to 1677*61
# plt.plot((f@np.abs(Zxx))/np.sum(np.abs(Zxx),axis=0))
xtrain_STFT_2D_mean = np.zeros
for i, Zxx in enumerate(xtrain_STFT):
    if i == 0:
        xtrain_STFT_2D_mean = f@Zxx/np.sum(Zxx, axis=0)
    else:
        xtrain_STFT_2D_mean = np.vstack(
            (xtrain_STFT_2D_mean, f@Zxx/np.sum(Zxx, axis=0)))


xtest_STFT_2D_mean = np.zeros
for i, Zxx in enumerate(xtest_STFT):
    if i == 0:
        xtest_STFT_2D_mean = f@Zxx/np.sum(Zxx, axis=0)
    else:
        xtest_STFT_2D_mean = np.vstack(
            (xtest_STFT_2D_mean, f@Zxx/np.sum(Zxx, axis=0)))
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:17: Ru
ntimeWarning: invalid value encountered in true_divide
```

In [100]:

```
xtest_STFT_2D_mean[2433]
plot_wave(xtest[2019])
plot_wave(xtest[2433])
```




In [93]:

```
# for data 2019 and 2433, we get Nan STFT value , we put into 0
# xtest_STFT_2D_mean=np.where(xtest_STFT_2D_mean=='nan',0,xtest_STFT_2D_mean)
xtest_STFT_2D_mean[2019] = 0
xtest_STFT_2D_mean[2433, :11] = 0
np.argwhere(np.isnan(xtest_STFT_2D_mean))
```

Out[93]:

```
array([], shape=(0, 2), dtype=int64)
```

In [ ]:

```
# Fit the low-dimensional method (61 dimentions)
lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(xtrain_STFT_2D_mean)

# Calculate anomaly score on the 61 dimentions test data
sscore = -lof1.score_samples(xtest_STFT_2D_mean)
print(np.argmax(sscore))
print(sscore[:100])
```

## LOF on a STFT PCA 10-dimensional space

- Features: STFT (average frequence for T )
- Features dimenstion: 10
- Data: train data
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :0.516262498919**

In [106]:

```
# PCA transform
pca1 = PCA(n_components = 10, whiten = True)
xtrain_fpca = pca1.fit_transform(xtrain_STFT_2D_mean)
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = Tru
e)
lof1.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca1.fit_transform(xtest_STFT_2D_mean)
sscore = -lof1.score_samples(xtest_fpca)
print(sscore[:100])
```

```
[3.01517002e+00 1.30388187e+00 1.44563409e+00 1.36828837e+00
 1.04447512e+00 1.13864849e+00 1.32161137e+00 1.48011556e+00
 1.24997893e+00 9.93220428e-01 1.46206566e+00 1.42423436e+00
 1.25280066e+00 1.08001269e+00 1.50302260e+00 1.12726388e+00
 1.14216855e+00 5.24836859e+03 1.53860835e+00 1.15970435e+00
 5.93068318e+02 1.22228785e+00 1.27887212e+00 1.51981131e+00
 1.20927726e+00 1.16550304e+00 1.18545922e+00 1.25453156e+00
 2.10031127e+03 1.65565833e+00 1.50664282e+00 1.20541997e+00
 2.00063726e+03 1.22879727e+00 1.46918214e+00 1.19674342e+00
 1.15323784e+00 1.60929937e+00 7.58104484e+02 2.00394428e+00
 1.22200512e+00 1.30557672e+00 4.73638749e+02 1.56901876e+00
 1.97156059e+00 1.28465525e+00 1.10269450e+00 1.16382347e+00
 1.57066663e+00 1.16182259e+00 1.24682574e+00 1.19902392e+00
 1.39318204e+00 1.55370802e+00 1.13448521e+00 1.13312108e+00
 1.12008381e+00 2.34207130e+00 2.31219825e+00 1.10113221e+00
 1.61151248e+00 1.39134691e+00 1.20778336e+00 1.05308193e+00
 1.46611631e+00 9.88476333e-01 2.54260923e+00 1.08118199e+00
 5.58391045e+03 1.22546445e+00 1.35931661e+00 1.11674125e+00
 1.45793981e+00 1.26118028e+03 1.20479820e+00 1.15941521e+03
 1.13719395e+00 1.20120601e+00 1.53656102e+00 1.26118505e+00
 1.35977025e+00 1.52267188e+00 1.67714548e+00 1.25525376e+00
 1.45548324e+00 1.40040541e+00 1.69057238e+00 1.67797370e+00
 1.33574912e+00 1.67423700e+00 1.10305197e+00 1.87865654e+00
 1.55493753e+00 1.14383238e+00 2.37920103e+00 1.03430418e+00
 1.26743166e+00 6.83410251e+03 1.17754789e+00 1.58537102e+00]
```

# 3D STFT Matrix and LSTM - Multivariate Time Series

In [102]:

```
def get_batch(source, i, seq_len, evaluation=False):
    # Deal with the possibility that there's not enough data left for a full seq
uence
    seq_len = min(seq_len, len(source) - 1 - i)
    # Take the input data seq
    data = source[:,i:i+seq_len]
    # target data is the next slice after input data
    target = source[:,i+1+seq_len]
    return data, target
```

In [192]:

```
"""for i in range(4):
    if i ==0:
        xtrain_STFT=np.abs(Zxx)
    else:
        Zxx=cal_STFT(xtrain[i])
        xtrain_STFT=np.dstack([xtrain_STFT,np.abs(Zxx)])
xtrain_STFT = np.moveaxis(xtrain_STFT, -1, 0)"""

seq_len=3

model=Sequential()
model.add(layers.LSTM(units=268,input_shape=(513,seq_len),return_sequences=False
))

model.add(layers.Dense(513))

model.compile(optimizer=RMSprop(lr=0.01),loss='mae')
#history=model.fit(data_batch,targets_batch,steps_per_epoch=5,          epochs=
2)
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 268)               291584
_________________________________________________________________
dense_13 (Dense)             (None, 513)               137997
=================================================================
Total params: 429,581
Trainable params: 429,581
Non-trainable params: 0
_________________________________________________________________
```

In [198]:

```
# k for each 60s data
for k in range(len(xtrain_STFT_3D[:])):

    # i:range(0,61,2)
    for i in (range(0, xtrain_STFT_3D[k].shape[-1], seq_len)):
        if i+seq_len+1 < xtrain_STFT_3D[k].shape[-1]:
            # get data:(513*seq_len)  targets:(513*1)
            data, targets = get_batch(xtrain_STFT_3D[k], i, seq_len)

        if i == 0:
            data_seq = data
            targets_seq = targets
        else:
            if data.shape[1] != seq_len:
                pass
            else:
                # build a batch of data/targets for xtrain_STFT_3D[k]
                data_seq = np.dstack([data_seq, data])
                targets_seq = np.dstack([targets_seq, targets])

        # print(data_seq.shape)
    data_seq = np.moveaxis(data_seq, -1, 0)
    targets_seq = np.moveaxis(targets_seq, -1, 0)

    # Build k batchs of data/targets
    if k == 0:
        data_batch = data_seq
        targets_batch = targets_seq
    else:
        data_batch = np.dstack([data_batch, data_seq])
        targets_batch = np.dstack([targets_batch, targets_seq])

    #targets_batch= np.moveaxis(targets_batch, -1, 1)
    targets_seq_squeezed = np.squeeze(targets_seq)
    history = model.fit(data_seq, targets_seq_squeezed, steps_per_epoch=5, verbo
se=0,
                        epochs=2)
```

In [1192]:

```
pred_STFT=model.predict(self, x, batch_size=32, verbose=0)
```

Out[1192]:

4

In [1542]:

```
# k for each 60s data
for k in range(20):

    # i:range(0,61,2)
    for i in (range(0, xtrain_STFT_3D[k].shape[-1])):
        if i+seq_len+1<xtrain_STFT_3D[k].shape[-1]:
            #get data:(513*seq_len)  targets:(513*1)
            data, targets = get_batch(xtrain_STFT_3D[k], i, seq_len)

        if i==0:
            data_seq=data
            targets_seq=targets
            pred_seq=pred
        else:
            if data.shape[1]!=seq_len:
                pass
            else:
                #build a batch of data/targets for xtrain_STFT_3D[k]
                data_seq=np.dstack([data_seq,data])
                targets_seq=np.dstack([targets_seq,targets])
                pred_seq=np.dstack([pred_seq,pred])

        #print(data_seq.shape)
    data_seq= np.moveaxis(data_seq, -1, 0)
    targets_seq= np.moveaxis(targets_seq, -1, 0)


    pred_seq=model.predict(data_seq, batch_size=32, verbose=1)

    #Build k batchs of data/targets
    if k==0:
        data_batch=data_seq
        targets_batch=targets_seq
        pred_batch=pred_seq
        print(pred_seq.shape)
        print(pred_batch.shape)

    else:
        data_batch=np.dstack([data_batch,data_seq])
        targets_batch=np.dstack([targets_batch,targets_seq])
        pred_batch=np.dstack([pred_batch,pred_seq])
        print(pred_seq.shape)
        print(pred_batch.shape)

    #targets_batch= np.moveaxis(targets_batch, -1, 1)
    targets_seq_squeezed= np.squeeze(targets_seq)

pred_batch= np.moveaxis(pred_batch, -1, 0)
```

```
61/61 [==============================] - 1s 15ms/step
(61, 513)
(61, 513)
(61, 513)
(61, 513)
61/61 [==============================] - 1s 14ms/step
(61, 513)
(61, 513, 2)
(61, 513)
(61, 513, 2)
61/61 [==============================] - 1s 20ms/step
(61, 513)
(61, 513, 3)
(61, 513)
(61, 513, 3)
61/61 [==============================] - 1s 15ms/step
(61, 513)
(61, 513, 4)
(61, 513)
(61, 513, 4)
61/61 [==============================] - 1s 16ms/step
(61, 513)
(61, 513, 5)
(61, 513)
(61, 513, 5)
61/61 [==============================] - 1s 15ms/step
(61, 513)
(61, 513, 6)
(61, 513)
(61, 513, 6)
61/61 [==============================] - 1s 18ms/step
(61, 513)
(61, 513, 7)
(61, 513)
(61, 513, 7)
61/61 [==============================] - 1s 18ms/step
(61, 513)
(61, 513, 8)
(61, 513)
(61, 513, 8)
61/61 [==============================] - 1s 18ms/step
(61, 513)
(61, 513, 9)
(61, 513)
(61, 513, 9)
61/61 [==============================] - 1s 15ms/step
(61, 513)
(61, 513, 10)
(61, 513)
(61, 513, 10)
61/61 [==============================] - 1s 14ms/step
(61, 513)
(61, 513, 11)
(61, 513)
(61, 513, 11)
61/61 [==============================] - 1s 14ms/step
(61, 513)
(61, 513, 12)
(61, 513)
(61, 513, 12)
61/61 [==============================] - 1s 15ms/step
```

```
(61, 513)
(61, 513, 13)
(61, 513)
(61, 513, 13)
61/61 [==============================] - 1s 21ms/step
(61, 513)
(61, 513, 14)
(61, 513)
(61, 513, 14)
61/61 [==============================] - 1s 20ms/step
(61, 513)
(61, 513, 15)
(61, 513)
(61, 513, 15)
61/61 [==============================] - 1s 19ms/step
(61, 513)
(61, 513, 16)
(61, 513)
(61, 513, 16)
61/61 [==============================] - 1s 21ms/step
(61, 513)
(61, 513, 17)
(61, 513)
(61, 513, 17)
61/61 [==============================] - 1s 16ms/step
(61, 513)
(61, 513, 18)
(61, 513)
(61, 513, 18)
61/61 [==============================] - 1s 18ms/step
(61, 513)
(61, 513, 19)
(61, 513)
(61, 513, 19)
61/61 [==============================] - 1s 16ms/step
(61, 513)
(61, 513, 20)
(61, 513)
(61, 513, 20)
```

In [1544]:

```
pred_batch.shape
```

Out[1544]:

```
(20, 61, 513)
```

In [ ]:

```
for i in range(1):
    plt.figure(figsize=(22, 4))
    plt.pcolormesh(t, f, pred.T, cmap='Reds')  # , vmin=0)
    plt.title('STFT Magnitude')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.colorbar()
    plt.show()
```

# 3D STFT VAE conv2D

I treat the STFT graph as an image, so a 2D convolutional layers best suits here.

Variational Autoencoder (VAE) is an autoencoder whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to generate some new data. Generally, due to the regularisation effect of sampling, VAE surperformes the autoencoders.

## Train VAE on Frequency Domain

In [9]:

```
import tensorflow as tf
#train VAE on frequency domain
#get data of 4Dimensions (samples, 513,61, channel=1)
fs = 1024
def cal_STFT(x):
    """f  Array of sample frequencies.
    t  Array of segment times.
    Zxx - STFT of x. By default, the last axis of Zxx corresponds to the segment
times."""
    f, t, Zxx = signal.stft(x, fs, nperseg=1024, noverlap=512)
    return f,Zxx

def STFT_matrix_3D_array(data, n_samples):
    f, Zxx = cal_STFT(data[0])
    xtrain_STFT_3D = np.zeros(Zxx.shape)
    for i in range(n_samples):
        if i == 0:
            xtrain_STFT_3D = np.abs(Zxx)
        else:
            f, Zxx = cal_STFT(data[i])
            xtrain_STFT_3D = np.dstack([xtrain_STFT_3D, np.abs(Zxx)])
    xtrain_STFT_3D = np.moveaxis(xtrain_STFT_3D, -1, 0)
    return xtrain_STFT_3D

x_train=STFT_matrix_3D_array(xtrain,len(xtrain))
x_train=np.expand_dims(x_train, axis=3)
x_train=tf.convert_to_tensor(x_train)

x_test=STFT_matrix_3D_array(xtest,len(xtest))
x_test=np.expand_dims(x_test, axis=3)
x_test=tf.convert_to_tensor(x_test)
print(x_train.shape,x_test.shape)
```

```
(1677, 513, 121, 1) (2511, 513, 121, 1)
```

In [ ]:

```
tf.io.write_file(
    x_test_tensorSTFT, x_test, name=None
)
tf.io.write_file(
    x_train_tensorSTFT, x_train, name=None
)
```

In [201]:

```
import keras
from tensorflow.keras import layers
from keras import backend as K
from keras.models import Model
from keras.losses import mse, binary_crossentropy
class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

In [19]:

```
# Train VAE
input_shape = (513, 121, 1)
original_dim = (513, 121, 1)
intermediate_dim = 64
latent_dim = 2
batch_size = 128
image_size1 = 513
image_size2 = 61
filters = 12
kernel_size = (2, 2)

# build encoder model
inputs = tf.keras.Input(shape=input_shape, name='encoder_input')
x = inputs

for i in range(2):
    filters *= 2
    x = layers.Conv2D(filters=filters,
                      kernel_size=kernel_size,
                      activation='relu',
                      strides=1, padding='valid')(x)

# shape info needed to build decoder model
shape_before_flattening = K.int_shape(x)

# generate latent vector Q(z|X)
x = layers.Flatten()(x)
x = layers.Dense(intermediate_dim, activation='relu')(x)

# use reparameterization trick to push the sampling out as input
z_mean = layers.Dense(latent_dim, name='z_mean')(x)
z_log_var = layers.Dense(latent_dim, name='z_log_var')(x)

z = Sampling()((z_mean, z_log_var))
print(z)

# instantiate encoder model
encoder = tf.keras.Model(inputs=inputs, outputs=[
                         z_mean, z_log_var, z], name='encoder')

###############
# build decoder model
latent_inputs = tf.keras.Input(shape=(latent_dim,), name='z_sampling')
x = layers.Dense(shape_before_flattening[1] * shape_before_flattening[2]
                 * shape_before_flattening[3], activation='relu')(latent_inputs)

x = layers.Reshape(
    (shape_before_flattening[1], shape_before_flattening[2], shape_before_flatte
ning[3]))(x)

for i in range(2):
    x = layers.Conv2DTranspose(filters=filters,
                               kernel_size=kernel_size,
                               activation='relu',
                               strides=1, padding='valid')(x)
    filters //= 2

outputs = layers.Conv2DTranspose(filters=1,
                                 kernel_size=kernel_size,
```

```
                                activation='sigmoid',
                                padding='same',
                                name='decoder_output')(x)

# instantiate decoder model
decoder = tf.keras.Model(inputs=latent_inputs, outputs=outputs, name='decoder')
encoder.summary()
decoder.summary()

# instantiate VAE model
#outputs2 = decoder(encoder(inputs)[2])
outputs2 = decoder(z)
vae = tf.keras.Model(inputs=inputs, outputs=outputs2, name='vae')

# VAE loss = mse_loss or xent_loss + kl_loss
reconstruction_loss = mse(K.flatten(inputs), K.flatten(outputs2))
reconstruction_loss *= image_size1 * image_size2
kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')  # 'rmsprop')
# vae.summary()
```

```
Tensor("sampling_4/Identity:0", shape=(None, 2), dtype=float32)
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Con
nected to
==================================================================
==============================
encoder_input (InputLayer)   [(None, 513, 121, 1) 0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 512, 120, 24) 120         enc
oder_input[0][0]
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 511, 119, 48) 4656        con
v2d_8[0][0]
_________________________________________________________________
flatten_4 (Flatten)          (None, 2918832)      0           con
v2d_9[0][0]
_________________________________________________________________
dense_8 (Dense)              (None, 64)           186805312   fla
tten_4[0][0]
_________________________________________________________________
z_mean (Dense)               (None, 2)            130         den
se_8[0][0]
_________________________________________________________________
z_log_var (Dense)            (None, 2)            130         den
se_8[0][0]
_________________________________________________________________
sampling_4 (Sampling)        (None, 2)            0           z_m
ean[0][0]
                                                             z_l
og_var[0][0]
==================================================================
==============================
Total params: 186,810,348
Trainable params: 186,810,348
Non-trainable params: 0
_________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      [(None, 2)]               0
_________________________________________________________________
dense_9 (Dense)              (None, 2918832)           8756496
_________________________________________________________________
reshape_4 (Reshape)          (None, 511, 119, 48)      0
_________________________________________________________________
conv2d_transpose_8 (Conv2DTr (None, 512, 120, 48)      9264
_________________________________________________________________
conv2d_transpose_9 (Conv2DTr (None, 513, 121, 24)      4632
_________________________________________________________________
decoder_output (Conv2DTransp (None, 513, 121, 1)       97
```

```
=================================================================
Total params: 8,770,489
Trainable params: 8,770,489
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Output decoder missing from loss dictionary. We a
ssume this was done on purpose. The fit and evaluate APIs will not b
e expecting any data to be passed to decoder.
```

In [209]:

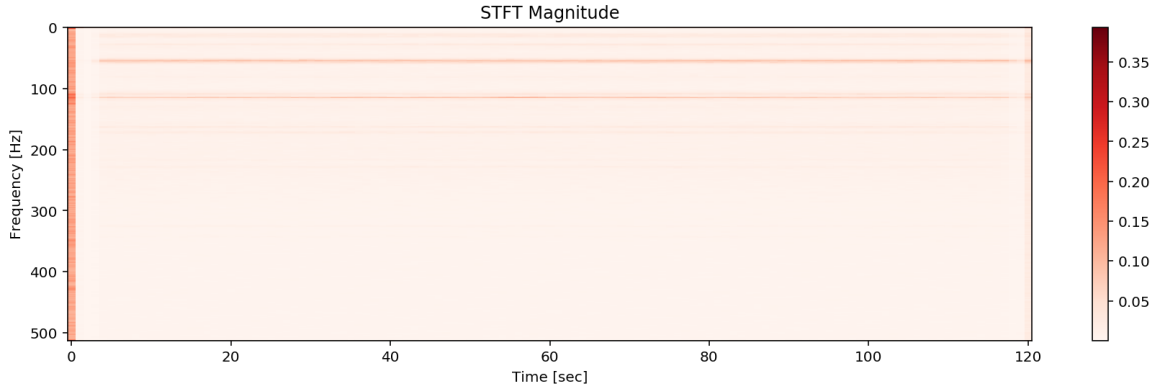
```
from keras.callbacks import ModelCheckpoint, TensorBoard
tensorboard = TensorBoard(log_dir='./logs',
                          histogram_freq=0,
                          write_graph=True,
                          write_images=True)
checkpointer = ModelCheckpoint(filepath="VAE0425_model.h5",
                               verbose=0,
                               save_best_only=True)
history=vae.fit(x_train[:100],epochs=1,batch_size=batch_size,shuffle=True,callba
cks=[ checkpointer,tensorboard]).history
                #validation_data=(x_test, None))
```

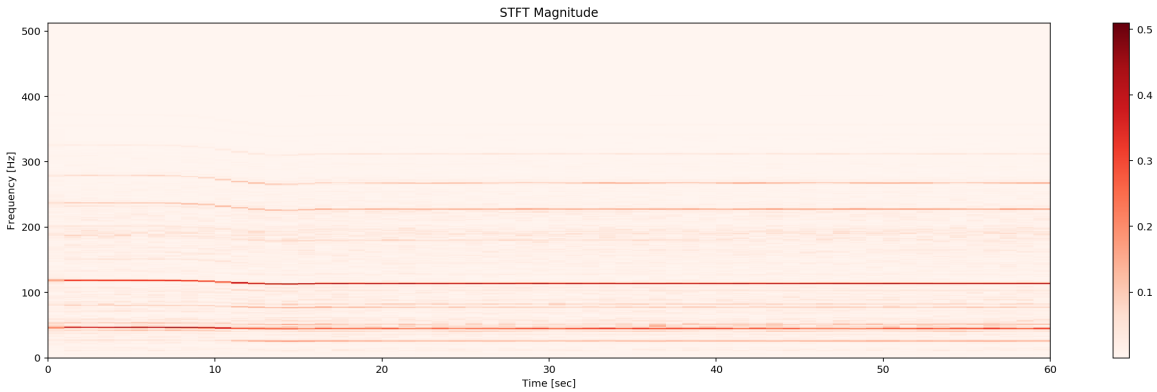
```
Train on 100 samples
100/100 [==============================] - 116s 1s/sample - loss: 73
25.8545

//anaconda3/lib/python3.7/site-packages/keras/callbacks/callbacks.p
y:707: RuntimeWarning: Can save best model only with val_loss availa
ble, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
```

## Reconstruction Error

In [40]:

```
#Get the prediction of STFT for test data
k=20
outputs=vae.predict(tf.reshape(x_test[k],[1,513,121,1]))
STFT_predction=tf.reshape(outputs, [513, 121])

def plot_STFT_imshow(data):
    plt.figure(figsize=(15,4))
    plt.imshow(data, cmap='Reds',aspect='auto')
    plt.title('STFT Magnitude')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.colorbar()
    plt.show()

plot_STFT_imshow(STFT_predction)
plot_STFT(xtest[k])
```




In [41]:

```
STFT_predction=vae.predict(x_test,batch_size=batch_size)
```

In [44]:

```
MAE=[]
MSE=[]
for i in range(len(x_test)):
    inputs=tf.reshape(x_test[i],[513,121])[:,1:]
    pred=tf.reshape(STFT_predction[i], [513, 121])[:,1:]
    MAE.append(mean_absolute_error(inputs, pred))
    MSE.append(mean_squared_error(inputs, pred,squared=False))
MSE=np.asarray(MSE)
MAE=np.asarray(MAE)
print(MAE[:20])
print(MSE[:20])
```

```
[0.00289638 0.0018085  0.01803598 0.00612773 0.00464895 0.00317461
 0.01407987 0.01283127 0.00166958 0.0349665  0.01240927 0.00334508
 0.00237204 0.00171939 0.00554883 0.00193388 0.0025196  0.00316206
 0.00427051 0.00404185]
[0.00866966 0.00390684 0.0495273  0.01114647 0.00850514 0.00685783
 0.04287197 0.04260274 0.00649017 0.09143113 0.03554526 0.0069261
 0.00566253 0.00422288 0.00969066 0.00550283 0.00524909 0.00748929
 0.00911729 0.00753795]
```

In [45]:

```
MSE[2433]
MSE[2019]
# baseline LOF
pca1 = PCA(n_components = 10, whiten = True)
pca1.fit(xtrain)
xtrain_fpca = pca1.fit_transform(xtrain[:,:])
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = Tru
e)
lof1.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca1.fit_transform(xtest)
sscore = -lof1.score_samples(xtest_fpca)
print(sscore[:100])
```

Out[45]:

```
0.005345745096942441
```

In [46]:

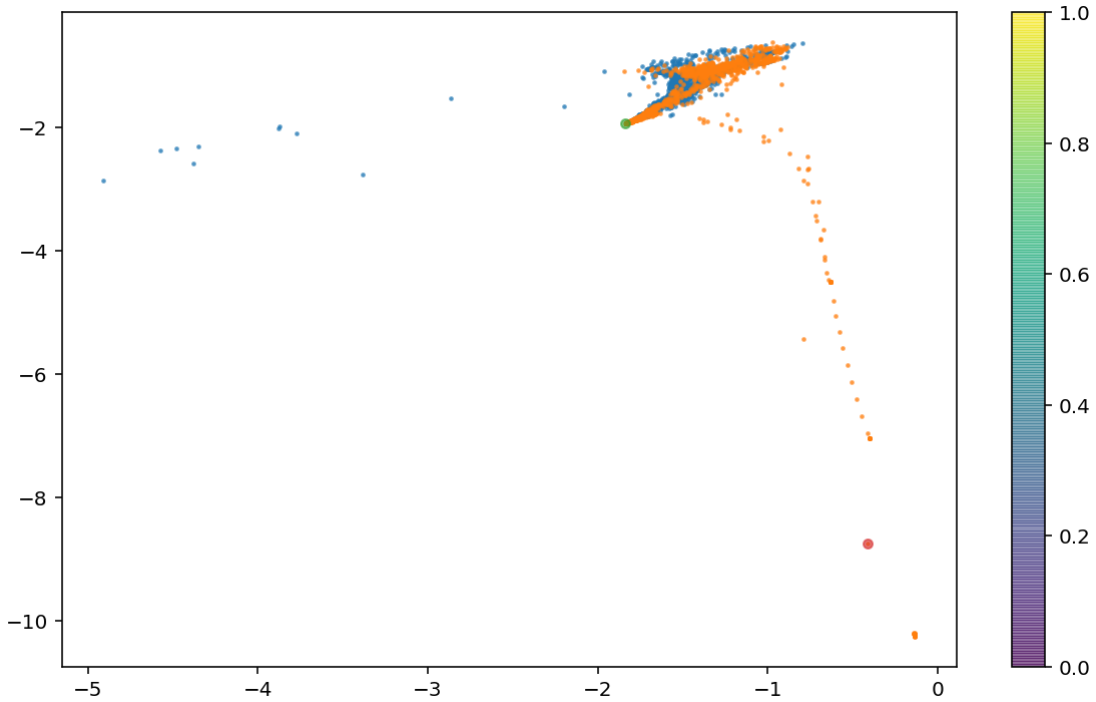
```
# Save the anomaly scores to file
print(MSE.shape)
#0.627493603767
#0.621231784642
np.savetxt('ytest_challenge_student15.csv', MSE, fmt = '%1.6f', delimiter=',')
np.savetxt('ytest_challenge_student16.csv', MAE, fmt = '%1.6f', delimiter=',')
```

```
(2511,)
```

## VAE Latent Space Visualisation

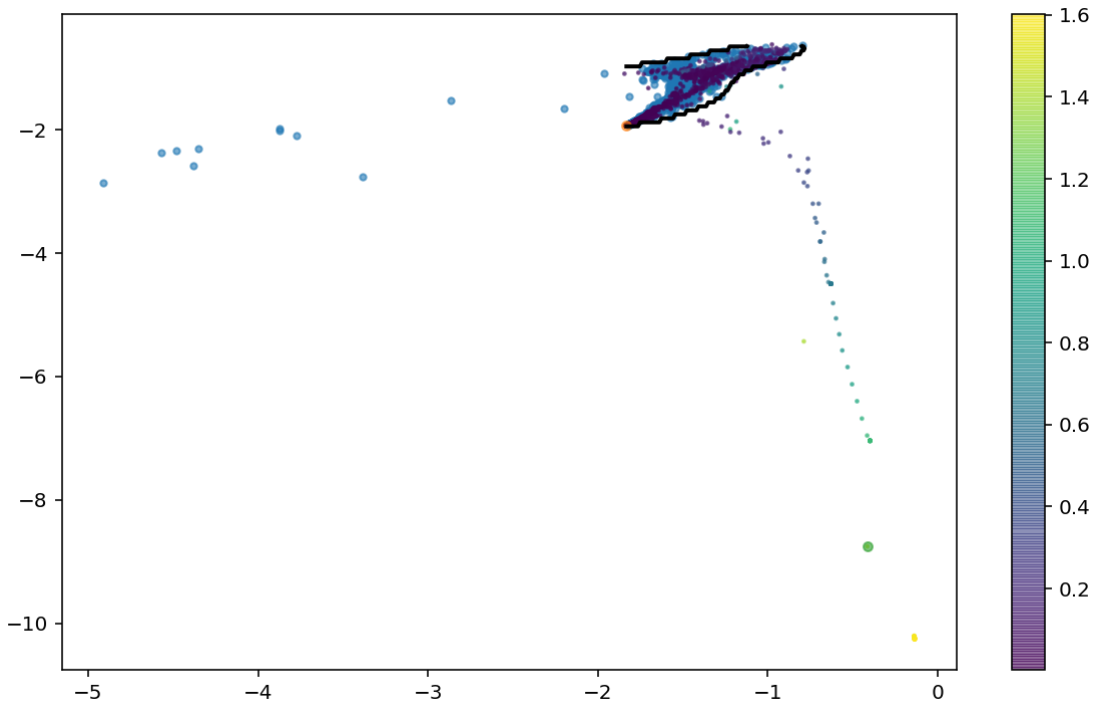
In [296]:

```
# display a 2D plot in the latent space
X_train_encoded = encoder.predict(x_train, batch_size=batch_size)
X_test_encoded = encoder.predict(x_test, batch_size=batch_size)
plt.figure(figsize=(10, 6))
plt.scatter(X_train_encoded[0][:, 0], X_train_encoded[0][:, 1],alpha=0.6,s=2)
plt.scatter(X_test_encoded[0][:, 0], X_test_encoded[0][:, 1],alpha=0.6,s=2)
plt.scatter(X_test_encoded[0][2019, 0], X_test_encoded[0][2019, 1],alpha=0.6,s=2
0)
plt.scatter(X_test_encoded[0][2433, 0], X_test_encoded[0][2433, 1],alpha=0.6,s=2
0)
plt.colorbar()
plt.show()
```



In [310]:

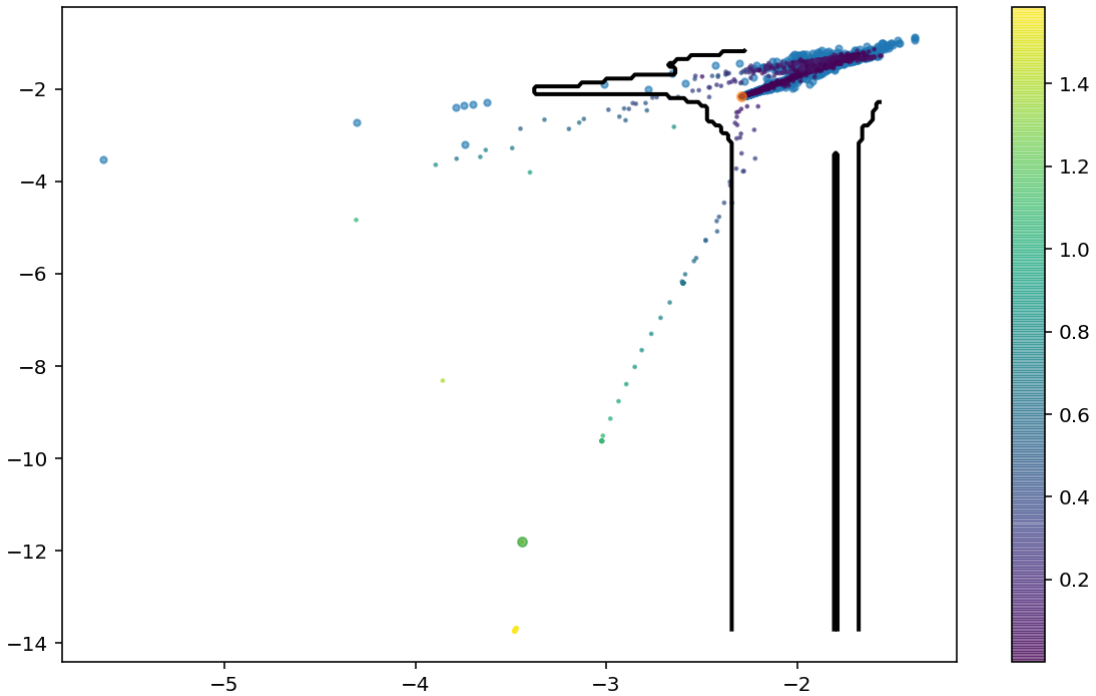
```
X = X_train_encoded[0]
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=4).fit(X)
clf.predict(X)
c=clf.score_samples(X)

# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(X_test_encoded[0][:,0].min(),X_test_encoded[0]
[:,0].max(), 150),
                     np.linspace(X_test_encoded[0][:,1].min(),X_test_encoded[0]
[:,1].max(), 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(10, 6))
plt.scatter(X_train_encoded[0][:, 0], X_train_encoded[0][:, 1],alpha=0.6,s=10)
plt.scatter(X_test_encoded[0][:, 0], X_test_encoded[0][:, 1],alpha=0.6,s=2,c=MSE
)
plt.colorbar()
plt.scatter(X_test_encoded[0][2019, 0], X_test_encoded[0][2019, 1],alpha=0.6,s=2
0)
plt.scatter(X_test_encoded[0][2433, 0], X_test_encoded[0][2433, 1],alpha=0.6,s=2
0)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

plt.show()
```




**IsolationForest**

In [283]:

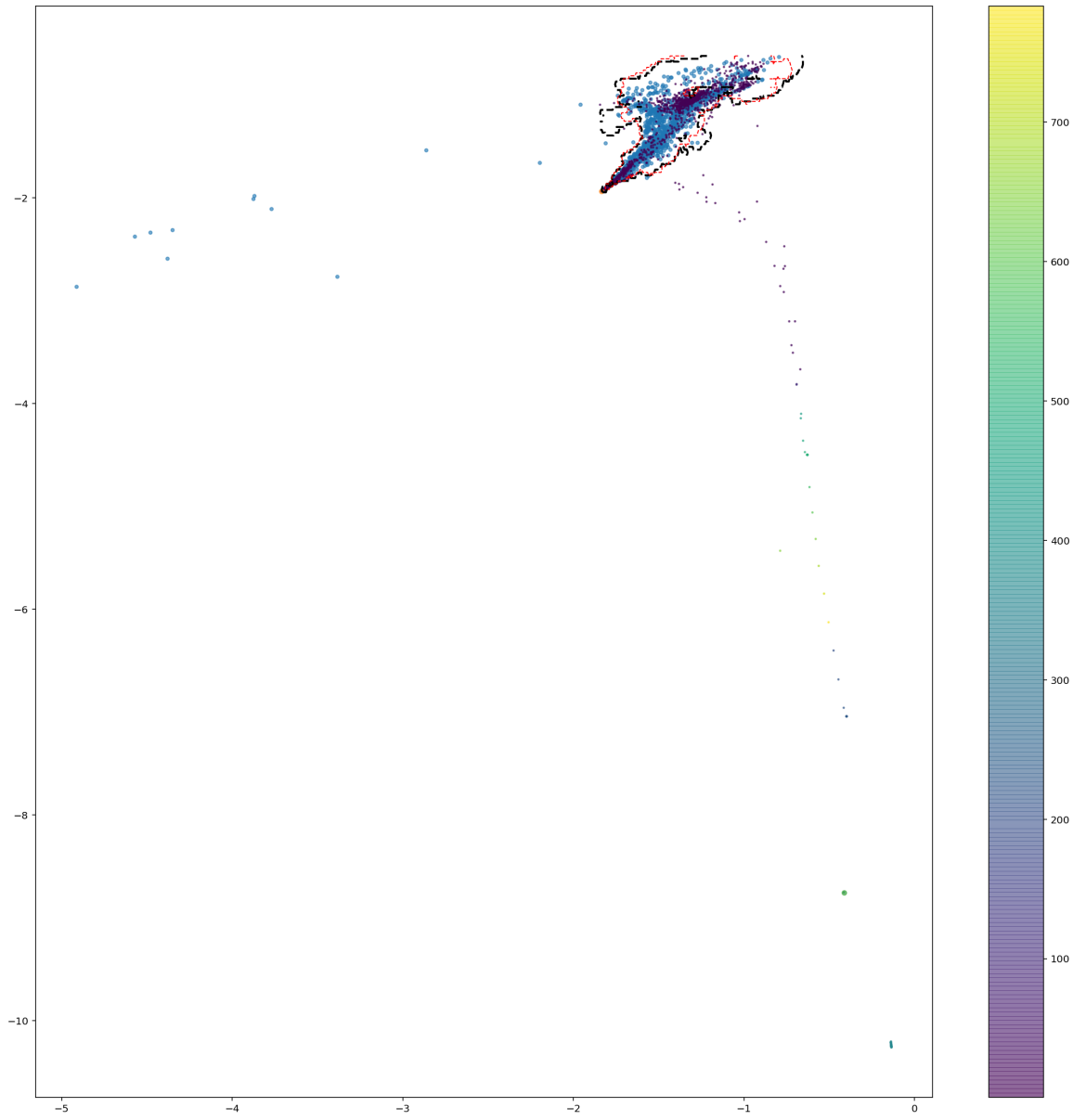
```
clf = IsolationForest(n_estimators=200, max_samples='auto', contamination=0.01,
                      max_features=1.0,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0, warm_start=True
                     ).fit(X)
clf.score_samples(X)
clf.predict(X[:100])
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(X_test_encoded[0][:,0].min(),X_test_encoded[0]
[:,0].max(), 150),
                     np.linspace(X_test_encoded[0][:,1].min(),X_test_encoded[0]
[:,1].max(), 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(10, 6))
plt.scatter(X_train_encoded[0][:, 0], X_train_encoded[0][:, 1],alpha=0.6,s=10)
plt.scatter(X_test_encoded[0][:, 0], X_test_encoded[0][:, 1],alpha=0.6,s=2,c=MSE
)
plt.colorbar()
plt.scatter(X_test_encoded[0][2019, 0], X_test_encoded[0][2019, 1],alpha=0.6,s=2
0)
plt.scatter(X_test_encoded[0][2433, 0], X_test_encoded[0][2433, 1],alpha=0.6,s=2
0)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

plt.show()
```



In [298]:

```
# Fit the low-dimensional method
lof = LocalOutlierFactor(n_neighbors = 2 ,contamination = 0.02, novelty = True)
lof.fit(X_train_encoded[0])
# Calculate anomaly score on the (PCA-transformed) test data
sscore = -lof.score_samples(X_test_encoded[0])


# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(X_test_encoded[0][:,0].min(),X_test_encoded[0]
[:,0].max(), 350),
                     np.linspace(X_test_encoded[0][:,1].min(),X_test_encoded[0]
[:,1].max(), 350))
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(20, 20))
plt.scatter(X_train_encoded[0][:, 0], X_train_encoded[0][:, 1],alpha=0.6,s=10)
plt.scatter(X_test_encoded[0][:, 0], X_test_encoded[0][:, 1],alpha=0.6,s=2,c=ssc
ore)
plt.colorbar()
plt.scatter(X_test_encoded[0][2019, 0], X_test_encoded[0][2019, 1],alpha=0.6,s=2
0)
plt.scatter(X_test_encoded[0][2433, 0], X_test_encoded[0][2433, 1],alpha=0.6,s=2
0)
#plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')


lof = LocalOutlierFactor(n_neighbors = 5 ,contamination = 0.02, novelty = True)
lof.fit(X_train_encoded[0])
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[2], linewidths=2, colors='black')
sscore = -lof.score_samples(X_test_encoded[0])

lof = LocalOutlierFactor(n_neighbors = 10 ,contamination = 0.02, novelty = True)
lof.fit(X_train_encoded[0])
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[2], linewidths=1, colors='red')
sscore = -lof.score_samples(X_test_encoded[0])
print(sscore[:10])
plt.show()
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:26: Us
erWarning: No contour levels were found within the data range.
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:33: Us
erWarning: No contour levels were found within the data range.

[1.57961424 1.07322015 1.4431127  1.86366691 0.9699948  1.06917981
 1.03822776 1.07182033 1.6861587  1.21862049]
```


−2
−4
−6
−8
−10
−5
−4
−3
−2
−1
0
700
600
500
400
300
200
100

In [285]:

```
# Save the anomaly scores to file
print(sscore.shape)
#0.580314435092
#0.598135822611
np.savetxt('ytest_challenge_student18.csv', sscore, fmt = '%1.6f', delimiter=','
)
```

```
(2511,)
```

# Raw Signal Time Series Approach

## LSTM

In order to use a LSTM to learn the time series, I divide each 60s sequence into smaller sequences. However, here the anormly doesn't lie in the general pattern of the time series, so LSTM learned is not the best solution in this case.

In [433]:

```
def get_batch(source, i, seq_len, evaluation=False):
    # Deal with the possibility that there's not enough data left for a full seq
uence
    seq_len = min(seq_len, len(source) - 1 - i)
    # Take the input data seq
    data = source[:,i*seq_len:i*seq_len+seq_len]
    # target data is the next slice after input data
    target = source[:,(i+1)*seq_len+seq_len]
    return data, target
```

In [164]:

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

"""for i in range(4):
    if i ==0:
        xtrain_STFT=np.abs(Zxx)
    else:
        Zxx=cal_STFT(xtrain[i])
        xtrain_STFT=np.dstack([xtrain_STFT,np.abs(Zxx)])
xtrain_STFT = np.moveaxis(xtrain_STFT, -1, 0)"""

seq_len = 3
model = Sequential()
model.add(layers.LSTM(units=268, input_shape=(
    1024, seq_len), return_sequences=False))

model.add(layers.Dense(512))
#model.add(layers.Dense(512))

model.compile(optimizer=RMSprop(lr=0.01), loss='mae')
#history=model.fit(data_batch,targets_batch,steps_per_epoch=5,          epochs=
2)
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 268)               291584
_________________________________________________________________
dense_4 (Dense)              (None, 512)               137728
=================================================================
Total params: 429,312
Trainable params: 429,312
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
# k for each 60s data
for k in range(20):

    # i:range(0,61,2)
    for i in (range(0, xtrain_STFT_3D[k].shape[-1])):
        if i+seq_len+1 < xtrain_STFT_3D[k].shape[-1]:
            # get data:(513*seq_len)  targets:(513*1)
            data, targets = get_batch(xtrain_STFT_3D[k], i, seq_len)

        if i == 0:
            data_seq = data
            targets_seq = targets
            pred_seq = pred
        else:
            if data.shape[1] != seq_len:
                pass
            else:
                # build a batch of data/targets for xtrain_STFT_3D[k]
                data_seq = np.dstack([data_seq, data])
                targets_seq = np.dstack([targets_seq, targets])
                pred_seq = np.dstack([pred_seq, pred])

        # print(data_seq.shape)
    data_seq = np.moveaxis(data_seq, -1, 0)
    targets_seq = np.moveaxis(targets_seq, -1, 0)

    pred_seq = model.predict(data_seq, batch_size=32, verbose=1)

    # Build k batchs of data/targets
    if k == 0:
        data_batch = data_seq
        targets_batch = targets_seq
        pred_batch = pred_seq
        print(pred_seq.shape)
        print(pred_batch.shape)

    else:
        data_batch = np.dstack([data_batch, data_seq])
        targets_batch = np.dstack([targets_batch, targets_seq])
        pred_batch = np.dstack([pred_batch, pred_seq])
        print(pred_seq.shape)
        print(pred_batch.shape)

    #targets_batch= np.moveaxis(targets_batch, -1, 1)
    targets_seq_squeezed = np.squeeze(targets_seq)

    pred_batch = np.moveaxis(pred_batch, -1, 0)
    print(data_seq.shape)
    print(data_batch.shape)
    print(targets_seq.shape)
    print(targets_batch.shape)
    history = model.fit(data_seq, targets_seq_squeezed, steps_per_epoch=5, verbo
se=1,
                        epochs=2)
```

# LOF + PCA

Using the projection on a low-dimensional space

- Features: Raw singal
- Data: train data
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :0.7143**

In [168]:

```
# PCA transform
pca1 = PCA(n_components=10, whiten=True)
pca1.fit(xtrain[:, :])
xtrain_fpca = pca1.fit_transform(xtrain[:, :])
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca1.fit_transform(xtest)
sscore = -lof1.score_samples(xtest_fpca)
print(sscore[:100])
```

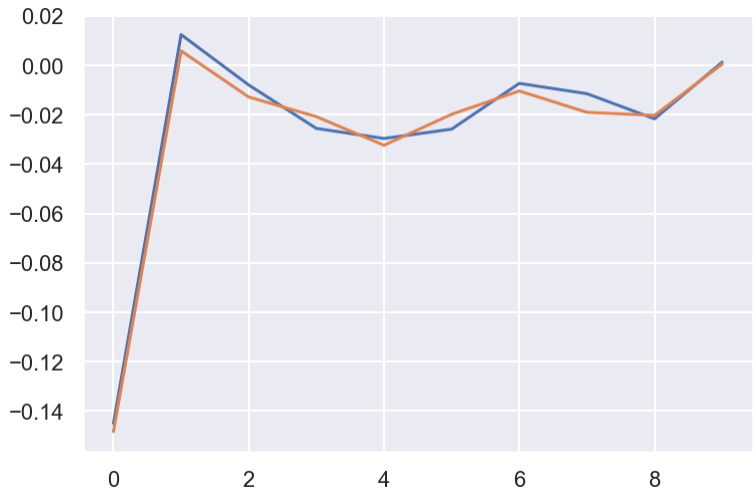
```
[2.31222147 3.21920169 2.32452172 5.2416519  3.77868812 2.48401327
 1.60473781 2.08579464 3.62659859 1.51251174 2.38662206 2.50738985
 2.53381672 3.7031445  2.54672502 1.7408619  3.73473917 2.84198658
 2.96931812 2.00665609 2.59616066 2.41765589 2.11218523 4.8581478
 3.82995185 2.34813006 2.33433001 2.87554115 2.21877324 3.45815205
 2.33508349 2.77534772 2.63497815 2.28615506 2.65251437 5.98637398
 1.74203358 3.02999749 2.63603981 3.89133284 2.99970299 2.93207293
 3.18996343 2.84854187 3.69942921 3.21495621 2.09673527 3.04956052
 2.07690722 3.20542142 2.48597064 2.08116068 2.89436276 2.53010919
 1.72682643 2.17525652 2.33569795 2.07993293 2.37915679 2.44682609
 1.88143338 2.51731689 1.88324066 2.70340383 2.64562268 2.45633819
 2.59444114 2.44419078 3.26238809 3.18518807 3.21228093 2.85333342
 3.11405378 2.7419172  2.24284782 2.23024267 3.07355636 2.06936584
 1.96645779 2.47559168 2.34208299 3.33483183 1.72857621 2.32563373
 2.7811836  1.36257703 2.81461841 3.46023745 2.9316191  2.81334974
 2.85784704 2.76561606 2.4402141  2.99155279 2.29768907 2.26602091
 2.1637626  2.7569695  2.33752717 4.35549251]
```

In [169]:

```
plt.plot(xtest_fpca[4])
plt.plot(xtest_fpca[2019])
```

Out[169]:

```
[<matplotlib.lines.Line2D at 0x22f9b5668>]
```




## Kernel PCA

In [173]:

```
from sklearn.decomposition import KernelPCA
pca1 = KernelPCA(n_components = 10, kernel='rbf')
#pca1 = PCA(n_components = 10,whiten = True)
pca1.fit(xtrain)
xtrain_fpca = pca1.fit_transform(xtrain)

# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = Tru
e)
lof1.fit(xtrain_fpca)

xtest_fpca = pca1.transform(xtest)

sscore = -lof1.score_samples(xtest_fpca )
print(sscore[:10])
```

```
[1.04133824 2.03553452 0.99993827 1.08801806 3.13066476 1.07867853
 0.97939616 0.97188638 1.00060463 1.23200188]
```

# OneClassSVM

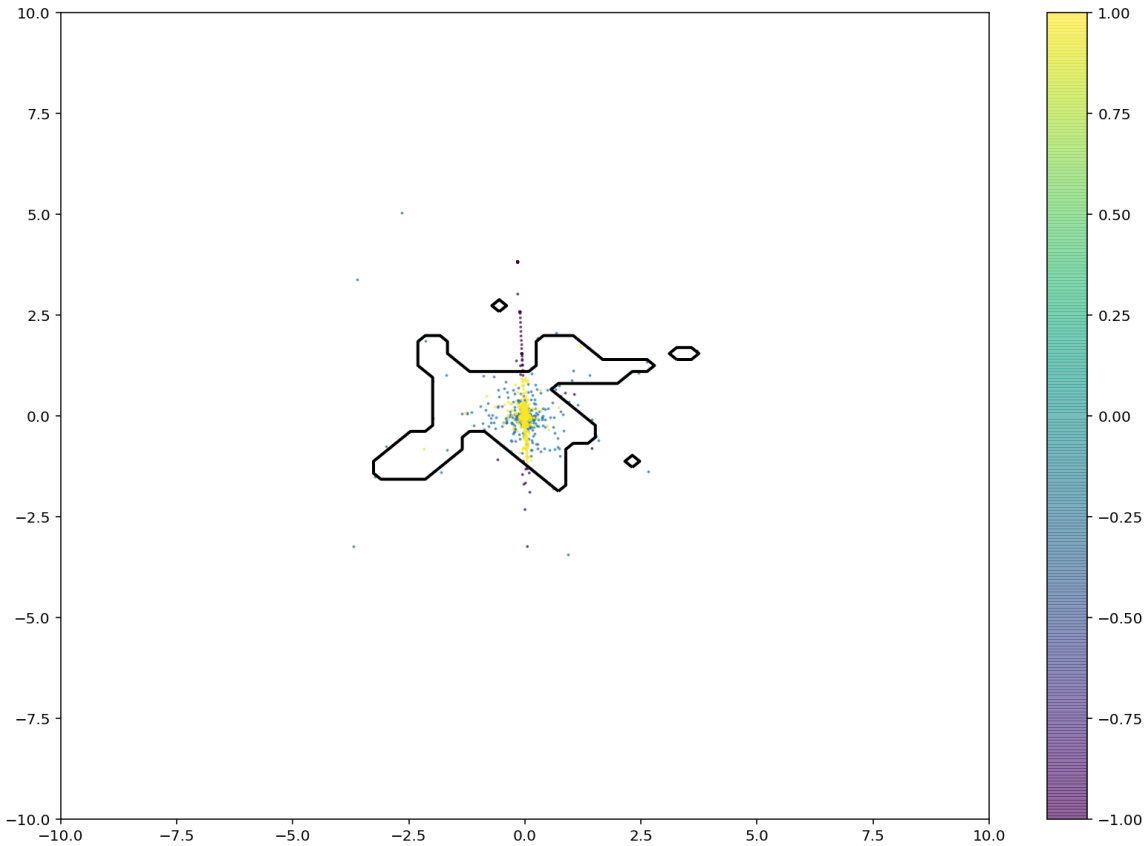
- Features: Raw singal+der1+der2
- Data: train data
- Algorithm: OneClassSVM
- Parameter: Novelty
- Scoring:
- **AUC :0.632120830089.**

In [34]:

```
clf = OneClassSVM(kernel='rbf',gamma='auto').fit(xtrain)
#clf.predict(Xtest)
sscore=clf.score_samples(Xtest)
```

In [466]:

```
pca2d = PCA(n_components=2, whiten=True)
# xtrain_fpca2d=pca2d.fit_transform(xtrain_scaled)
X = xtrain_fpca2d
clf = OneClassSVM(nu=0.01, kernel="rbf", gamma=1).fit(X)
c = clf.predict(xtest_fpca2d)
#############
xmin = np.min([xtest_fpca2d[:, 0].min(), xtrain_fpca2d[:, 0].min()])
xmax = np.max([xtest_fpca2d[:, 0].max(), xtrain_fpca2d[:, 0].max()])
ymin = np.min([xtest_fpca2d[:, 1].min(), xtrain_fpca2d[:, 1].min()])
ymax = np.max([xtest_fpca2d[:, 1].max(), xtrain_fpca2d[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(14, 10))
plt.xlim(-10, 10)
plt.ylim(-10, 10)
plt.scatter(xtrain_fpca2d[:, 0], xtrain_fpca2d[:, 1],
            alpha=0.6, s=1)  # ,c=MSE)
plt.scatter(xtest_fpca2d[:, 0], xtest_fpca2d[:, 1], alpha=0.6, s=1, c=c)
plt.colorbar()
#plt.scatter(X_test_encoded[0][2019, 0], X_test_encoded[0][2019, 1],alpha=0.6,s=
20)
#plt.scatter(X_test_encoded[0][2433, 0], X_test_encoded[0][2433, 1],alpha=0.6,s=
20)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

plt.show()
```




# Isolation Forest Raw data

- Features: interpolated data
- Features dimenstion: 61440
- Data: train data
- Algorithm: Isolation Forest
- Parameter:
- Scoring:
- **AUC : 0.623563165116**

In [ ]:

```
clf = IsolationForest(n_estimators=100, max_samples='auto', contamination='auto'
,
                      max_features=1.0,
                      bootstrap=False, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0, warm_start=False
                     ).fit(xtrain[:])
clf.predict(xtest[:100])
sscore = -clf.score_samples(xtest)
print(sscore[:100])
```

# LOF + PCA + Derivatives

- Features: Raw singal + Derivatives 1 + Derivatives 2
- Features dimenstions: 61440*3
- Data:
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :0.634** contamination='auto',neigbour=5
- **AUC :0.605713967889** contamination='0.02' ,neigbour=3

In [ ]:

```
# interpolation matrix
def derivative_matrix(xtrain):
    xnew = np.arange(0, 61440, 1)
    for i in range(len(xtrain)):

        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i], s=0)
        #ynew = interpolate.splev(xnew, tck, der=0)
        yder1 = interpolate.splev(xnew, tck, der=1)
        #yder2 = interpolate.splev(xnew, tck, der=2)
        if i % 200 == 0:
            print(i)
        if i == 0:
            interpolation = interpolate.splev(xnew, tck, der=0)
            derivative1=interpolate.splev(xnew, tck, der=1)
            derivative2=interpolate.splev(xnew, tck, der=2)
        else:
            #interpolation = np.vstack((interpolation, ynew))
            derivative1=np.vstack((derivative1,yder1))
            #derivative2=np.vstack((derivative2,yder2))
    return interpolation,derivative1, derivative2


#xtrain_interpolation, xtrain_derivative1, xtrain_derivative2 = derivative_matri
x(
 #   xtrain)
xtest_interpolation, xtest_derivative1, xtest_derivative2 = derivative_matrix(
    xtest)
```

In [41]:

```
#np.savetxt('xtrain_der1_X1.csv', xtrain_derivative1, fmt = '%1.6f', delimiter
=',')
np.savetxt('xtest_der1_X1.csv', xtest_derivative1, fmt = '%1.6f', delimiter=',')
```

In [31]:

```
#2nd order derivative

# PCA transform
pca = PCA(n_components = 10, whiten = True)
xtrain_fpca = pca.fit_transform(xtrain_derivative2[:,:])
# Fit the low-dimensional method
lof = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = True
)
lof.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca.fit_transform(xtest_derivative2)
sscore = -lof.score_samples(xtest_fpca)
print(sscore[:100])
```

```
[1.9657198  1.89369335 2.71453218 2.14009202 2.20136775 2.46513439
 2.07411111 1.35850272 1.85261668 2.47175589 1.48242438 2.21448029
 2.28309438 2.2066407  2.41108312 3.0308287  2.18820082 1.62349879
 3.1400977  2.49080711 1.18929089 2.47646837 2.05284307 1.99468207
 2.18218016 2.57229875 2.3292533  1.87836024 1.45782464 2.23660653
 2.04563339 2.17438899 1.57928881 2.54198428 1.32957589 1.91710495
 2.20723822 1.46594127 1.86789527 1.91628492 2.14031512 2.17470193
 1.83302977 2.1966174  2.20770657 2.25973704 1.82274552 1.46202263
 2.00424114 1.56239172 2.31952236 1.53690084 2.18453936 2.1030268
 1.78425253 2.30196645 2.18780778 2.55552586 1.89328896 2.51818672
 2.20046202 2.35402657 2.49411554 2.18085714 2.31706203 3.03510245
 1.92146982 2.02760991 1.42323704 1.84803682 1.61381357 2.24932298
 2.18218241 2.01969501 1.73088564 1.59496846 1.89269809 1.73224706
 1.99320935 2.48486155 1.76472983 2.17538929 2.76379497 2.48048523
 2.3545205  2.42354869 2.19281497 2.46080312 3.32599624 2.27491265
 1.74616344 2.90330104 2.13862312 2.47999414 1.86202863 2.16732512
 3.13488846 2.12129493 2.47562085 1.9378683 ]
```

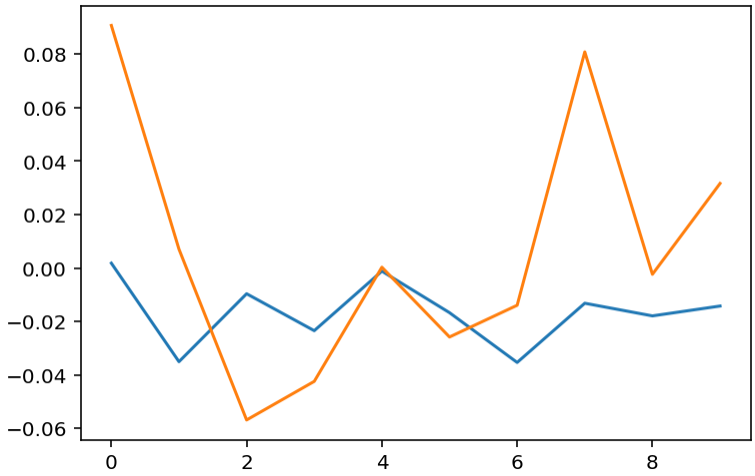
In [35]:

```
print(sscore[2019])
print(sscore[2433])
plt.plot(xtest_fpca[2019])
plt.plot(xtest_fpca[24])
```

```
2.1955592080920874
2.233305828807223
```

Out[35]:

```
[<matplotlib.lines.Line2D at 0x11e8f1b00>]
```




In [36]:

```
# Save the anomaly scores to file
print(sscore.shape)
np.savetxt('ytest_challenge_student8.csv', sscore, fmt = '%1.6f', delimiter=',')
```

```
(2511,)
```

In [33]:

```
k=np.hstack((xtrain[:5], xtrain_derivative1[:5]))
k.shape
xtrain_derivative2[:5].shape
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-33-efaff975cd13> in <module>
----> 1 k=np.hstack((xtrain[:5], xtrain_derivative1[:5]))
      2 k.shape
      3 xtrain_derivative2[:5].shape

TypeError: 'int' object is not subscriptable
```

In [5]:

```
import numpy as np
xtrain_derivative1 = np.loadtxt(
    'xtrain_der1_X1.csv',  delimiter=',')
xtest_derivative1 = np.loadtxt(
    'xtest_der1_X1.csv',  delimiter=',')
```

In [6]:

```
xtrain = np.loadtxt('airbus_train.csv', delimiter= ' ')
print(xtrain.shape)
xtest = np.loadtxt('airbus_test.csv', delimiter= ' ')
print(xtest.shape)
```

```
(1677, 61440)
(2511, 61440)
```

In [8]:

```
xtrain_derivative2 = np.loadtxt(
    'xtrain_der2_X1.csv',  delimiter=',')
xtest_derivative2 = np.loadtxt(
    'xtest_der2_X1.csv',  delimiter=',')
```

In [9]:

```
Xtrain = np.hstack(
    (np.hstack((xtrain, xtrain_derivative1)), xtrain_derivative2))
xtrain_derivative1=0
xtrain_derivative2=0
xtrain=0
```

In [10]:

```
Xtest = np.hstack(
    (np.hstack((xtest, xtest_derivative1)), xtest_derivative2))
```

In [12]:

```
np.savetxt('Xtrain_xder1der2_X1.csv', Xtrain, fmt = '%1.6f', delimiter=',')
np.savetxt('Xtest_xder1der2_X1.csv', Xtest, fmt = '%1.6f', delimiter=',')
```

In [28]:

```
# PCA transform
pca = PCA(n_components = 100, whiten = True)
xtrain_fpca = pca.fit_transform(Xtrain)
# Fit the low-dimensional method
lof = LocalOutlierFactor(n_neighbors = 3 ,contamination = 0.02, novelty = True)
lof.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca.fit_transform(Xtest)
sscore = -lof.score_samples(xtest_fpca)
print(sscore[:100])
```

```
[7.50622507 3.86520882 3.32975609 4.52529125 4.45480612 3.90876835
 3.63493963 5.97026324 4.28146037 4.93607468 7.70226021 3.78271975
 3.98802516 4.23438377 3.95705119 4.74002022 3.50773592 4.11545262
 3.68818656 4.50619935 4.69065993 4.80546824 4.04409106 4.83160186
 6.3936794  4.11205354 3.87467241 4.05699438 4.43256059 3.98467251
 2.80108062 3.5422597  4.51179453 3.91992717 5.55568172 4.55613969
 4.11261856 2.9236154  3.14939203 4.44395722 3.91946878 4.49711229
 4.32656275 3.907553   3.84453833 4.17645336 9.72486055 4.07683804
 4.80175048 2.74408388 3.43758017 5.33151827 3.97071217 5.45253351
 4.60981764 4.41924586 3.77658276 6.59953766 3.53471133 3.80661963
 6.52097732 4.00685744 4.21081324 3.8961832  2.6366209  3.58136571
 5.54887849 4.07154448 2.81249747 3.48593397 3.36170104 3.54616866
 3.86046898 4.16263916 6.3269842  5.22071969 3.80735558 3.70582189
 6.14297969 4.26695605 3.94358637 4.12735538 6.11501779 3.8524693
 3.53082237 3.56601758 3.88437897 3.92912534 3.89616352 3.82607112
 3.36240029 5.84937521 3.93222561 4.16161312 9.18148479 4.46393922
 6.27423249 3.12656612 4.12243296 5.0843745 ]
```

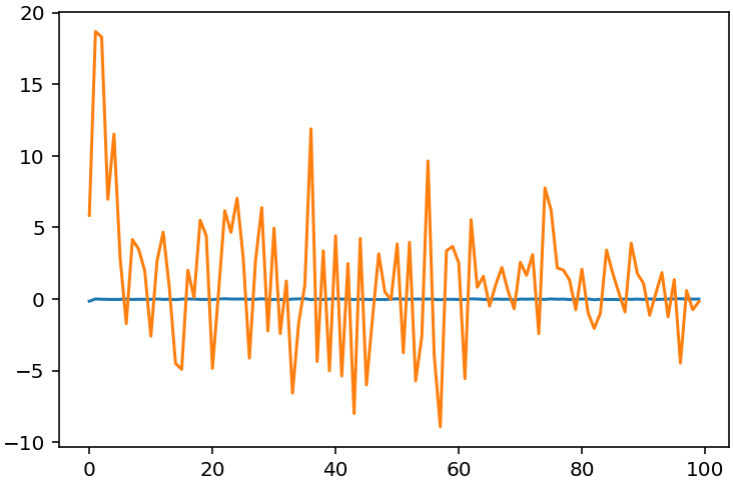
In [30]:

```
print(sscore[2019])
print(sscore[2433])
plt.plot(xtest_fpca[2019])
plt.plot(xtest_fpca[2433])
plt.plot(xtest_fpca[2434])
plt.plot(xtest_fpca[2435])
pca.explained_variance_ratio_[:5]
```

```
3.903252170507732
7.7768009926364705
```

Out[30]:

```
array([0.88093371, 0.01077892, 0.00413147, 0.00316665, 0.00199988])
```




# LOF interpolated data

- Features: interpolated data
- Features dimenstion: 61440*2
- Data: train data
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :0.52973475719**

In [14]:

```
# interpolation matrix
def interpolation_matrix(xtrain):
    xnew = np.arange(0, 61440, 1/2)
    for i in range(len(xtrain)):

        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i], s=0)
        ynew = interpolate.splev(xnew, tck, der=0)
        yder1 = interpolate.splev(xnew, tck, der=1)
        yder2 = interpolate.splev(xnew, tck, der=2)
        if i % 100 == 0:
            print(i)
        if i == 0:
            interpolation = ynew
            derivative1=yder1
            derivative2=yder2
        else:
            interpolation = np.vstack((interpolation, ynew))
            # derivative1=np.vstack((derivative1,yder1))
            # derivative2=np.vstack((derivative2,yder2))
    return interpolation, derivative1, derivative2


xtrain_interpolation, xtrain_derivative1, xtrain_derivative2 = interpolation_mat
rix(
    xtrain)
xtest_interpolation, xtest_derivative1, xtest_derivative2 = interpolation_matrix
(
    xtest)
```

```
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
```

In [15]:

```
xtest_interpolation.shape
np.savetxt('xtrain_interpolationX2.csv',
           xtrain_interpolation, fmt='%1.6f', delimiter=',')
np.savetxt('xtest_interpolationX2.csv',
           xtest_interpolation, fmt='%1.6f', delimiter=',')
```

In [33]:

```
# PCA transform
pca1 = PCA(n_components=10, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_interpolation)
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors=2, contamination='auto', novelty=True)
# n_neighbors=10 AUC 0.408336114304
# n_neighbors=3 AUC 0.63531276651
# n_neighbors=2 AUC 0.649104836417
lof1.fit(xtrain_fpca)
# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca1.fit_transform(xtest_interpolation)
sscore = -lof1.score_samples(xtest_fpca)
print(sscore[:100])
```

```
[2.44608028 1.20979371 1.37994128 4.39178276 1.20252021 1.32738297
 1.50007418 1.40581575 1.34423649 1.76622263 1.34666284 1.21258653
 1.2153796  1.73399158 1.22673102 1.43962163 1.23088521 1.18514108
 1.24965326 1.21260342 1.32080322 1.21626098 1.17573295 1.50393369
 4.20098683 1.19165419 1.18218612 1.18644271 1.36615569 1.22475717
 2.19806899 1.19201142 1.39802541 1.14792968 2.61491177 2.9487786
 1.24558579 3.62474499 1.79907625 2.34036014 1.37500319 1.23074342
 1.27815007 1.1870581  1.19505357 1.13698656 1.79126795 1.22644014
 1.19005779 1.11830462 1.40898562 1.81693093 1.16882405 1.0786123
 2.50779501 1.22870927 1.17306883 1.74194629 1.20267528 1.23351206
 1.56333707 1.29310016 2.62804277 1.20430337 3.29462154 1.24524194
 2.84010488 1.23395688 1.42509432 3.5555574  4.69884524 1.22047924
 2.20274324 1.27639435 2.55693042 1.33395909 1.14909813 1.32523576
 3.01582116 1.24647039 1.22085888 1.09295067 1.66009465 1.18404245
 1.27594041 1.18993434 1.17473509 1.21345797 1.17807037 1.23218048
 1.18825151 1.98248893 2.75783858 1.24774629 3.20299447 1.22027401
 1.17874733 2.93871593 1.18060206 2.82448725]
```

In [34]:

```
print(sscore[2019])
print(sscore[2433])
pca1.explained_variance_ratio_
```

```
1.1860836140297217
4.382270555059261
```

Out[34]:

```
array([0.91600621, 0.01113232, 0.00421567, 0.00323661, 0.00200126,
       0.00194058, 0.00147035, 0.00142686, 0.00135086, 0.00124056])
```

# Isolation Forest Interpolation / 1st order derivative

- Features: interpolated data / 1st order derivative
- Features dimenstion: 61440*2
- Data: train data
- Algorithm: Isolation Forest
- Parameter:
- Scoring:
- **AUC :**

In [72]:

```
# interpolation matrix
def interpolation_matrix(xtrain):
    xnew = np.arange(0, 61440, 1/2)
    for i in range(len(xtrain)):

        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i], s=0)
        #ynew = interpolate.splev(xnew, tck, der=0)
        yder1 = interpolate.splev(xnew, tck, der=1)
        yder2 = interpolate.splev(xnew, tck, der=2)
        if i % 100 == 0:
            print(i)
        if i == 0:
            interpolation = interpolate.splev(xnew, tck, der=0)
            derivative1 = interpolate.splev(xnew, tck, der=1)
            derivative2 = interpolate.splev(xnew, tck, der=2)
        else:
            #interpolation = np.vstack((interpolation, ynew))
            # derivative1=np.vstack((derivative1,yder1))
            derivative2 = np.vstack((derivative2, yder2))
    return interpolation, derivative1, derivative2


# xtrain_interpolation, xtrain_derivative1, xtrain_derivative2 = interpolation_m
atrix(
#    xtrain)
xtest_interpolation, xtest_derivative1, xtest_derivative2 = interpolation_matrix
(
    xtest)
```

```
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
```

In [39]:

```
xtrain_derivative2.shape
```

Out[39]:

```
(122880,)
```

In [74]:

```
np.savetxt('xtrain_derivative1_interpolationX2.csv',
           xtrain_derivative1, fmt='%1.6f', delimiter=',')
np.savetxt('xtrain_derivative2_interpolationX2.csv',
           xtrain_derivative2, fmt='%1.6f', delimiter=',')
np.savetxt('xtest_derivative1_interpolationX2.csv',
           xtest_derivative1, fmt='%1.6f', delimiter=',')
np.savetxt('xtest_derivative2_interpolationX2.csv',
           xtest_derivative2, fmt='%1.6f', delimiter=',')
```

In [ ]:

```
#reload saved data
xtrain_derivative1=np.loadtxt('xtrain_derivative1_interpolationX2.csv', fmt = '%
1.6f', delimiter=',')
xtrain_derivative2=np.loadtxt('xtrain_derivative2_interpolationX2.csv', fmt = '%
1.6f', delimiter=',')
xtest_derivative1=np.loadtxt('xtest_derivative1_interpolationX2.csv', fmt = '%1.
6f', delimiter=',')
xtest_derivative2=np.loadtxt('xtest_derivative2_interpolationX2.csv', fmt = '%1.
6f', delimiter=',')
```

In [ ]:

```
clf = IsolationForest(n_estimators=200, max_samples='auto', contamination='auto'
,
                      max_features=1.0,
                      bootstrap=False, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0, warm_start=False
                      ).fit(xtrain[:])
clf.predict(xtest[:100])
```

In [ ]:

```
# Calculate anomaly score on the 61 dimentions test data
sscore = -clf.score_samples(xtest[:])
print(np.argmax(sscore))
print(sscore[:100])
```

## Autoencoder with data +der1+der2

- Features: interpolated data
- Features dimenstion: 61440*3
- Data: train data
- Algorithm: convlutional autoencoder
- Parameter:
- Scoring:
- **AUC :**

In [11]:

```
Xtrain = np.loadtxt('Xtrain_xder1der2_X1.csv',  delimiter=',')
Xtest = np.loadtxt('Xtest_xder1der2_X1.csv',  delimiter=',')
Xtrain.shape
Xtrain=Xtrain.reshape(1677,3,61440)
```

Out[11]:

```
(1677, 184320)
```

In [185]:

```
import keras
#from tensorflow import keras
from keras import layers
from keras import backend as K

latent_dim = 10
#tensor=(samples, height, width, channels)

# encoder:
encoder_input = keras.Input(shape=(1, 61440, 3), name='sequence')
x = layers.Conv2D(filters=12, kernel_size=(1, 12), strides=2,
                  activation='relu', padding='same')(encoder_input)
x = layers.MaxPooling2D((1, 2))(x)
x = layers.Conv2D(filters=5, kernel_size=(1, 5), strides=2,
                  activation='relu', padding='same')(x)
x = layers.MaxPooling2D((1, 2))(x)
shape_before_flattening = K.int_shape(x)
x = layers.Flatten()(x)
encoder_output = layers.Dense(10)(x)
#encoder_output = layers.Dense(latent_dim + latent_dim)(x)
encoder = keras.Model(encoder_input, encoder_output, name='encoder')
# encoder.summary()

# decoder:
x = layers.InputLayer(input_shape=(latent_dim,))(encoder_output)
x = layers.Dense(np.prod(shape_before_flattening[1:]), activation='relu')(x)
x = layers.Reshape(shape_before_flattening[1:])(x)
x = layers.Conv2DTranspose(5, kernel_size=(1, 5), strides=(
    1, 2), padding='same', activation='relu')(x)
x = layers.UpSampling2D((1, 2))(x)
x = layers.Conv2DTranspose(12, (1, 3), strides=(
    1, 2), activation='relu', padding='same')(x)
#x = layers.Conv2D(1, 3, activation='relu')(x)
x = layers.UpSampling2D((1, 2))(x)
x = layers.Conv2DTranspose(3, (1, 3), activation='relu', padding='same')(x)
decoder_output = x
#decoder = keras.Model(encoder_output, decoder_output)

autoencoder = keras.Model(encoder_input, decoder_output, name='autoencoder')
autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequence (InputLayer)        (None, 1, 61440, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 30720, 12)      444
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 15360, 12)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 7680, 5)        305
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 3840, 5)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 19200)             0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                192010
_________________________________________________________________
input_2 (InputLayer)         (None, 10)                0
_________________________________________________________________
dense_12 (Dense)             (None, 19200)             211200
_________________________________________________________________
reshape_2 (Reshape)          (None, 1, 3840, 5)        0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 1, 7680, 5)        130
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 1, 15360, 5)       0
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 1, 30720, 12)      192
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 1, 61440, 12)      0
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 1, 61440, 3)       111
=================================================================
Total params: 404,392
Trainable params: 404,392
Non-trainable params: 0
_________________________________________________________________
```

In [88]:

```
Xtrain=np.moveaxis(Xtrain,1,0)
#autoencoder.fit(Xtrain)
```

In [111]:

```
autoencoder.compile(optimizer='adam', loss='mse')
log_dir = "logs"
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir, histogram_freq=1)
autoencoder.fit(Xtrain, Xtrain,
                epochs=1,
                batch_size=256, #callbacks=[tensorboard_callback],
                shuffle=True)
```

```
Epoch 1/1
1677/1677 [==============================] - 106s 63ms/step - loss:
0.9771
```

Out[111]:

```
<keras.callbacks.callbacks.History at 0x1501bd3c8>
```

In [ ]:

```
%tensorboard --logdir logs/fit
```

In [ ]:

```
autoencoder(tf.convert_to_tensor(xtrain[0]))
```

# VAE data+der1+der2

In [129]:

```
from tensorflow.keras import layers
from keras import backend as K
```

In [154]:

```
#prepare data for VAE
Xtrain = Xtrain.reshape(1677, 3,61440).astype('float32')
```

In [ ]:

```
# Train VAE
original_dim = 61440
intermediate_dim = 64
latent_dim = 32
time_window_size = 5


class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon


# Define encoder model.
original_inputs = tf.keras.Input(shape=(3, 61440,), name='encoder_input')
x = layers.Conv1D(filters=32, kernel_size=3, activation='relu',
                  input_shape=(time_window_size, 1))(original_inputs)
# x=layers.MaxPooling1D()(x)
x = layers.Dense(intermediate_dim, activation='relu')(x)
z_mean = layers.Dense(latent_dim, name='z_mean')(x)
z_log_var = layers.Dense(latent_dim, name='z_log_var')(x)
z = Sampling()((z_mean, z_log_var))
encoder = tf.keras.Model(inputs=original_inputs, outputs=z, name='encoder')

# Define decoder model.
latent_inputs = tf.keras.Input(shape=(latent_dim,), name='z_sampling')
x = layers.Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = layers.Dense(original_dim, activation='sigmoid')(x)
decoder = tf.keras.Model(inputs=latent_inputs, outputs=outputs, name='decoder')

# Define VAE model.
outputs = decoder(z)
vae = tf.keras.Model(inputs=original_inputs, outputs=outputs, name='vae')

# Add KL divergence regularization loss.
kl_loss = - 0.5 * tf.reduce_mean(
    z_log_var - tf.square(z_mean) - tf.exp(z_log_var) + 1)
vae.add_loss(kl_loss)

# Train.
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-2)
vae.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError())

vae.fit(Xtrain, Xtrain, epochs=10, batch_size=64)
```

In [166]:

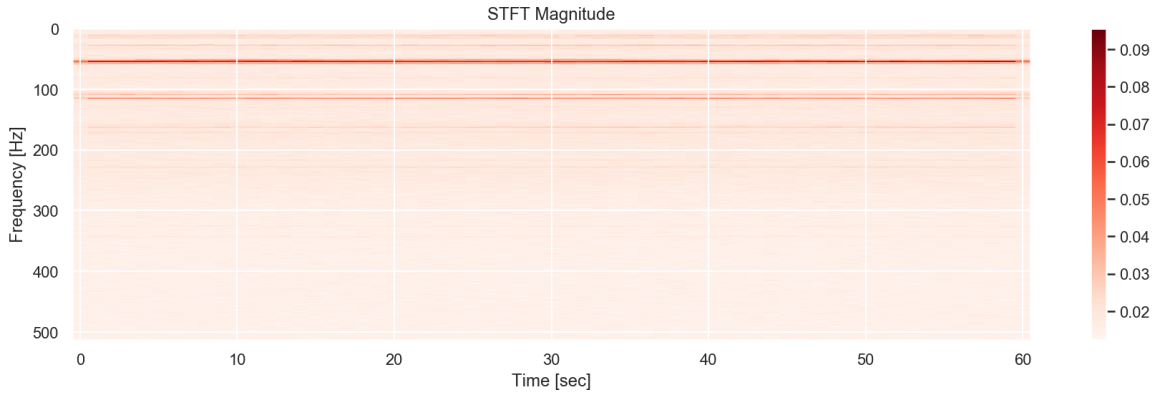
```
vae.summary()
```

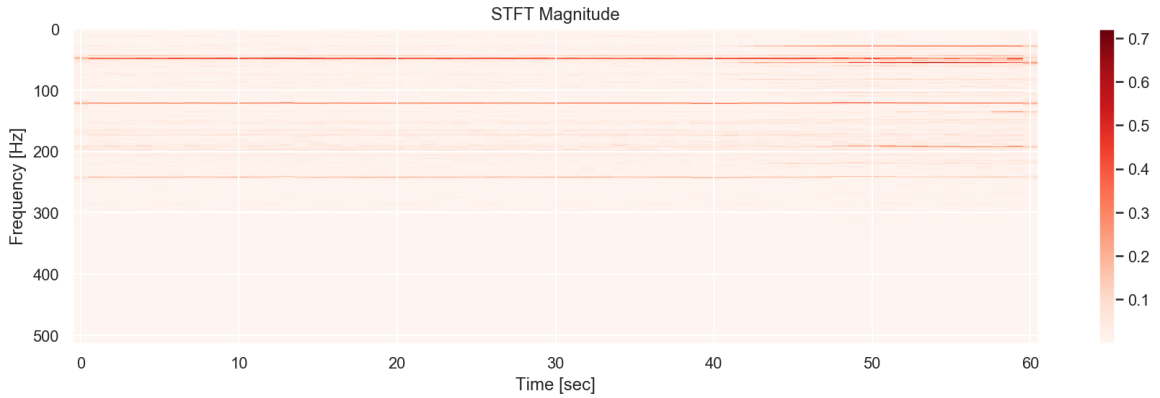
```
Model: "vae"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Con
nected to
==================================================================
==============================
encoder_input (InputLayer)   [(None, 3, 61440)]        0
_________________________________________________________________
_____________________________
conv1d_22 (Conv1D)           (None, 1, 32)             5898272   enc
oder_input[0][0]
_________________________________________________________________
_____________________________
dense_38 (Dense)             (None, 1, 64)             2112      con
v1d_22[0][0]
_________________________________________________________________
_____________________________
z_mean (Dense)               (None, 1, 32)             2080      den
se_38[0][0]
_________________________________________________________________
_____________________________
z_log_var (Dense)            (None, 1, 32)             2080      den
se_38[0][0]
_________________________________________________________________
_____________________________
sampling_11 (Sampling)       (None, None, 32)          0         z_m
ean[0][0]
                                                       z_l
og_var[0][0]
_________________________________________________________________
_____________________________
decoder (Model)              multiple                  3995712   sam
pling_11[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_Square_165 (TensorF [(None, 1, 32)]        0         z_m
ean[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_sub_506 (TensorFlow [(None, 1, 32)]        0         z_l
og_var[0][0]
                                                       tf_
op_layer_Square_165[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_Exp_11 (TensorFlowO [(None, 1, 32)]        0         z_l
og_var[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_sub_507 (TensorFlow [(None, 1, 32)]        0         tf_
op_layer_sub_506[0][0]
                                                       tf_
op_layer_Exp_11[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_add_484 (TensorFlow [(None, 1, 32)]        0         tf_
op_layer_sub_507[0][0]
_________________________________________________________________
_____________________________
tf_op_layer_Mean_26 (TensorFlow [()]                   0         tf_
```

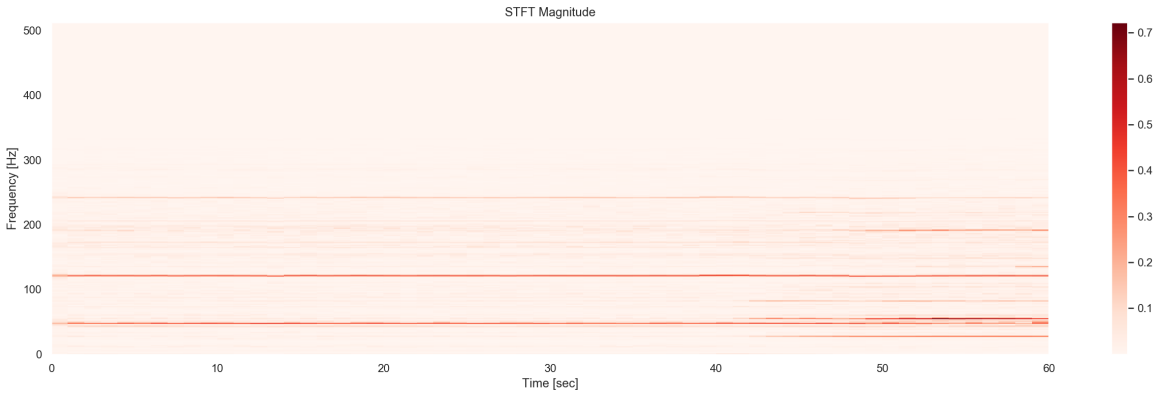
```
op_layer_add_484[0][0]
__________________________________________________________________________________
_____________________________
tf_op_layer_mul_792 (TensorFlow [()]                0           tf_
op_layer_Mean_26[0][0]
__________________________________________________________________________________
_____________________________
add_loss_11 (AddLoss)           ()                  0           tf_
op_layer_mul_792[0][0]
==================================================================================
==============================
Total params: 9,900,256
Trainable params: 9,900,256
Non-trainable params: 0
__________________________________________________________________________________
_____________________________
```

In [1728]:

```
#Get the prediction for test data
k=7

test_data=STFT_matrix_3D_array_1sample(xtest[k])
test_data_NN = test_data.reshape(1, 31293).astype('float32')
outputs=vae(test_data_NN)

STFT_predction=tf.reshape(outputs, [513, 61])

def plot_STFT_imshow(data):
    plt.figure(figsize=(15,4))
    plt.imshow(data, cmap='Reds',aspect='auto')
    plt.title('STFT Magnitude')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.colorbar()
    plt.show()

plot_STFT_imshow(STFT_predction)
plot_STFT_imshow(test_data[0])
plot_STFT(xtest[k])
```


STFT Magnitude
Frequency [Hz]
Time [sec]
0
100
200
300
400
500
0
10
20
30
40
50
60
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02


STFT Magnitude
Frequency [Hz]
Time [sec]
0
100
200
300
400
500
0
10
20
30
40
50
60
0.7
0.6
0.5
0.4
0.3
0.2
0.1


STFT Magnitude
Frequency [Hz]
Time [sec]
0
100
200
300
400
500
0
10
20
30
40
50
60
0.7
0.6
0.5
0.4
0.3
0.2
0.1

In [1742]:

```
# Compare the difference
def compare_diff(data):

    test_data = STFT_matrix_3D_array_1sample(data)
    test_data_NN = test_data.reshape(1, 31293).astype('float32')
    outputs = vae(test_data_NN)
    STFT_predction = tf.reshape(outputs, [513, 61])
    diff = np.sum(STFT_predction-test_data)
    return diff

compare_diff(xtrain[3])
```

Out[1742]:

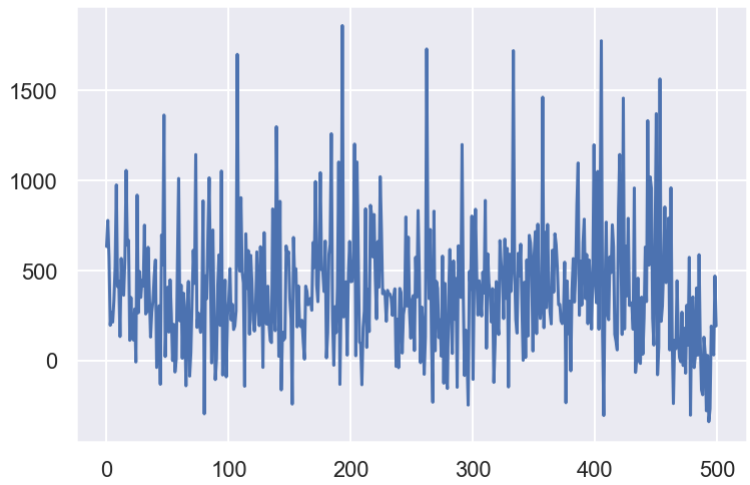
```
439.7153
```

In [1747]:

```
diff=[]
for i in range(500):
    diff.append(compare_diff(xtrain[i]))
plt.plot(diff)
```

Out[1747]:

```
[<matplotlib.lines.Line2D at 0x1e15252cf8>]
```



In [1746]:

```
pd.DataFrame(STFT_predction).describe()
```

Out[1746]:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **count** | 513 | 513 | 513 | 513 |
| **unique** | 511 | 513 | 512 | 512 |
| **top** | tf.Tensor(0.018061578, shape=(), dtype=float32) | tf.Tensor(0.015036911, shape=(), dtype=float32) | tf.Tensor(0.014560431, shape=(), dtype=float32) | tf.Tensor(0.017131567, shape=(), dtype=float32) |
| **freq** | 2 | 1 | 2 | 2 |

4 rows × 61 columns

In [ ]:

```
# 3D STFT Matrix and VAE
import keras
from tensorflow.keras import layers
from keras import backend as K

# prepare STFT data for VAE
n_samples = 800
data = STFT_matrix_3D_array(xtrain, n_samples)
x_train = data.reshape(n_samples, 31293).astype('float32')

# Train VAE
original_dim = 31293
intermediate_dim = 64
latent_dim = 32

class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon


# Define encoder model.
original_inputs = tf.keras.Input(shape=(original_dim,), name='encoder_input')
x = layers.Dense(intermediate_dim, activation='relu')(original_inputs)
z_mean = layers.Dense(latent_dim, name='z_mean')(x)
z_log_var = layers.Dense(latent_dim, name='z_log_var')(x)
z = Sampling()((z_mean, z_log_var))
encoder = tf.keras.Model(inputs=original_inputs, outputs=z, name='encoder')

# Define decoder model.
latent_inputs = tf.keras.Input(shape=(latent_dim,), name='z_sampling')
x = layers.Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = layers.Dense(original_dim, activation='sigmoid')(x)
decoder = tf.keras.Model(inputs=latent_inputs, outputs=outputs, name='decoder')

# Define VAE model.
outputs = decoder(z)
vae = tf.keras.Model(inputs=original_inputs, outputs=outputs, name='vae')

# Add KL divergence regularization loss.
kl_loss = - 0.5 * tf.reduce_mean(
    z_log_var - tf.square(z_mean) - tf.exp(z_log_var) + 1)
vae.add_loss(kl_loss)

# Train.
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
vae.compile(optimizer, loss=tf.keras.losses.MeanSquaredError())
vae.fit(x_train, x_train, epochs=40, batch_size=64)
vae.summary()

# Get the prediction of STFT for test data
k = 7
test_data = STFT_matrix_3D_array_1sample(xtest[k])
test_data_NN = test_data.reshape(1, 31293).astype('float32')
```

```
outputs = vae(test_data_NN)

STFT_predction = tf.reshape(outputs, [513, 61])

def plot_STFT_imshow(data):
    plt.figure(figsize=(15, 4))
    plt.imshow(data, cmap='Reds', aspect='auto')
    plt.title('STFT Magnitude')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.colorbar()
    plt.show()

plot_STFT_imshow(STFT_predction)
plot_STFT_imshow(test_data[0])
plot_STFT(xtest[k])

# Compare the difference
def compare_diff(data):

    test_data = STFT_matrix_3D_array_1sample(data)
    test_data_NN = test_data.reshape(1, 31293).astype('float32')
    outputs = vae(test_data_NN)
    STFT_predction = tf.reshape(outputs, [513, 61])
    diff = np.sum(STFT_predction-test_data)
    return diff

compare_diff(xtrain[3])

diff = []
for i in range(500):
    diff.append(compare_diff(xtrain[i]))
plt.plot(diff)

pd.DataFrame(STFT_predction).describe()
```

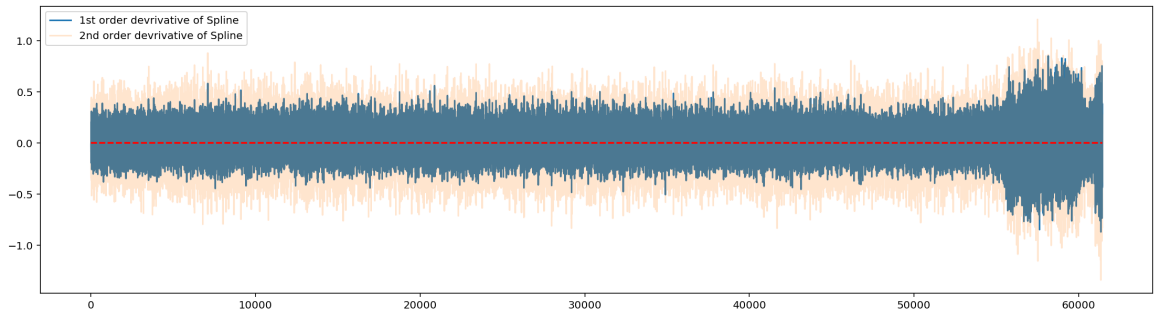
## LOF - 1st order derivative and 2nd order derivative

- Features: STFT (Flatten STFT Matrix into 31293 features )
- Features dimenstion: 61440*3
- Data: train data
- Algorithm: PCA + LOF
- Parameter: Novelty
- Scoring:
- **AUC :**

In [181]:

```
plot_derive(xtrain[6])
```

```
//anaconda3/lib/python3.7/site-packages/IPython/core/pylabtools.py:1
28: UserWarning: Creating legend with loc="best" can be slow with la
rge amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




In [175]:

```
# spline interpolation
i = 1
tck = interpolate.splrep(range(61440), xtrain[i], s=0)

xnew = np.arange(0, 61440, 1/10)
ynew = interpolate.splev(xnew, tck, der=0)
yder1 = interpolate.splev(xnew, tck, der=1)
yder2 = interpolate.splev(xnew, tck, der=2)
```

# Data Augementation

I divided the each signal into small sequences and get a score on each small pieces, then make an aggregation of the scores to have a global score. However, this doesn't work the best.

In [1775]:

```
def cut_sequences(data, seq_len):
    sequences = np.zeros(((len(data))*int(61440/seq_len), seq_len))
    print(sequences.shape)
    if 61440 % seq_len != 0:
        return "seq_len not comapatible"
    for j in range(len(data)):
        for k in range(int(61440/seq_len)):
            sequences[int(j*(61440/seq_len))+k] = data[j,
                                                       k*seq_len:(k+1)*seq_len]
    return sequences

s = cut_sequences(xtrain[:6], 2048)
s.shape
```

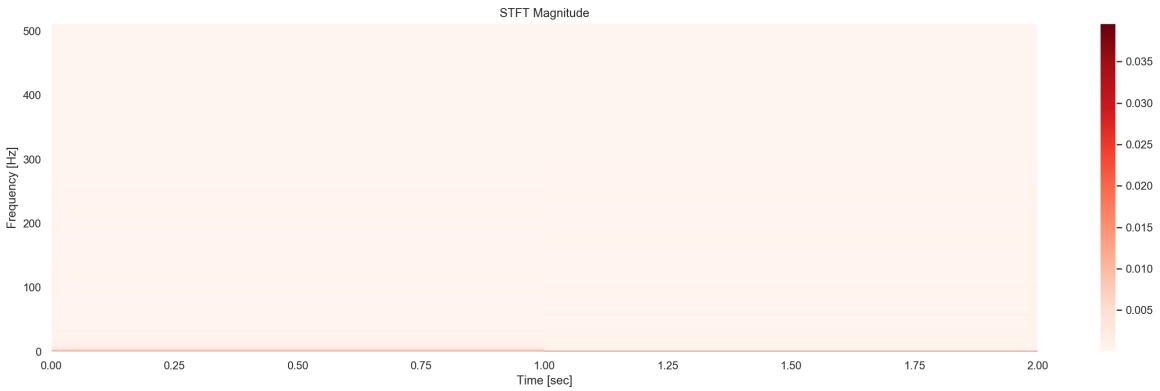
```
(180, 2048)
```

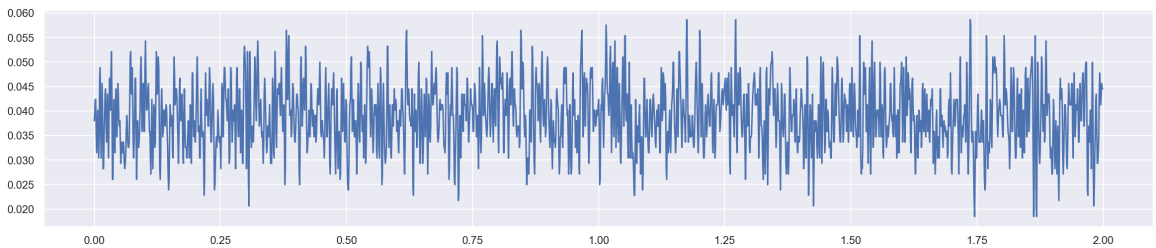
Out[1775]:

```
(180, 2048)
```

In [1788]:

```
plot_STFT(s[1])
plot_wave(np.arange(0.0, 2.0, 1/1024),s[1])
#plt.plot(s[1])
```



In [ ]:

```
# STFT matrix for neuro networks
def STFT_matrix_3D_array(data, n_samples):
    f, Zxx = cal_STFT(data[0])
    xtrain_STFT_3D = np.zeros(Zxx.shape)
    for i in range(n_samples):
        if i == 0:
            xtrain_STFT_3D = np.abs(Zxx)
        else:
            f, Zxx = cal_STFT(data[i])
            xtrain_STFT_3D = np.dstack([xtrain_STFT_3D, np.abs(Zxx)])
    xtrain_STFT_3D = np.moveaxis(xtrain_STFT_3D, -1, 0)
    return xtrain_STFT_3D
```

Treat the score of cutted sequences to calculated a score for the data

- take the highest abnormaly score as the score
- take the average score
- take the average of the 3 highest score

In [1797]:

```
seq_scores = sscore

def seq_scores(socre_list, seq_len):
    if 61440 % seq_len != 0:
        return "seq_len not comapatible"
    for j in range(int(len(seq_scores)/(61440/seq_len))):
        scores.append(np.max(score_list[int(j*(61440/seq_len)):int((j+1)*(61440/
seq_len))]))
        # print(j, k*seq_len,(k+1)*seq_len)
    return scores
```

In [95]:

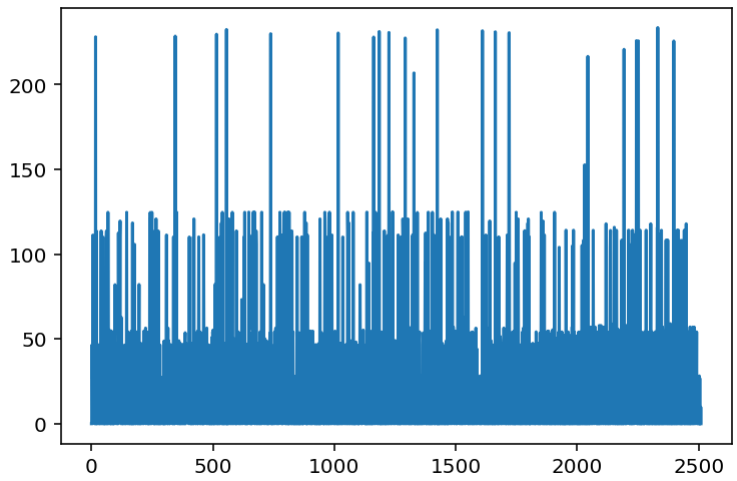
```
from scipy import fftpack
xtrain_FFT = np.abs(fftpack.fft(xtrain)[:30720])
xtest_FFT = np.abs(fftpack.fft(xtest)[:30720])
#freqs = fftpack.fftfreq(len(x)) * f_s
#plt.plot(xtrain_FFT)

#must use a one hot encoder
xtrain_FFT_argmax=np.argmax(xtrain_FFT,axis=1)/60
xtest_FFT_argmax=np.argmax(xtest_FFT,axis=1)/60
```

In [107]:

```
plt.plot(xtest_FFT_argmax)
```

Out[107]:

```
[<matplotlib.lines.Line2D at 0x1316f6fd0>]
```

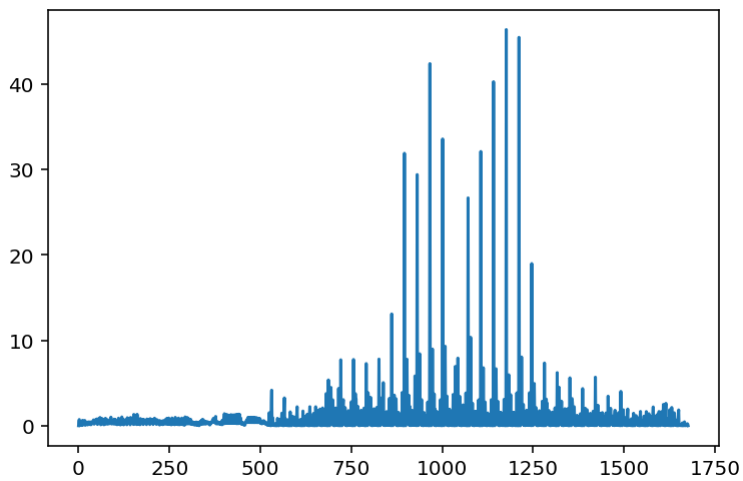


In [143]:

```
plt.plot(xtrain_var)

def derive(xtrain):
    xnew = np.arange(0, 61440, 1/1)
    for i in range(len(xtrain)):
        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i, :], s=0)

        ynew = interpolate.splev(xnew, tck, der=0)
        derive1 = interpolate.splev(xnew, tck, der=1)
        derive2 = interpolate.splev(xnew, tck, der=2)
        if i == 0:
            xtrain_interpolated = ynew
            derive1train = derive1
            derive2train = derive2
        else:
            #xtrain_interpolated = np.vstack((xtrain_interpolated, ynew))
            derive1train = np.vstack((derive1train, derive1))
            derive2train = np.vstack((derive2train, derive2))
    return derive1train, derive2train


derive1train, derive2train = derive(xtrain)
derive1test, derive2test = derive(xtest)
derive1test.shape
```

Out[143]:

```
(2511, 61440)
```




In [144]:

```
xtrain_der1_mean=np.mean(derive1train,axis=1)
xtrain_der2_mean=np.mean(derive2train,axis=1)
xtrain_der2_var=np.var(derive2train,axis=1)
xtrain_der1_var=np.var(derive1train,axis=1)

pca3 = PCA(n_components = 10, whiten = True)
pca4 = PCA(n_components = 10, whiten = True)
xtrain_der1_fpca = pca3.fit_transform(derive1train)
xtrain_der2_fpca = pca4.fit_transform(derive2train)
```

In [145]:

```
xtest_der1_mean=np.mean(derive1test,axis=1)
xtest_der2_mean=np.mean(derive2test,axis=1)
xtest_der1_var=np.var(derive1test,axis=1)
xtest_der2_var=np.var(derive2test,axis=1)
pca5 = PCA(n_components = 10, whiten = True)
pca6 = PCA(n_components = 10, whiten = True)
xtest_der1_fpca = pca5.fit_transform(derive1test)
xtest_der2_fpca = pca6.fit_transform(derive2test)
```

In [158]:

```
from sklearn import preprocessing
xtrain_scaled = preprocessing.scale([2000,-1000],with_mean=False,axis=0) # axis0 feature axis1 sample
xtrain_scaled
```

Out[158]:

```
array([ 1.33333333, -0.66666667])
```

In [160]:

```
#test standarsation

from sklearn import preprocessing
xtrain_scaled = preprocessing.scale(xtrain,with_mean=True,axis=0) # axis0 feature axis1 sample
xtrain_FFT_scaled=preprocessing.scale(xtrain_FFT,with_mean=True,axis=0)
```

In [161]:

```
# PCA transform
pca1 = PCA(n_components = 10, whiten = True)
pca1.fit(xtrain_scaled)
xtrain_fpca = pca1.fit_transform(xtrain_scaled)

pca2 = PCA(n_components = 10, whiten = True)
pca2.fit(xtrain_FFT_scaled)
xtrain_FFT_fpca = pca2.fit_transform(xtrain_FFT)
```

In [164]:

```
xtrain_new=np.vstack((xtrain_fpca.T, xtrain_mean.T,xtrain_var.T)).T
xtrain_new=np.vstack((xtrain_fpca.T, xtrain_FFT_fpca.T,xtrain_mean.T,xtrain_var.T,xtrain_der1_fpca.T)).T
xtrain_new=np.vstack((xtrain_fpca.T, xtrain_FFT_fpca.T,xtrain_mean.T,xtrain_var.T,xtrain_der1_fpca.T,xtrain_der2_fpca.T)).T
xtrain_new.shape
```

Out[164]:

```
(1677, 12)
```

In [165]:

```
# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = Tru
e)
lof1.fit(xtrain_new)
```

Out[165]:

```
LocalOutlierFactor(algorithm='auto', contamination='auto', leaf_size
=30,
                   metric='minkowski', metric_params=None, n_jobs=No
ne,
                   n_neighbors=5, novelty=True, p=2)
```

In [166]:

```
# Calculate anomaly score on the (PCA-transformed) test data
xtest_scaled = preprocessing.scale(xtest,with_mean=True,axis=0)
xtest_FFT_scaled = preprocessing.scale(xtest_FFT,with_mean=True,axis=0)
xtest_fpca = pca1.fit_transform(xtest_scaled)
xtest_FFT_fpca=pca2.fit_transform(xtest_FFT_scaled)

xtest_new=np.vstack((xtest_fpca.T, xtest_mean.T,xtest_var.T)).T
#xtest_new=np.vstack((xtest_fpca.T,xtest_FFT_fpca.T, xtest_mean.T,xtest_var.T,xt
est_der1_fpca.T)).T
#xtest_new=np.vstack((xtest_fpca.T,xtest_FFT_fpca.T, xtest_mean.T,xtest_var.T,xt
est_der1_fpca,xtest_der2_fpca)).T

sscore = -lof1.score_samples(xtest_new)
print(sscore[:100])
```

```
[2.96631004 7.51950888 1.02182625 4.47440661 5.63716981 5.3059072
 1.07680825 1.13299931 4.71543374 1.0326938  1.70644744 6.30516673
 5.04266581 8.08815426 5.00712392 5.5745228  6.18193673 5.10039765
 5.17026449 5.34470472 1.54604433 5.51260902 5.08592955 2.25369766
 4.08835422 5.04420483 5.15562568 6.7589202  1.80284525 7.07802628
 7.52252    6.09435151 5.89952323 5.14734845 3.94055281 1.20934195
 3.59096578 3.32497    1.52666236 1.09835138 1.67183069 6.36462038
 1.75164925 5.22988643 4.19394938 5.053928   1.36104129 1.62125934
 1.05832515 1.01999993 4.22597259 1.52917881 5.06026666 2.61179959
 1.21527974 5.63392013 5.06384721 2.47799754 5.28244428 5.82728728
 4.89680339 1.0270796  1.51983436 5.02586593 4.48139017 5.2538816
 1.70289421 6.21807991 1.02469768 1.0809021  5.31116301 6.15642125
 2.41158111 2.32998645 1.87021194 1.43960786 5.10786479 4.97614594
 1.74889255 5.975441   6.22927689 8.51856241 1.12067359 5.16273692
 5.27462886 5.58814008 5.16749932 6.22436792 6.75699154 3.93079298
 5.25013945 5.99018379 1.94308508 6.98232723 2.96852092 5.97909966
 1.07577733 0.94365866 5.07827752 1.18470254]
```

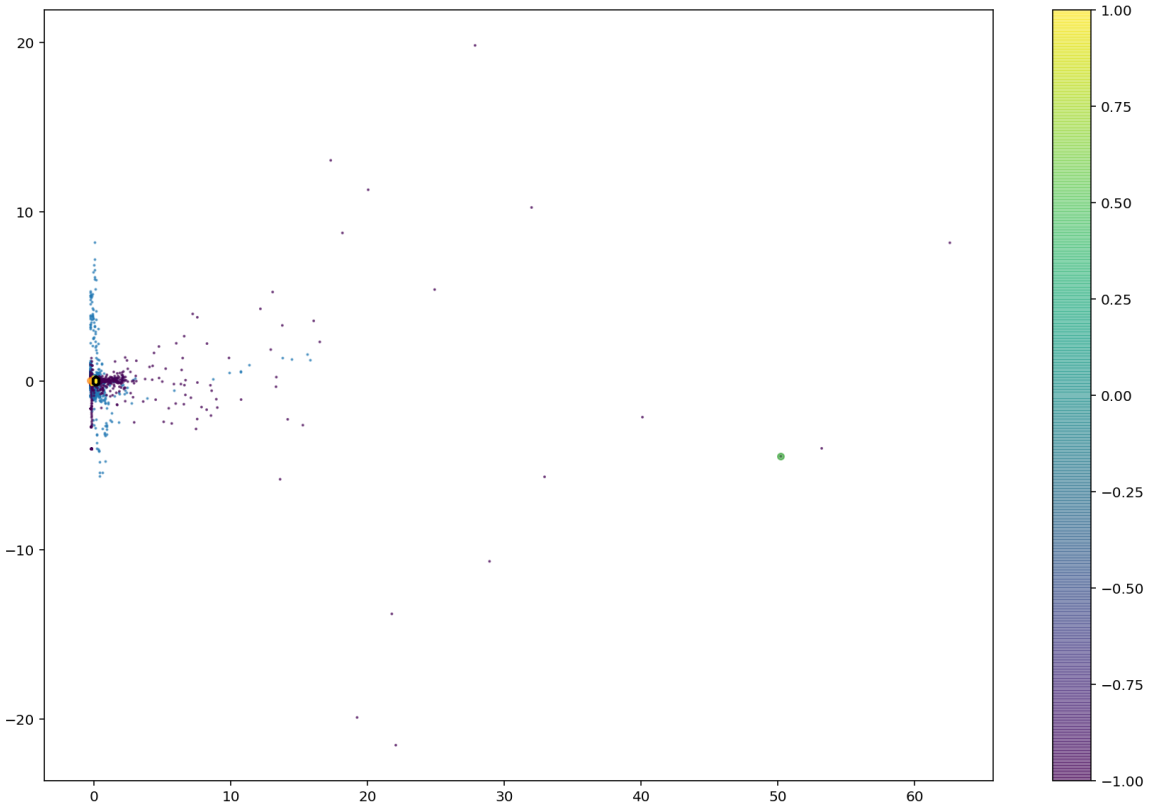
In [168]:

```
# Save the anomaly scores to file
print(sscore.shape)
np.savetxt('ytest_challenge_student01.csv', sscore, fmt = '%1.6f', delimiter=','
)
#0.672196163496 PCA + mean + var
#0.530522698901 PCA(raw+mean + var )
#0.46071106332 PCA(raw+mean + var+FFT )
#0.342011507039 sans whitening
#0.610795419432 PCA + + mean + var + argmaxFFT
#0.574823562856 PCA(raw) + PCA(FFT) + mean + var
```

```
(2511,)
```

In [458]:

```
from sklearn.svm import OneClassSVM
pca2d= PCA(n_components = 2, whiten = True)
xtrain_fpca2d=pca2d.fit_transform(xtrain_new)
X = xtrain_fpca2d
clf = OneClassSVM(nu=0.5,kernel="rbf", gamma=0.01).fit(X)
xtest_fpca2d=pca2d.transform(xtest_new)
c=clf.predict(xtest_fpca2d)
#############
xmin=np.min([xtest_fpca2d[:,0].min(),xtrain_fpca2d[:,0].min()])
xmax=np.max([xtest_fpca2d[:,0].max(),xtrain_fpca2d[:,0].max()])
ymin=np.min([xtest_fpca2d[:,1].min(),xtrain_fpca2d[:,1].min()])
ymax=np.max([xtest_fpca2d[:,1].max(),xtrain_fpca2d[:,1].max()])
xx, yy = np.meshgrid(np.linspace(xmin,xmax, 150),
                     np.linspace(ymin,ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(15, 10))
plt.scatter(xtrain_fpca2d[:, 0], xtrain_fpca2d[:, 1],alpha=0.6,s=1) #,c=MSE)
plt.scatter(xtest_fpca2d[:, 0], xtest_fpca2d[:, 1],alpha=0.6,s=1,c=c)
plt.colorbar()
plt.scatter(xtest_fpca2d[2019, 0], xtest_fpca2d[2019, 1],alpha=0.6,s=20)
plt.scatter(xtest_fpca2d[2433, 0], xtest_fpca2d[2433, 1],alpha=0.6,s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.show()
```




## Standartized data

0.551120422213

In [314]:

```
xtrain_standartize=np.zeros(xtrain.shape)
for i in range(len(xtrain)):
    xtrain_standartize[i]=xtrain[i]/xtrain[i].std()

xtest_standartize=np.zeros(xtest.shape)
for i in range(len(xtest)):
    xtest_standartize[i]=xtest[i]/xtest[i].std()
```

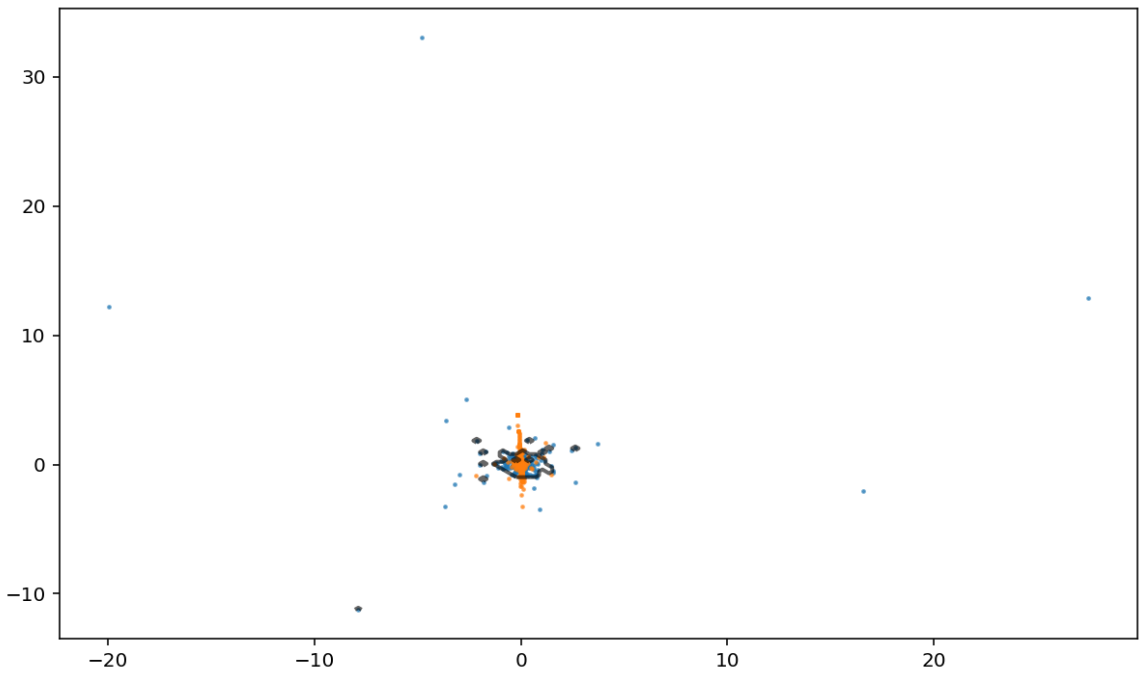
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: Run
timeWarning: invalid value encountered in true_divide
  import sys
```

In [460]:

```
####################
pca2d = PCA(n_components=2, whiten=True)
xtrain_fpca2d = pca2d.fit_transform(xtrain)
xtest_fpca2d = pca2d.transform(xtest)

# SVM
from sklearn.svm import OneClassSVM
clf = OneClassSVM(nu=0.005, kernel="rbf", gamma=4).fit(xtrain_fpca2d)
c=clf.predict(xtest_fpca2d)
```

In [461]:

```
# Compare given classifiers under given settings
xmin = np.min([xtest_fpca2d[:, 0].min(), xtrain_fpca2d[:, 0].min()])
xmax = np.max([xtest_fpca2d[:, 0].max(), xtrain_fpca2d[:, 0].max()])
ymin = np.min([xtest_fpca2d[:, 1].min(), xtrain_fpca2d[:, 1].min()])
ymax = np.max([xtest_fpca2d[:, 1].max(), xtrain_fpca2d[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)


plt.figure(figsize=(10, 6))
plt.scatter(xtrain_fpca2d[:, 0], xtrain_fpca2d[:, 1],  alpha=0.6, s=2)  # ,c=MS
E)

plt.scatter(xtest_fpca2d[:, 0], xtest_fpca2d[:, 1], alpha=0.6, s=2)
#plt.colorbar()

plt.contour(xx, yy, Z, levels=[0], alpha=0.6, linewidths=2, colors='black')

plt.show()
```




## construction of new features

In [ ]:

```
# interpolation matrix
xnew = np.arange(0,61440,1/10)
def derivative_matrix(xtrain):
    xnew = np.arange(0, 61440, 1)
    for i in range(len(xtrain)):

        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i], s=0)
        #ynew = interpolate.splev(xnew, tck, der=0)
        yder1 = interpolate.splev(xnew, tck, der=1)
        #yder2 = interpolate.splev(xnew, tck, der=2)
        if i % 200 == 0:
            print(i)
        if i == 0:
            interpolation = interpolate.splev(xnew, tck, der=0)
            derivative1=interpolate.splev(xnew, tck, der=1)
            derivative2=interpolate.splev(xnew, tck, der=2)
        else:
            #interpolation = np.vstack((interpolation, ynew))
            derivative1=np.vstack((derivative1,yder1))
            #derivative2=np.vstack((derivative2,yder2))
    return interpolation,derivative1, derivative2


#xtrain_interpolation, xtrain_derivative1, xtrain_derivative2 = derivative_matri
x(
 #   xtrain)
xtest_interpolation, xtest_derivative1, xtest_derivative2 = derivative_matrix(
    xtest)
```

In [321]:

```
xtrain_interpolated.shape
```

Out[321]:

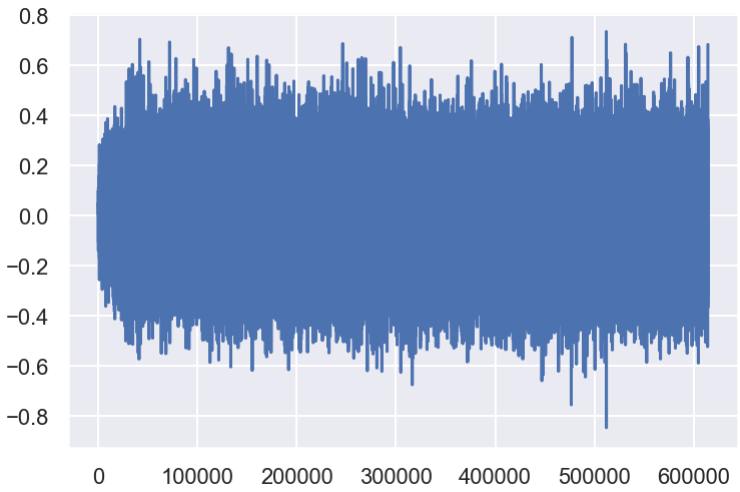
```
(2, 614400)
```

In [332]:

```
plt.plot(derive2train[1])
```

Out[332]:

```
[<matplotlib.lines.Line2D at 0x1a2b6f06d8>]
```




In [ ]:

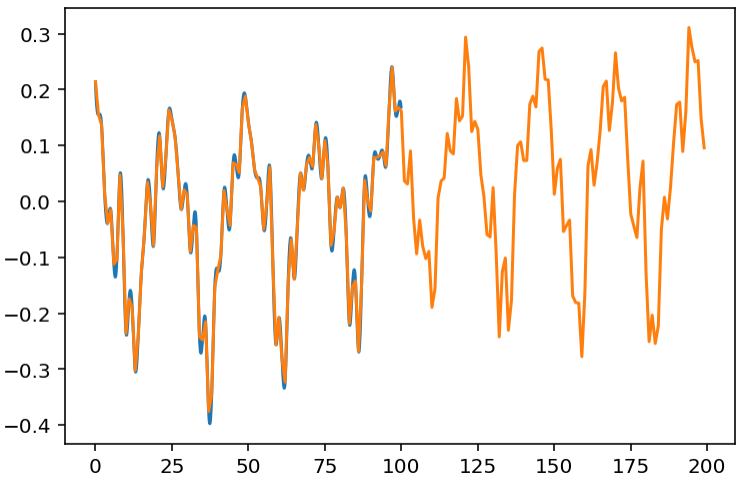
```
# basic statistic features
xmax = np.max(xtrain_interpolated,axis=1)
xmin = np.max(xtrain_interpolated,axis=1)
xmean = np.mean(xtrain_interpolated,axis=1)
xmaxabs=np.max(np.abs(xtrain_interpolated),axis=1)


# basic statistic features on 1st order derives
xmax = np.max(xtrain_interpolated,axis=1)
xmin = np.max(xtrain_interpolated,axis=1)
xmean = np.mean(xtrain_interpolated,axis=1)
xmaxabs=np.max(np.abs(xtrain_interpolated),axis=1)


# basic statistic features on 2nd order derive

division = x_train[:, 13:25]/(x_train[:, 0:12]+0.00000000000001)
xtrain_new = np.c_[xtrain_interpolated, substraction, division]
```

In [16]:

```
xtrain.shape
xnew = np.arange(0,61440,1/10)
tck = interpolate.splrep(range(61440), xtrain[2], s=0)
ynew = interpolate.splev(xnew, tck, der=0)
plt.plot(xnew[:1000],ynew[:1000])
plt.plot(xtrain[2][:200])
```

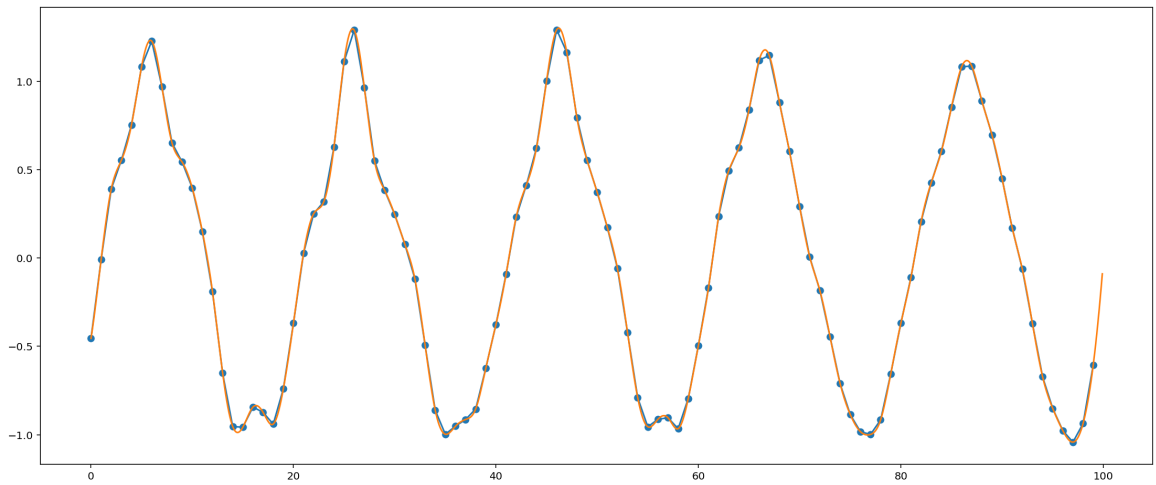
Out[16]:

```
[<matplotlib.lines.Line2D at 0x12116d710>]
```



In [21]:

```
i = 10
k = 1
start = 0
end = 100
length = end-start
ynew = []

plt.figure(figsize=(19, 8))
for n in range(k):
    # spline interpolation
    tck = interpolate.splrep(range(start, end), xtrain[i+n, start:end], s=0)
    #xnew = np.linspace(start,end,length*10)
    xnew = np.arange(start, end, 1/10)
    ynew = interpolate.splev(xnew, tck, der=0)

    # raw data
    plt.scatter(range(start, end), xtrain[i+n, start:end])
    # plt.scatter(xnew,ynew)
    plt.plot(range(start, end), xtrain[i+n, start:end])

    # spline
    plt.plot(xnew, ynew)
```



In [20]:

```
def plot_wave(t,x):
    dt = 1/1024
    t = np.arange(0.0, 60.0, dt)
    plt.figure(figsize=(20, 4))
    plt.plot(t,x)
    plt.show()

for i in range(20):
    k=10
    plot_wave(t,xtrain[k+i])
    plot_derive(xtrain[k+i])
    #plot_spectrum(xtrain[k+i])
    f,Zxx = plot_STFT(xtrain[k+i])
    #average frequence
    #plt.plot((f@np.abs(Zxx))/np.sum(np.abs(Zxx),axis=0))
```

```
---------------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-20-9b159017506c> in <module>
      8 for i in range(20):
      9     k=10
---> 10     plot_wave(t,xtrain[k+i])
     11     plot_derive(xtrain[k+i])
     12     #plot_spectrum(xtrain[k+i])

NameError: name 't' is not defined
```

In [921]:

```
def viz_PCA(x,labels):
    pca = PCA(n_components=2).fit(xtrain[:100])
    datapoint = pca.transform(xtrain[:100])
    plt.figure(figsize=(5,5))
    plt.scatter(datapoint[:, 0], datapoint[:, 1],c  = labels,cmap='Set1')
    plt.show()

def viz_PCA3D(x,labels):
    pca3D = PCA(n_components=3).fit(xtrain[:100])
    datapoint3D = pca3D.transform(xtrain[:100])
    fig = plt.figure(figsize=(5,5))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(datapoint3D[:, 0], datapoint3D[:, 1], datapoint3D[:, 2],c  = labe
ls,cmap='Set1')
    plt.show()
```

# Statistiques Feature Engineering Approche

## Baseline models with hyperparameter tuning

In [900]:

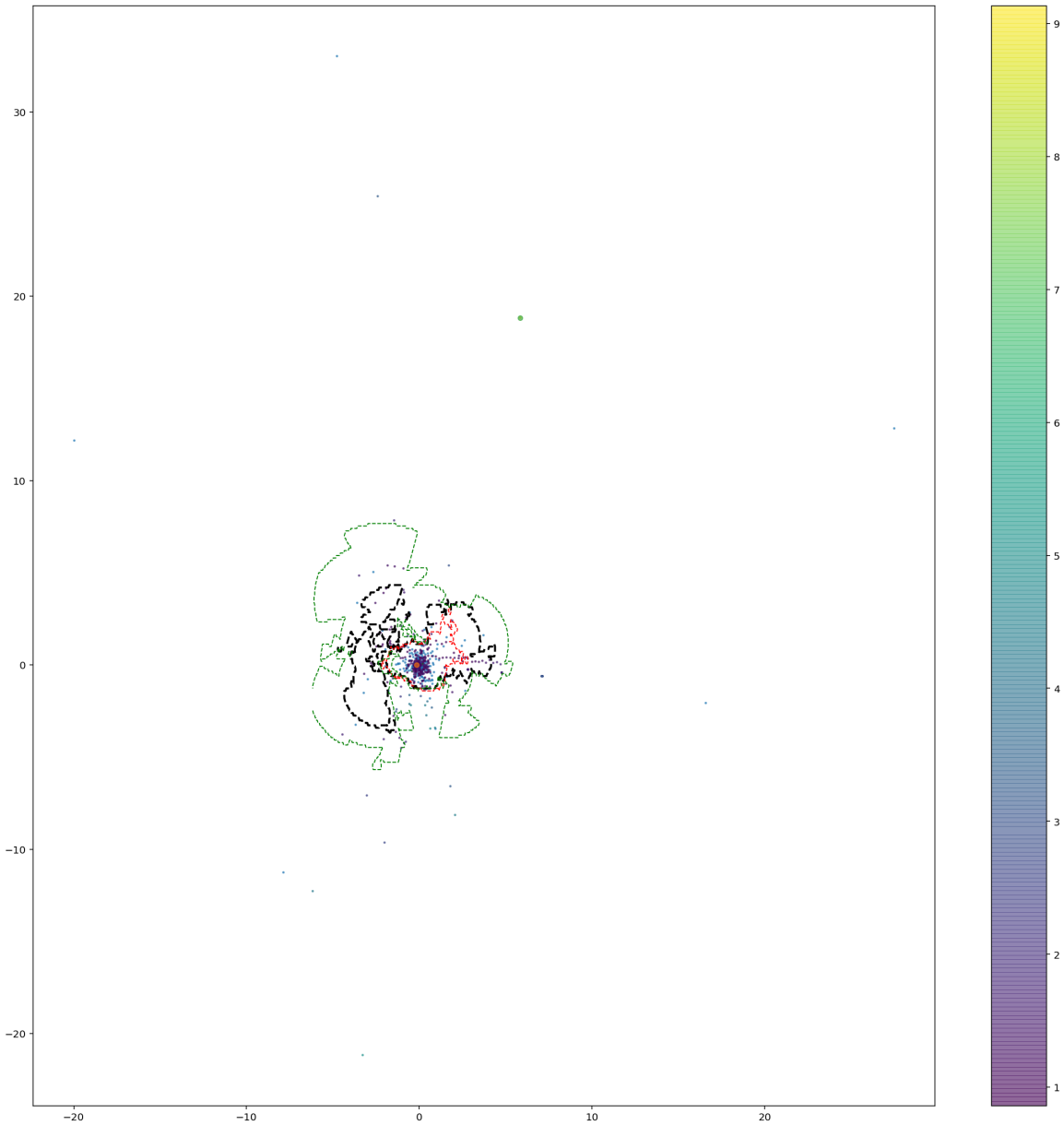
```
pca1 = PCA(n_components=2, whiten=True)
#pca1.fit(xtrain[:500, :])
xtrain_fpca = pca1.fit_transform(xtrain)

# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(xtrain_fpca)

# Calculate anomaly score on the (PCA-transformed) test data
xtest_fpca = pca1.fit_transform(xtest)
sscore = -lof1.score_samples(xtest_fpca)
print(np.argmax(sscore))
```

```
2007
```

In [6]:

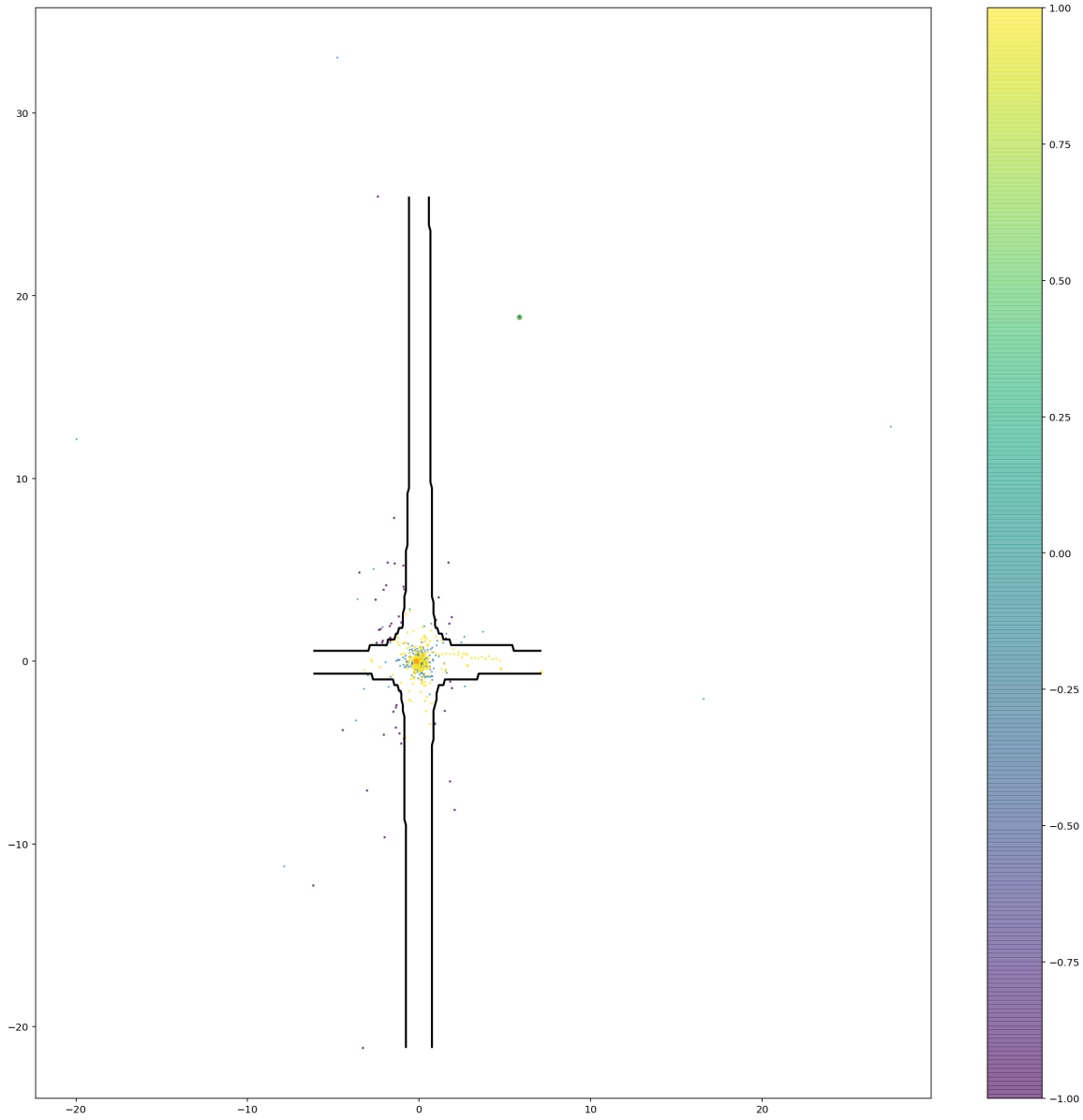
```
# Fit the low-dimensional method
lof = LocalOutlierFactor(n_neighbors=3, contamination=0.02, novelty=True)
lof.fit(xtrain_fpca)
sscore = -lof.score_samples(xtest_fpca)

xx, yy = np.meshgrid(np.linspace(xtest_fpca[:, 0].min(), xtest_fpca[:, 0].max(),
350),
                     np.linspace(xtest_fpca[:, 1].min(), xtest_fpca[:, 1].max(),
350))
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(20, 20))
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=2)
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=sscore)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
#plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')


lof = LocalOutlierFactor(n_neighbors=5, contamination=0.02, novelty=True)
lof.fit(xtrain_fpca)
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[2], linewidths=2, colors='black')
sscore = -lof.score_samples(xtest_fpca)

lof = LocalOutlierFactor(n_neighbors=10, contamination=0.02, novelty=True)
lof.fit(xtrain_fpca)
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[2], linewidths=1, colors='red')
sscore = -lof.score_samples(xtest_fpca)

lof = LocalOutlierFactor(n_neighbors=3, contamination=0.02, novelty=True)
lof.fit(xtrain_fpca)
Z = lof.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[2], linewidths=1, colors='green')
plt.show()
```

```
/Users/yang/anaconda3/lib/python3.7/site-packages/matplotlib/contou
r.py:1243: UserWarning: No contour levels were found within the data
range.
  warnings.warn("No contour levels were found"
/Users/yang/anaconda3/lib/python3.7/site-packages/matplotlib/contou
r.py:1243: UserWarning: No contour levels were found within the data
range.
  warnings.warn("No contour levels were found"
/Users/yang/anaconda3/lib/python3.7/site-packages/matplotlib/contou
r.py:1243: UserWarning: No contour levels were found within the data
range.
  warnings.warn("No contour levels were found"
```



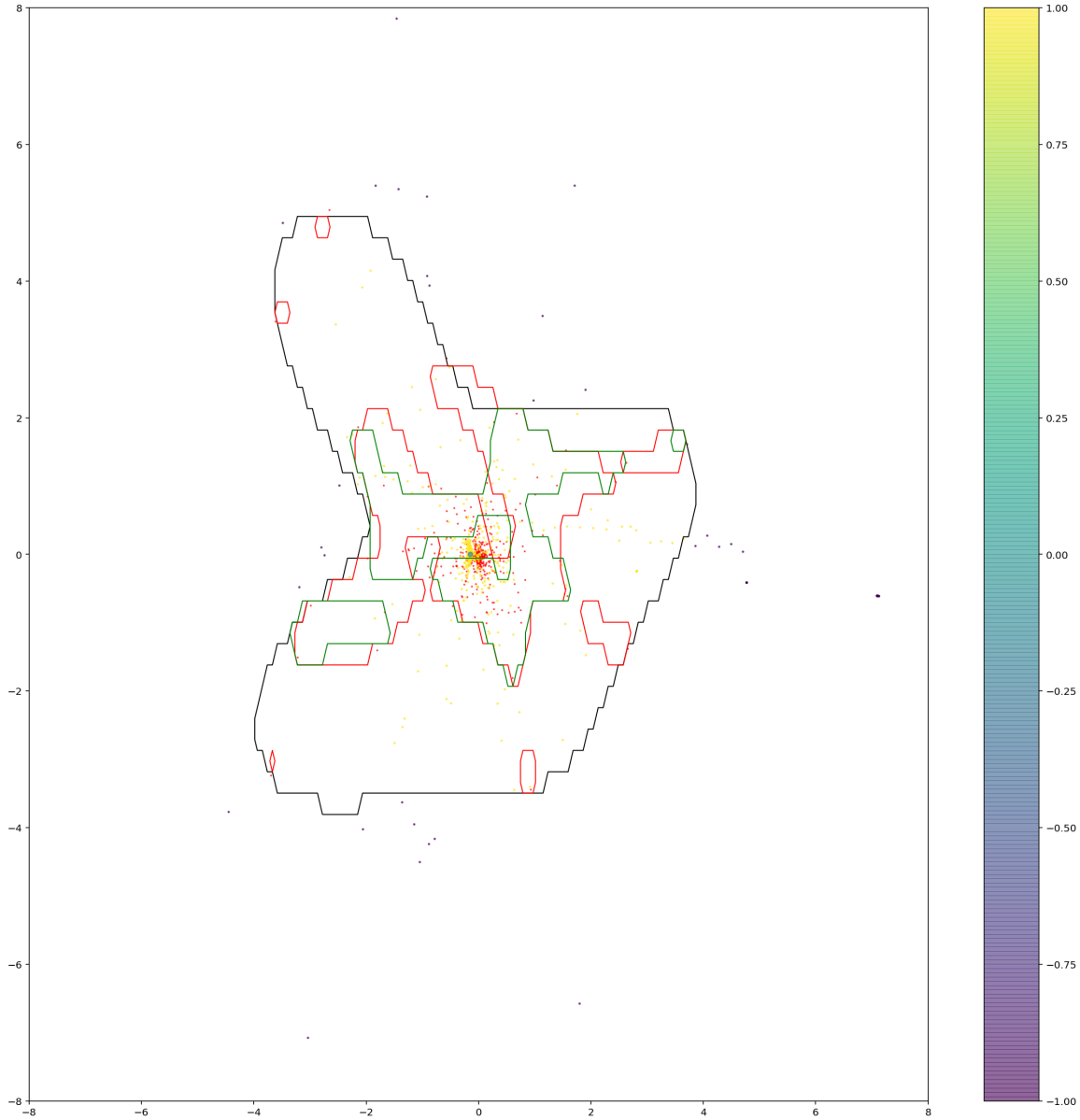
In [22]:

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=200, max_samples='auto', contamination=0.01,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0 #, warm_start=True
                     ).fit(xtrain_fpca)
clf.score_samples(xtest_fpca)
c=clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(xtest_fpca[:,0].min(),xtest_fpca[:,0].max(), 150),
                     np.linspace(xtest_fpca[:,1].min(),xtest_fpca[:,1].max(), 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(20, 20))
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1],alpha=0.6,s=1)
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1],alpha=0.6,s=2,c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1],alpha=0.6,s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1],alpha=0.6,s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

plt.show()
```


30
20
10
0
−10
−20
−20
−10
0
10
20
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [44]:

```
from sklearn.svm import OneClassSVM
clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.1).fit(xtrain_fpca)

c = clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(xtest_fpca[:, 0].min(), xtest_fpca[:, 0].max(),
150),
                     np.linspace(xtest_fpca[:, 1].min(), xtest_fpca[:, 1].max(),
150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(20, 20))
plt.xlim(-8, 8)
plt.ylim(-8, 8)
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=1, c='red')
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')

clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.8).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='red')

clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=1.5).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='green')
plt.show()
```




# New Features Construction

Features:high peaks, low peaks, average wave, peak length... **- Relative Amplitude approach: use moving average curve to define wave Peaks - Absolute Amplitude apporach: use 0 to define wave Peaks**

Statistiques: standard deviation, average, max, min , absolute max...

In [3]:

```
np.convolve(xtest[1794], np.ones((100,))/200, mode='same').mean()
xtest[1794,-1]
```

Out[3]:

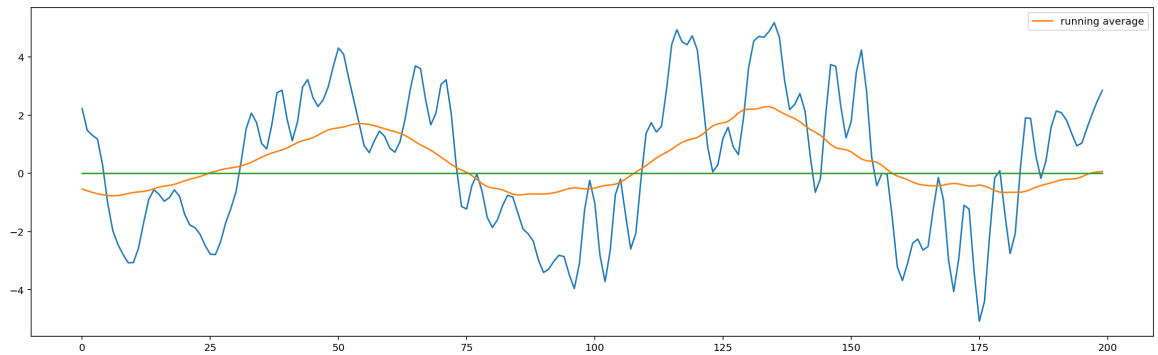
```
34.015848
```

In [20]:

```
# Moving Average example
plt.figure(figsize=(20,6))
i=9
plt.plot(xtest[i][:200])
plt.plot(np.convolve(xtest[i, :200], np.ones((50,)) /
                     50, mode='same'),label='running average')[:200]
plt.plot(range(200),np.zeros(200))
plt.legend()
```

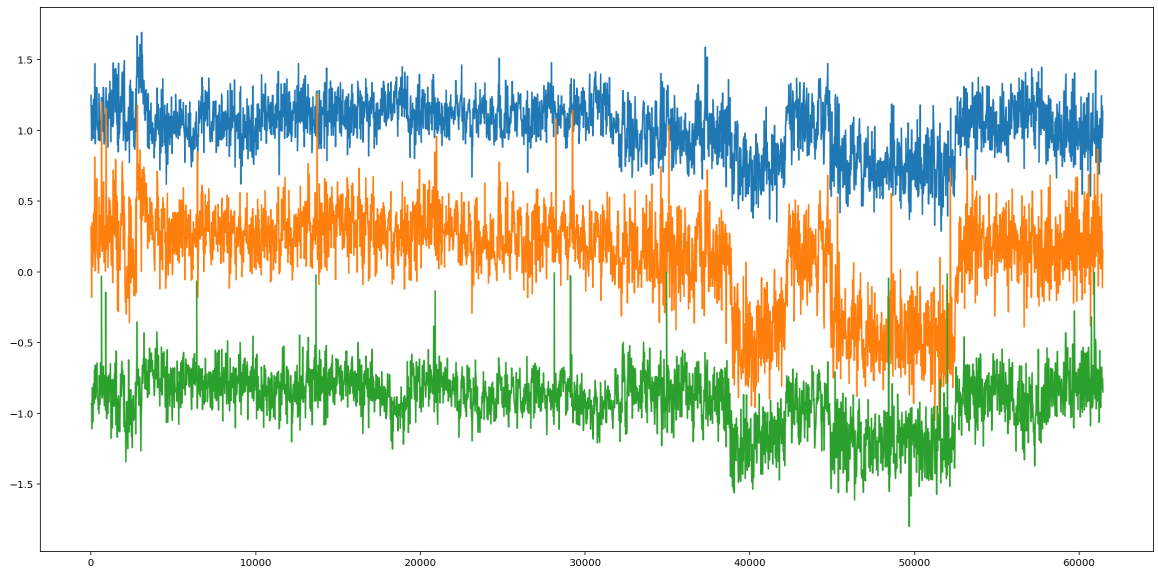
Out[20]:

```
<matplotlib.legend.Legend at 0x1c3ef677b8>
```




In [35]:

```
def plot_max_min_mean(x):
    point_max, n_max, point_min, n_min, point_mean, n_mean, interval_p, interval
_n = point_max_min_mean(
        x)
    plt.figure(figsize=(20, 10))
    plt.plot(n_max, point_max)
    plt.plot(n_max, point_mean)
    plt.plot(n_min, point_min)

plot_max_min_mean(xtrain[20])
```



In [24]:

```
def point_max_min_mean(x):
    point_max = []
    point_min = []
    point_mean = []
    n_max = []
    n_min = []
    interval_p = []
    interval_n = []

    if x[0] >= 0:
        nextsign = 1
    else:
        nextsign = -1

    n = 0
    while True:
        if n >= 61439:
            break

        if x[n] > 0:
            i = 0
            serie_relative_p = []
            serie_p = []
            while True:
                serie_p.append(x[n+i])
                serie_relative_p.append(x[n+i])
                if x[n+i] < 0 or (n+i) >= 61439:
                    n = n+i
                    break
                i = i+1
                if (n+i) >= 61439:
                    n = n+i
                    break
            point_max.append(np.max(serie_p))
            n_max.append(n+np.argmax(serie_relative_p))
            interval_p.append(len(serie_p))

        if x[n] < 0:
            i = 0
            serie_relative_n = []
            serie_n = []
            while True:
                serie_relative_n.append(x[n+i])
                serie_n.append(x[n+i])
                if x[n+i] > 0 or (n+i) >= 61439:
                    n = n+i
                    break
                i = i+1
                if (n+i) >= 61439:
                    n = n+i
                    break
                # print(n+i)
            point_min.append(np.min(serie_relative_n))
            n_min.append(n+np.argmin(serie_n))
            interval_n.append(len(serie_n))

        n = n+1

    if point_max == []:
```

```
        point_max = [0]
        n_max = 0
    if point_min == []:
        point_min = [0]
        n_min = 0
    if interval_p == []:
        interval_p = [0]
    if interval_n == []:
        interval_n = [0]

    if len(point_max) < len(point_min):
        n_mean = n_max
    else:
        n_mean = n_min
    for i in range(np.min((len(point_max), len(point_min)))):
        point_mean.append(point_max[i]+point_min[i])
    running_mean = np.convolve(x, np.ones((20,)) /
                               20, mode='valid')

    return point_max, n_max, point_min, n_min, point_mean, n_mean, interval_p, i
nterval_n

point_max, n_max, point_min, n_min, point_mean, n_mean, interval_p, interval_n =
point_max_min_mean(
    xtest[2])
```

In [816]:

```
def point_max_min_mean_relative(x):
    relative_peak_p = []
    relative_peak_n = []
    peak_length_p = []
    peak_length_n = []
    point_max = []
    point_min = []
    point_mean = []
    # The running mean is a case of the mathematical operation of convolution.
    moving_average = np.convolve(x, np.ones((500,)) /
                                 500, mode='same')
    if x[0] >= moving_average[0]:
        nextsign = 1
    else:
        nextsign = -1

    n = 0
    while True:
        if n >= 61439:
            break

        if x[n] > moving_average[n]:
            i = 0
            serie_relative_p = []
            serie_p = []
            while True:

                serie_relative_p.append(x[n+i]-moving_average[n+i])
                if x[n+i] < moving_average[n+i] or (n+i) >= 61439:
                    n = n+i
                    break
                i = i+1
                if (n+i) >= 61439:
                    n = n+i
                    break
            relative_peak_p.append(np.max(serie_relative_p))
            peak_length_p.append(len(serie_relative_p))
            point_max.append(np.max(serie_relative_p))

        if x[n] < moving_average[n]:
            i = 0
            serie_relative_n = []

            while True:
                serie_relative_n.append(moving_average[n+i]-x[n+i])
                if x[n+i] > moving_average[n+i] or (n+i) >= 61439:
                    n = n+i
                    break
                i = i+1
                if (n+i) >= 61439:
                    n = n+i
                    break
                # print(n+i)
            relative_peak_n.append(np.max(serie_relative_n))
            peak_length_n.append(len(serie_relative_n))
            point_min.append(np.min(serie_relative_n))

        n = n+1
        # print(n)
```

```
    if point_max == []:
        point_max = [0]

    if point_min == []:
        point_min = [0]


    if len(point_max) < len(point_min):
        n_mean = n_max
    else:
        n_mean = n_min
    for i in range(np.min((len(point_max), len(point_min)))):
        point_mean.append(point_max[i]+point_min[i])
    running_mean = np.convolve(x, np.ones((20,)) /
                                                20, mode='valid')
    return relative_peak_p, relative_peak_n, peak_length_p, peak_length_n, point
_max, n_max, point_min, n_min, point_mean, n_mean, interval_p, interval_n,runnin
g_mean

relative_peak_p, relative_peak_n, peak_length_p, peak_length_n, point_max, n_max
, point_min, n_min, point_mean, n_mean, interval_p, interval_n,running_mean = po
int_max_min_mean_relative(
    xtest[2])
len(relative_peak_p)
```

Out[816]:

```
5321
```

In [635]:

```
relative_peak_p, relative_peak_n, peak_length_p, peak_length_n, point_max, n_max
, point_min, n_min, point_mean, n_mean, interval_p, interval_n,running_mean = po
int_max_min_mean(
    xtrain[0])
```

In [652]:

```
def feature_matrix_a(x):
    fa1 = []
    fa2 = []
    fa3 = []
    fa4 = []
    fa5 = []
    fa6 = []

    for i in range(len(x)):
        #relative_peak_p, relative_peak_n, peak_length_p, peak_length_n, point_m
ax, n_max, point_min, n_min, point_mean, n_mean, interval_p, interval_n,running_
mean = point_max_min_mean(
         #   x[i])

        # feature added
        fa1.append(np.var(x[i]))
        fa2.append(np.mean(x[i]))
        fa3.append(np.max(x[i]))
        fa4.append(np.max(np.abs(x[i])))
        fa5.append(np.min(x[i]))
        fa6.append(np.std(x[i]))

    return  fa1, fa2, fa3, fa4, fa5, fa6

feature_matrix_a(-xtrain[4:5])
```

Out[652]:

```
([0.43035279756439065],
 [-0.03225759601236979],
 [1.213873],
 [1.402794],
 [-1.402794],
 [0.6560128028967046])
```

In [879]:

```
def feature_matrix_b(x):
    point_max = []
    n_max = []
    point_min = []
    n_min = []
    point_mean = []
    n_mean = []
    interval_p = []
    interval_n = []
    fb1 = []
    fb2 = []
    fb3 = []
    fb4 = []
    fb5 = []
    fb6 = []
    fb7 = []
    fb8 = []
    fb9 = []
    fb10 = []
    fb11 = []
    fb12 = []
    fb13 = []
    fb14 = []
    fb15 = []
    fb16 = []
    fb17 = []
    fb18 = []
    fb19 = []
    fb20 = []
    fb21 = []
    fb22 = []
    fb23 = []
    fb24 = []

    for i in range(len(x)):
        point_max, n_max, point_min, n_min, point_mean, n_mean, interval_p, inte
rval_n = point_max_min_mean(
            x[i])
        relative_peak_p, relative_peak_n, peak_length_p, peak_length_n, point_ma
x, n_max, point_min, n_min, point_mean, n_mean, interval_p, interval_n,running_m
ean = point_max_min_mean_relative(
            x[i])
        if relative_peak_p==[]:
            relative_peak_p=[0]
        if relative_peak_n==[]:
            relative_peak_n=[0]
        if peak_length_p==[]:
            peak_length_p=[0]
        if peak_length_n==[]:
            peak_length_n=[0]

        # feature added
        fb1.append(np.std(point_max))
        fb2.append(np.std(point_min))
        fb3.append(np.std(point_mean))
        if np.mean(interval_p)==0:
            fb4.append(0)
        else:
            fb4.append(np.max(interval_p)/np.mean(interval_p))
```

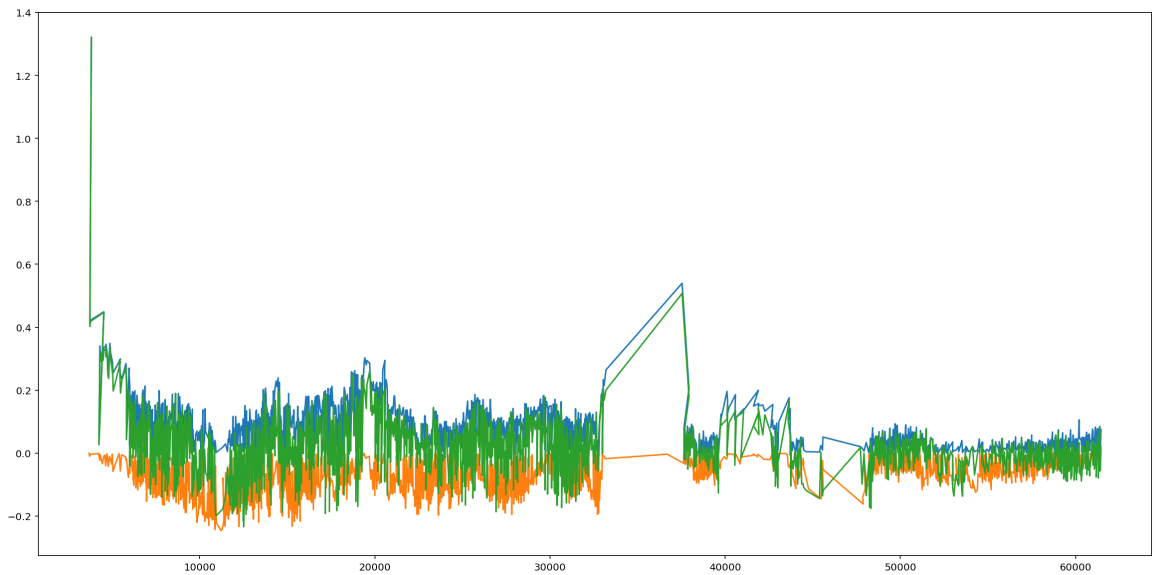
```
        if np.mean(interval_n)==0:
            fb5.append(0)
        else:
            fb5.append(np.max(interval_n)/np.mean(interval_n))
        fb6.append(np.mean(point_max))
        fb7.append(np.mean(point_min))
        fb8.append(np.max(np.abs(point_mean)))
        fb9.append(np.var(point_max))
        fb10.append(np.var(point_min))

        fb11.append(np.min(relative_peak_n))
        fb12.append(np.max(relative_peak_p))
        fb13.append(np.std(relative_peak_p))
        fb14.append(np.std(relative_peak_n))

        fb15.append(np.mean(relative_peak_p))
        fb16.append(np.mean(relative_peak_n))

        l1=np.max(peak_length_p)
        l2=np.max(peak_length_n)
        fb17.append(np.max((l1,l2)))
        #fb18.append(np.std(peak_length_p))
        #fb19.append(np.max(np.abs(running_mean)))
        fb20.append(np.std(running_mean))
        fb21.append(np.mean(running_mean))

        fb22.append(len(relative_peak_p))
        peak_length=[]
        peak_length=peak_length_p+peak_length_n
        lp=np.std(peak_length)
        if lp==0:
            fb23.append(0)
            fb24.append(0)
        else:
            fb23.append(lp)
            fb24.append(np.max((l1/lp,l2/lp)))

    return fb1, fb2, fb3, fb4, fb5, fb6, fb7, fb8, fb9, fb10, fb11, fb12, fb13,
fb14, fb15, fb16, fb17, fb18,fb19, fb20, fb21, fb22, fb23, fb24


fb1, fb2, fb3, fb4, fb5, fb6, fb7, fb8, fb9, fb10, fb11, fb12, fb13, fb14, fb15,
fb16, fb17, fb18,fb19, fb20, fb21, fb22, fb23, fb24=feature_matrix_b(xtest[2418:
2419])
```

In [248]:

```
def feature_matrix_c(x):
    fc1 = []
    fc2 = []
    fa3 = []

    for i in range(len(x)):
        xtrain_point_max, xtrain_n_max, xtrain_point_min, xtrain_n_min, xtrain_p
oint_mean, xtrain_n_mean, xtrain_interval_p, xtrain_interval_n = point_max_min_m
ean(
            x[i])

        # feature added
        fa1.append(np.var(x[i]))
        fa2.append(np.mean(x[i]))
        fa3.append(np.mean(x[i]))
        fa4.append(np.max(x[i]))
        fa5.append(np.min(x[i]))
        fa6.append(np.std(x[i]))

    return fa1, fa2, fa3, fa4, fa5, fa6


fa1, fa2, fa3, fa4, fa5, fa6 = feature_matrix_a(xtrain[:5])
```

In [240]:

```
plot_max_min_mean(xtest[0])
```



In [881]:

```
# construct new features
fa1, fa2, fa3, fa4, fa5, fa6 = feature_matrix_a(xtrain)
fb1, fb2, fb3, fb4, fb5, fb6, fb7, fb8, fb9, fb10, fb11, fb12 ,fb13,fb14,fb15,fb
16,fb17,fb18,fb19,fb20, fb21, fb22, fb23, fb24= feature_matrix_b(
    xtrain)
xtrain_matrix = np.vstack((np.array(fa1), np.array(fa2), np.array(fa3), np.array
(fa4), np.array(fa5), np.array(fa6), np.array(
    fb1), np.array(fb2), np.array(fb3), np.array(fb4), np.array(fb5), np.array(f
b6), np.array(fb7),  np.array(fb8), np.array(fb9), np.array(fb10),
    np.array(fb11), np.array(fb12), np.array(fb13), np.array(fb14), np.array(fb1
5), np.array(fb16), np.array(fb17), np.array(fb20), np.array(fb21), np.array(fb2
2), np.array(fb23), np.array(fb24))).T

xtrain_matrix.shape
```

In [884]:

```
fa1, fa2, fa3, fa4, fa5, fa6 = feature_matrix_a(xtest)
fb1, fb2, fb3, fb4, fb5, fb6, fb7, fb8, fb9, fb10, fb11, fb12, fb13, fb14, fb15,
fb16, fb17, fb18, fb19, fb20, fb21, fb22, fb23, fb24 = feature_matrix_b(
    xtest)

xtest_matrix = np.vstack((np.array(fa1), np.array(fa2), np.array(fa3), np.array(
fa4), np.array(fa5), np.array(fa6), np.array(
    fb1), np.array(fb2), np.array(fb3), np.array(fb4), np.array(fb5), np.array(f
b6), np.array(fb7),  np.array(fb8), np.array(fb9), np.array(fb10),
    np.array(fb11), np.array(fb12), np.array(fb13), np.array(fb14), np.array(fb1
5), np.array(fb16), np.array(fb17), np.array(fb20), np.array(fb21), np.array(fb2
2), np.array(fb23), np.array(fb24))).T

xtest_matrix.shape
```

Out[884]:

```
(2511, 28)
```

In [857]:

```
xtrain_matrix_standartize[:,6].std()
len(xtrain_matrix[:,1])
```

Out[857]:

```
1677
```

In [885]:

```
xtrain_matrix_standartize=np.zeros(xtrain_matrix.shape)
for i in range(len(xtrain_matrix[1])):
    #if xtrain_matrix_standartize[:,i].std()==0:
     #   xtrain_matrix_standartize[:,i]=xtrain_matrix[:,i]
    #else:
    xtrain_matrix_standartize[:,i]=xtrain_matrix[:,i]/xtrain_matrix[:,i].std()

xtest_matrix_standartize=np.zeros(xtest_matrix.shape)
for i in range(len(xtest_matrix[1])):
    if xtest_matrix[:,i].std()==0:
        xtest_matrix_standartize[:,i]=xtest_matrix[:,i]
    else:
        xtest_matrix_standartize[:,i]=xtest_matrix[:,i]/xtest_matrix[:,i].std()

xtrain_matrix_standartize.var(axis=0)
xtest_matrix_standartize.var(axis=0)
```

Out[885]:

```
array([1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0,
       1.00000000e+00, 1.06276000e+05, 1.93290643e-27, 1.00000000e+0
0,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+0
0])
```

In [862]:

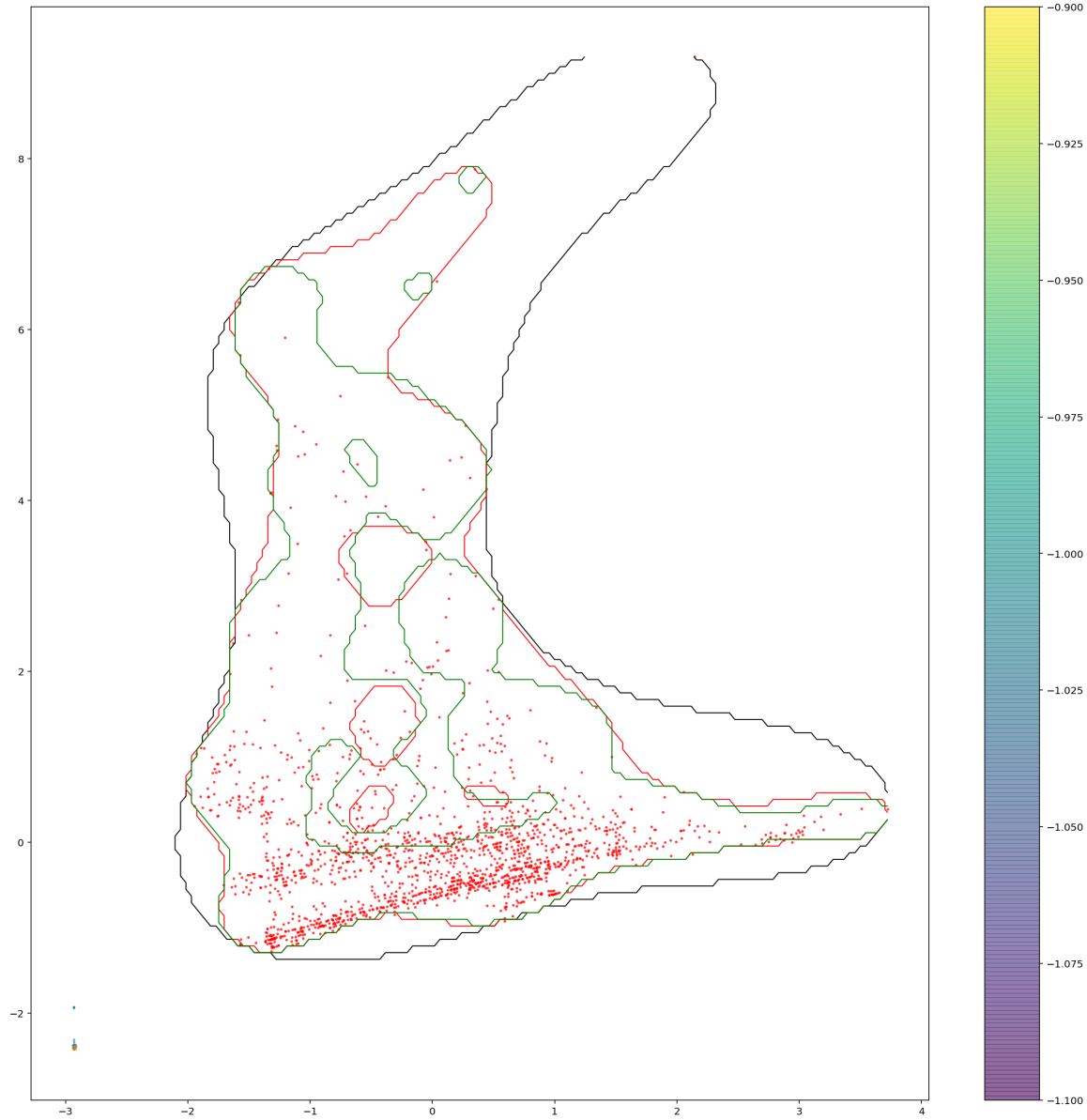
```
np.isnan(xtrain_matrix_standartize)
xtest_matrix_standartize[:,-10].var()
```

Out[862]:

```
0.9999999999999998
```

In [851]:

```
#PCA
pca1 = PCA(n_components=2, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_matrix_standartize[:,:-1])
xtest_fpca = pca1.transform(xtest_matrix_standartize[:,:-1])

from sklearn.svm import OneClassSVM
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_fpca)

c=clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xmin = np.min([xtest_fpca[:, 0].min(), xtrain_fpca[:, 0].min()])
xmax = np.max([xtest_fpca[:, 0].max(), xtrain_fpca[:, 0].max()])
ymin = np.min([xtest_fpca[:, 1].min(), xtrain_fpca[:, 1].min()])
ymax = np.max([xtest_fpca[:, 1].max(), xtrain_fpca[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(20, 20))
#plt.xlim(-8,8)
#plt.ylim(-8,8)
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1],alpha=0.6,s=3,c='red')
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1],alpha=0.6,s=2,c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1],alpha=0.6,s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1],alpha=0.6,s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')

clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.8).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='red')

clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=1.5).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='green')


plt.show()
```


8
6
4
2
0
−2
−3
−2
−1
0
1
2
3
4
−0.900
−0.925
−0.950
−0.975
−1.000
−1.025
−1.050
−1.075
−1.100

In [ ]:

```
# Save the anomaly scores to file
print(sscore.shape)
np.savetxt('ytest_challenge_student01.csv', sscore, fmt = '%1.6f', delimiter=','
)
```

In [838]:

```
# LOF PCA baseline

#pca1 = PCA(n_components = 10, whiten = True)
#xtrain_fpca = pca1.fit_transform(xtrain_matrix)
#xtest_fpca = pca1.transform(xtest_matrix)

lof1 = LocalOutlierFactor(n_neighbors = 5 ,contamination = 'auto', novelty = Tru
e)
lof1.fit(xtrain_matrix)

sscore = -lof1.score_samples(xtest_matrix)

print(sscore.shape)
np.savetxt('ytest_challenge_student01.csv', sscore, fmt = '%1.6f', delimiter=','
)
```

```
---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-838-d0da10577dcb> in <module>
      8 lof1.fit(xtrain_matrix)
      9
---> 10 sscore = -lof1.score_samples(xtest_matrix)
     11
     12 print(sscore.shape)

~/anaconda3/lib/python3.7/site-packages/sklearn/neighbors/lof.py in
_score_samples(self, X)
    463         check_is_fitted(self, ["offset_", "negative_outlier_
factor_",
    464                                "_distances_fit_X_"])
--> 465         X = check_array(X, accept_sparse='csr')
    466
    467         distances_X, neighbors_indices_X = (

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py
 in check_array(array, accept_sparse, accept_large_sparse, dtype, or
der, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_sample
s, ensure_min_features, warn_on_dtype, estimator)
    571         if force_all_finite:
    572             _assert_all_finite(array,
--> 573                                allow_nan=force_all_finite ==
'allow-nan')
    574
    575     shape_repr = _shape_repr(array.shape)

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py
 in _assert_all_finite(X, allow_nan)
     54                 not allow_nan and not np.isfinite(X).all()):
     55             type_err = 'infinity' if allow_nan else 'NaN, in
finity'
---> 56             raise ValueError(msg_err.format(type_err, X.dtyp
e))
     57
     58

ValueError: Input contains NaN, infinity or a value too large for dt
ype('float64').
```

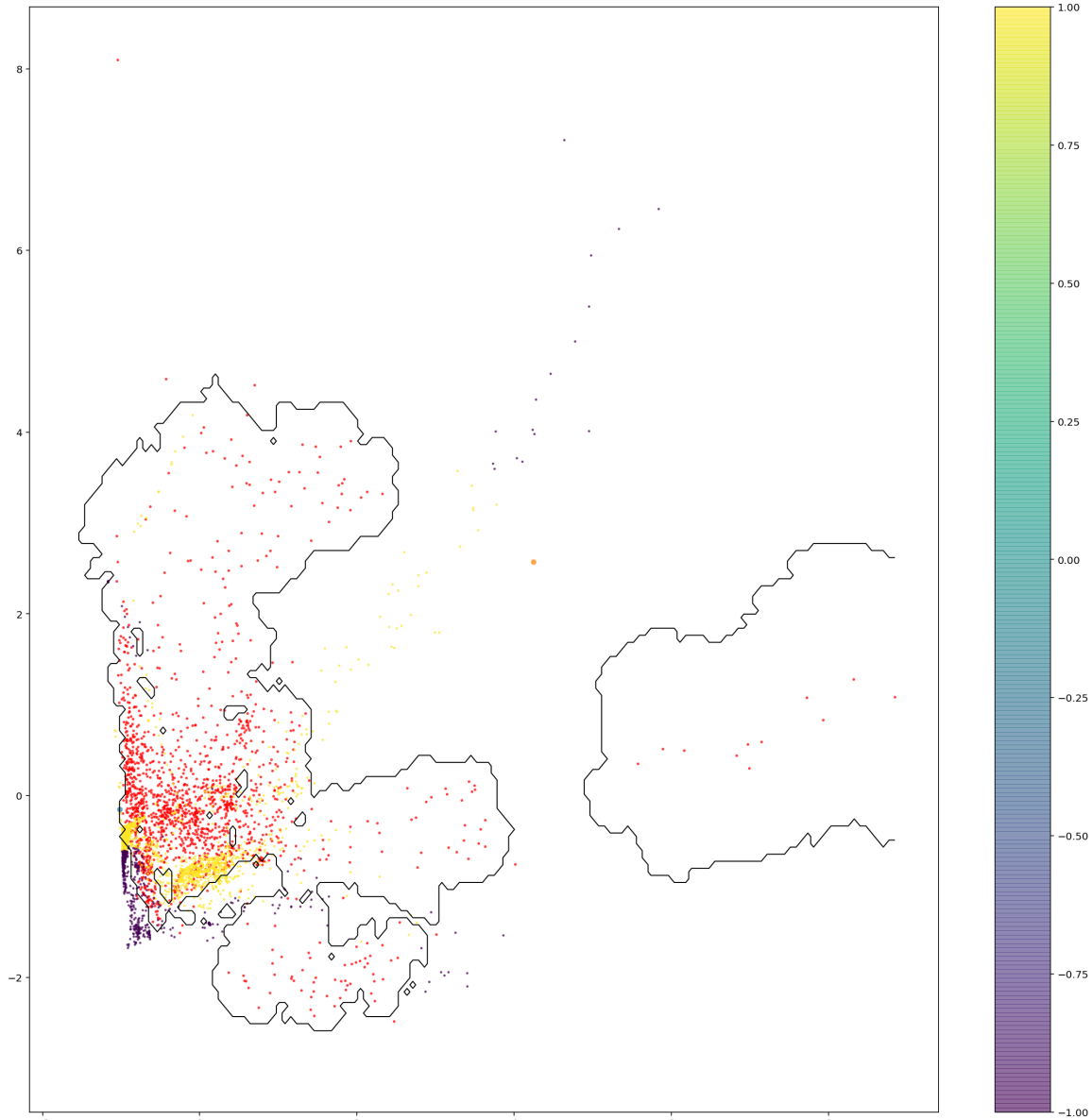
In [344]:

```
sscore[2019]=0.9
sscore[2433]=0.1
print(sscore.shape)
#0.564914037104
np.savetxt('ytest_challenge_student02.csv', sscore, fmt = '%1.6f', delimiter=','
)
```

```
(2511,)
```

In [777]:

```
# PCA
pca1 = PCA(n_components=2, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_matrix_standartize)
xtest_fpca = pca1.transform(xtest_matrix_standartize)

lof1 = LocalOutlierFactor(n_neighbors=4, contamination='auto', novelty=True)
lof1.fit(xtrain_fpca)

c = lof.predict(xtest_fpca)
# Compare given classifiers under given settings
xmin = np.min([xtest_fpca[:, 0].min(), xtrain_fpca[:, 0].min()])-1
xmax = np.max([xtest_fpca[:, 0].max(), xtrain_fpca[:, 0].max()])
ymin = np.min([xtest_fpca[:, 1].min(), xtrain_fpca[:, 1].min()])-1
ymax = np.max([xtest_fpca[:, 1].max(), xtrain_fpca[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = lof1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(20, 20))
# plt.xlim(-8,8)
# plt.ylim(-8,8)
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=3, c='red')
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')
plt.show()
```


8
6
4
2
0
−2
−2
0
2
4
6
8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [133]:

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=200, max_samples='auto', contamination=0.03,
                      max_features=1.0,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0  # , warm_start=True
                      )
clf.fit(xtrain_fpca)
clf.fit(xtest_fpca)
c = clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(xtest_fpca[:, 0].min(), xtest_fpca[:, 0].max(),
150),
                     np.linspace(xtest_fpca[:, 1].min(), xtest_fpca[:, 1].max(),
150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(20, 20))
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=10)
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.show()
```

```
---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-133-e0d546c0ff10> in <module>
     11 xx, yy = np.meshgrid(np.linspace(xtest_fpca[:,0].min(),xtest
_fpca[:,0].max(), 150),
     12                     np.linspace(xtest_fpca[:,1].min(),xtest
_fpca[:,1].max(), 150))
---> 13 Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
     14 Z = Z.reshape(xx.shape)
     15 plt.figure(figsize=(20, 20))

~/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/iforest.py
 in predict(self, X)
    310         is_inlier = np.ones(X.shape[0], dtype=int)
    311         threshold = self.threshold_ if self.behaviour == 'ol
d' else 0
--> 312         is_inlier[self.decision_function(X) < threshold] = -
1
    313         return is_inlier
    314

~/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/iforest.py
 in decision_function(self, X)
    342         # an outlier:
    343
--> 344         return self.score_samples(X) - self.offset_
    345
    346     def score_samples(self, X):

~/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/iforest.py
 in score_samples(self, X)
    377                                 "match the input. Model n_featu
res is {0} and "
    378                                 "input n_features is {1}."
--> 379                                 "".format(self.n_features_, X.s
hape[1]))
    380         n_samples = X.shape[0]
    381

ValueError: Number of features of the model must match the input. Mo
del n_features is 10 and input n_features is 2.
```

## Combine raw data + new features

New 29 features :

19: xtrain_matrix_standartize: feature_a feature_b

10: pca of raw data on 10 dimensions

In [792]:

```
pca1 = PCA(n_components=10, whiten=True)
xtrain_fpca10 = pca1.fit_transform(xtrain)
xtest_fpca10 = pca1.fit_transform(xtest)
```

In [886]:

```
xtrain_new = np.hstack((xtrain_matrix_standartize[:,:-1],xtrain_fpca10))
xtest_new = np.hstack((xtest_matrix_standartize[:,:-1],xtest_fpca10))
xtest_new.shape
```

Out[886]:

```
(2511, 37)
```

In [890]:

```
# 0.664386270656 xtrain_new -- PCA
# 0.691679953527 xtrain_new(29 feautres)
# 0.753602902097 xtrain_new(34 feautres)
# 0.687094441766 new 40 features
# 0.617238001656  31

# Fit the low-dimensional method
lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(xtrain_new)

# Calculate anomaly score on the (PCA-transformed) test data
sscore = -lof1.score_samples(xtest_new)
print(sscore[:20])
print(sscore.shape)
np.savetxt('ytest_challenge_studentlof01.csv',
           sscore, fmt='%1.6f', delimiter=',')
```

```
[6.07252432e+15 6.07252432e+15 6.07252432e+15 6.07252432e+15
 6.07252432e+15 6.07252432e+15 6.07252432e+15 6.07252432e+15
 6.07252432e+15 6.07252432e+15 6.07252432e+15 6.07252432e+15
 6.07252432e+15 6.07252432e+15 6.07252432e+15 6.07252432e+15
 6.07252432e+15 6.07252432e+15 6.07252432e+15 6.07252432e+15]
(2511,)
```

In [889]:

```
#0.770208387408 xtrain_new(29 feautres)
#0.756658879949 (34 feautres)
#0.683106838716
#0.690119519943 new feautres
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_new)
sscore1 = -clf.score_samples(xtest_new)
print(sscore1[:20])
np.savetxt('ytest_challenge_student01.csv', sscore1, fmt = '%1.6f', delimiter=
',')
```

```
[-0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0. -0.
-0.
 -0. -0.]
```

In [755]:

```
sscore = sscore/sscore.std()
print(sscore[:20])
sscore1 = sscore1/sscore1.std()
print(sscore1[:20])
```

```
[2.20590897 1.95340632 1.7587603  2.51017645 2.25007037 2.08739446
 1.91194587 1.93651295 2.52997391 1.4946492  2.71109208 2.41227834
 2.35116062 1.96438381 2.06853526 1.98841955 1.96201589 2.22558532
 2.41392858 1.97994159]
[-3.01256163 -3.09561494 -4.36564777 -2.67118193 -3.26724146 -2.9532
4284
 -4.11111098 -4.16322051 -2.71583841 -2.70387845 -4.16618136 -2.7225
2645
 -2.73981116 -3.13923    -2.93541497 -3.07185823 -3.07673684 -2.8634
244
 -2.76951371 -3.05936789]
```

In [756]:

```
# 0.780337300852
# 0.785834353023
# 0.695279765657
# stacking
sscore2 = sscore+sscore1
print(sscore2.var())
print(sscore2[:20])
np.savetxt('ytest_challenge_student01.csv',
           sscore2, fmt='%1.6f', delimiter=',')
```

```
3.2072709226324285
[-0.80665266 -1.14220862 -2.60688747 -0.16100548 -1.01717109 -0.8658
4838
 -2.1991651  -2.22670756 -0.1858645  -1.20922925 -1.45508928 -0.3102
4811
 -0.38865054 -1.17484619 -0.86687971 -1.08343868 -1.11472095 -0.6378
3908
 -0.35558513 -1.07942629]
```

In [802]:

```
# 0.768677308515 gamma=0.09 xtrain_new(29 feautres)
# 0.770208387408 gamma=0.1 xtrain_new(29 feautres)
# 0.775855303002 gamma=0.18 xtrain_new(29 feautres)
# 0.792928918388 gamma=0.2 xtrain_new(29 feautres)
# 0.785746288949 gamma=0.22 xtrain_new(29 feautres)
# 0.792882568875 gamma=0.25 xtrain_new(29 feautres)
# 0.783968012657 gamma=0.3 xtrain_new(29 feautres)
# 0.772294115466 gamma=0.4 xtrain_new(29 feautres)
# 0.774543611801 gamma=0.2 xtrain_new(34 feautres)
# 0.756658879949 gamma=0.1 xtrain_new(34 feautres)
# 0.778957630366 gamma=0.25 xtrain_new(34 feautres)
# 0.604087100004 gamma=0.25 xtrain_new(40 feautres)
# 0.647896659127 gamma=0.25
# 0.682175213517 gamma=0.2
# 0.564299133573 only 21 new features
# 0.671857812056 gamma=0.2
# 0.683100658781 gamma=0.1
clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.1).fit(xtrain_new)
sscoreSVM34 = -clf.score_samples(xtest_new)
print(sscore1[:20])
np.savetxt('ytest_challenge_studentOCSVM34.csv',
           sscoreSVM34, fmt='%1.6f', delimiter=',')
```

```
[-3.01256163 -3.09561494 -4.36564777 -2.67118193 -3.26724146 -2.9532
4284
 -4.11111098 -4.16322051 -2.71583841 -2.70387845 -4.16618136 -2.7225
2645
 -2.73981116 -3.13923    -2.93541497 -3.07185823 -3.07673684 -2.8634
244
 -2.76951371 -3.05936789]
```

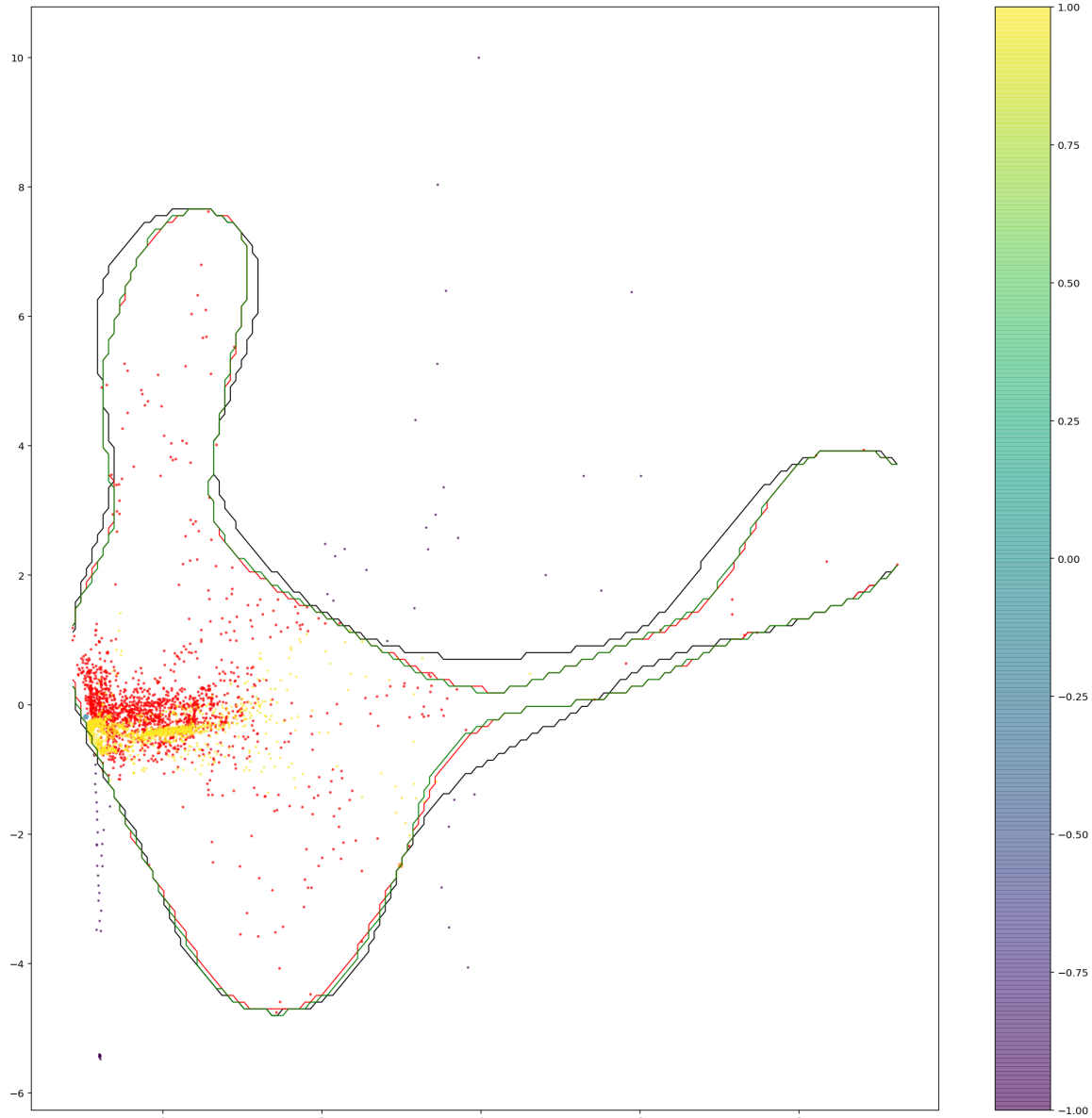
In [214]:

```
# stacking
# PCA transform
# 0.530290951339
# 0.523656791131 ????? bizzare
pca1 = PCA(n_components=10, whiten=True)
pca1.fit(xtrain)
xtrain_fpca = pca1.fit_transform(xtrain)
xtest_fpca = pca1.fit_transform(xtest)

lof1 = LocalOutlierFactor(n_neighbors=5, contamination='auto', novelty=True)
lof1.fit(xtrain_fpca)

sscore = -lof1.score_samples(xtest_fpca)

sscore2 = sscore+sscore1
print(sscore.var())
print(sscore[:20])
np.savetxt('ytest_challenge_student01.csv', sscore, fmt='%1.6f', delimiter=',')
# 2.3077829  3.20908069 2.32249655 5.24419322 3.7676995  2.48231917
```

```
1.3801805325933025
[2.31045244 3.24376794 2.32453097 5.23719547 3.80458585 2.48420202
 1.60378987 2.08478507 3.62894758 1.5123427  2.40609821 2.50594487
 2.53395641 3.7271263  2.54693773 1.73850519 3.74686362 2.85881084
 2.96734624 2.00687026]
```

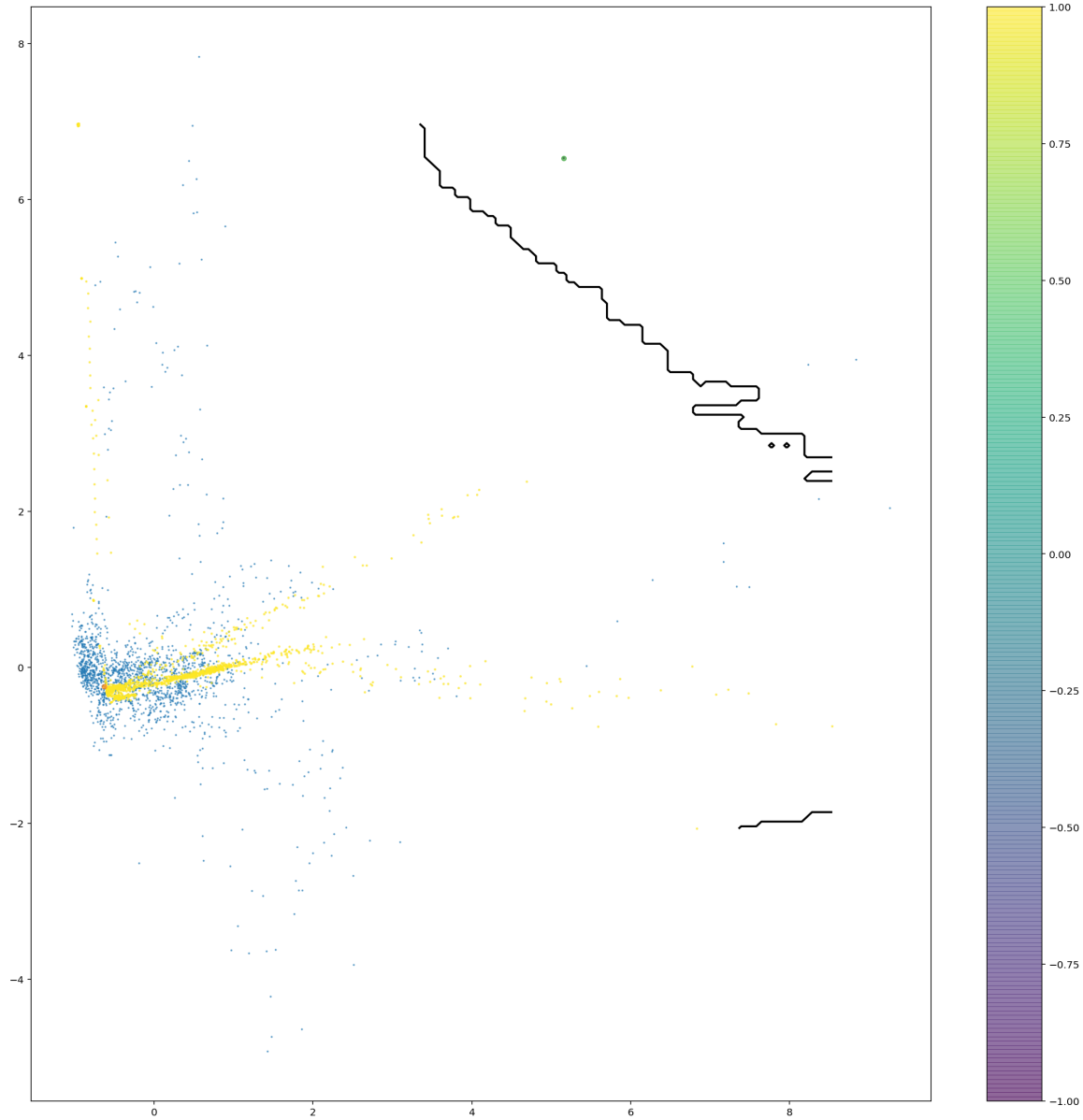
In [801]:

```
# PCA
from sklearn.svm import OneClassSVM
pca1 = PCA(n_components=2, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_new)
xtest_fpca = pca1.transform(xtest_new)

clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.1).fit(xtrain_fpca)
c = clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xmin = np.min([xtest_fpca[:, 0].min(), xtrain_fpca[:, 0].min()])
xmax = np.max([xtest_fpca[:, 0].max(), xtrain_fpca[:, 0].max()])
ymin = np.min([xtest_fpca[:, 1].min(), xtrain_fpca[:, 1].min()])
ymax = np.max([xtest_fpca[:, 1].max(), xtrain_fpca[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(20, 20))
# plt.xlim(-8,8)
# plt.ylim(-8,8)
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=3, c='red')
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')

clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.2).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='red')

clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.19).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='green')
plt.show()
```


10
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [772]:

```
from sklearn.ensemble import IsolationForest
pca1 = PCA(n_components=2, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_new)
xtest_fpca = pca1.fit_transform(xtest_new)

clf = IsolationForest(n_estimators=200, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0  # , warm_start=True
                      ).fit(xtrain_fpca)
clf.score_samples(xtest_fpca)
c = clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(xtest_fpca[:, 0].min(), xtest_fpca[:, 0].max(),
150),
                     np.linspace(xtest_fpca[:, 1].min(), xtest_fpca[:, 1].max(),
150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(20, 20))
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1], alpha=0.6, s=1)
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1], alpha=0.6, s=2, c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1], alpha=0.6, s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1], alpha=0.6, s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.show()
```


8
6
4
2
0
−2
−4
0
2
4
6
8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [773]:

```
# 0.754179181035  contamination='auto'
# 0.77046485471 contamination=0.001 ytest_challenge_studentIF
# 0.730471405441 contamination=0.001
# 0.75837072194
# 0.752929289184 new

from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0  # , warm_start=True
                      ).fit(xtrain_new)

sscore = -clf.score_samples(xtest_new)
print(sscore.var())
print(sscore[:20])
np.savetxt('ytest_challenge_studentIF2.csv',
           sscore, fmt='%1.6f', delimiter=',')
```

```
0.0016621665502416952
[0.46794729 0.46413916 0.44172638 0.4656757  0.46133657 0.45878322
 0.43326885 0.42991634 0.44967071 0.49115709 0.42799131 0.46589037
 0.47965303 0.47450299 0.48009644 0.46704321 0.47059914 0.45211557
 0.45353215 0.46562746]
```

In [229]:

```
# 0.75837072194
score1 = sscore/sscore.std()
print(score1[:20])
score2 = sscore1/sscore1.std()
print(score2[:20])
print(score1.var(), score2.var())
stack = score1+score2
np.savetxt('ytest_challenge_studentstacking1.csv',
           sscore, fmt='%1.6f', delimiter=',')
```

```
[12.07884623 10.83562943 10.09277201 11.18210646 10.84034786 10.6834
2236
  9.97302631  9.9371983  10.77125512 10.86964653  9.92276691 10.8649
259
 11.10707602 11.43993498 11.27006591 10.97791172 11.06669021 10.7121
2938
 10.56759247 10.86982538]
[-1.78011806 -2.56167143 -3.4529981  -2.29612489 -2.80500883 -2.6325
4979
 -3.63963812 -3.52979695 -2.40380335 -2.11234954 -3.5414618  -2.3936
7259
 -2.41687533 -1.84591588 -2.45415999 -2.71549783 -2.65308998 -2.1415
8536
 -2.41447697 -2.71707027]
1.0 1.0
```

# Feature construction with 1st/2nd order derivatives and FFT

(Doesn't work very well)

In [323]:

```
def derivative_matrix(xtrain):
    xnew = np.arange(0, 61440, 1)
    for i in range(len(xtrain)):

        # spline interpolation
        tck = interpolate.splrep(range(61440), xtrain[i], s=0)
        #ynew = interpolate.splev(xnew, tck, der=0)
        yder1 = interpolate.splev(xnew, tck, der=1)
        yder2 = interpolate.splev(xnew, tck, der=2)
        if i % 200 == 0:
            print(i)
        if i == 0:
            interpolation = interpolate.splev(xnew, tck, der=0)
            derivative1 = interpolate.splev(xnew, tck, der=1)
            derivative2 = interpolate.splev(xnew, tck, der=2)
        else:
            #interpolation = np.vstack((interpolation, ynew))
            derivative1 = np.vstack((derivative1, yder1))
            derivative2 = np.vstack((derivative2, yder2))
    return interpolation, derivative1, derivative2
```

In [324]:

```
xtrain_interpolation, xtrain_derivative1, xtrain_derivative2 = derivative_matrix
(
    xtrain)
xtest_interpolation, xtest_derivative1, xtest_derivative2 = derivative_matrix(
    xtest)
```

```
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
```

In [429]:

```
def feature_matrix_derivative12(x):
    fa1 = []
    fa2 = []
    fa3 = []
    fa4 = []
    fa5 = []
    fa6 = []
    fa7 = []
    fa8 = []
    fa9 = []
    fa10 = []

    for i in range(len(x)):
        # xtrain_point_max, xtrain_n_max, xtrain_point_min, xtrain_n_min, xtrain
_point_mean, xtrain_n_mean, xtrain_interval_p, xtrain_interval_n = point_max_min
_mean(
         #   x[i])
        # feature added
        fa1.append(np.max(x[i]))
        fa2.append(np.mean(x[i]))
        fa3.append(np.max(x[i]))
        fa4.append(np.max(np.abs(x[i])))
        fa5.append(np.min(x[i]))
        fa6.append(np.std(x[i]))
        if np.mean(x[i]) == 0:
            fa7.append(0)
            fa8.append(0)
        else:
            fa7.append(np.max(x[i])/np.mean(x[i]))
            fa8.append(np.min(x[i])/np.mean(x[i]))
        fa9.append(np.mean(x[i][x[i] >= 0.0]))
        fa10.append(np.mean(x[i][x[i] <= 0.0]))

    return fa1, fa2, fa3, fa4, fa5, fa6, fa7, fa8, fa9, fa10

feature_matrix_derivative12(xtest_derivative1[3:4])
```

Out[429]:

```
([0.09350316879665418],
 [-0.0013074236640784817],
 [-0.039206737120112196],
 [0.09350316879665418],
 [-0.07980692133453732],
 [0.016826481308956795],
 [-128568.05807933948],
 [109735.54190004335],
 [0.014334893790744251],
 [-0.012690375692312121])
```

In [455]:

```
fa1, fa2, fa3, fa4, fa5, fa6, fa7, fa8, fa9, fa10 = feature_matrix_derivative12(
    xtrain_derivative1)
fa1_, fa2_, fa3_, fa4_, fa5_, fa6_, fa7_, fa8_, fa9_, fa10_ = feature_matrix_der
ivative12(
    xtrain_derivative2)
xtrain_der_matrix = np.vstack((np.array(fa2), np.array(fa3), np.array(fa5), np.a
rray(fa6),np.array(fa2_), np.array(fa3_), np.array(fa5_), np.array(fa6_))).T
xtrain_der_matrix.shape
```

Out[455]:

```
(1677, 8)
```

In [456]:

```
fa1, fa2, fa3, fa4, fa5, fa6, fa7, fa8, fa9, fa10 = feature_matrix_derivative12(
    xtest_derivative1)
fa1_, fa2_, fa3_, fa4_, fa5_, fa6_, fa7_, fa8_, fa9_, fa10_ = feature_matrix_der
ivative12(
    xtest_derivative2)

xtest_der_matrix = np.vstack(( np.array(fa2), np.array(fa3),  np.array(fa5), np.
array(fa6),
                              np.array(fa2_), np.array(fa3_),  np.array(fa5_), n
p.array(fa6_))).T
xtest_der_matrix.shape
```

Out[456]:

```
(2511, 8)
```

In [458]:

```
xtrain_der_matrix_standartize = np.zeros(xtrain_der_matrix.shape)
for i in range(len(xtrain_der_matrix[1])):
    xtrain_der_matrix_standartize[:, i] = xtrain_der_matrix[:,
                                                            i]/xtrain_der_matrix
[:, i].std()

xtest_der_matrix_standartize = np.zeros(xtest_der_matrix.shape)
for i in range(len(xtest_der_matrix[1])):
    # if xtest_matrix[i].std()==0:
    #    xtest_matrix_standartize[i]=xtest_matrix[i]
    # else:
    xtest_der_matrix_standartize[:, i] = xtest_der_matrix[:,
                                                          i]/xtest_der_matrix[:,
i].std()
```

In [387]:

```
pca10 = PCA(n_components=10, whiten=True)
xtrain_fpca10 = pca10.fit_transform(xtrain)
xtest_fpca10 = pca10.fit_transform(xtest)
```

In [459]:

```
xtrain_der_new = np.hstack(
    (xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_standartize))
xtest_der_new = np.hstack(
    (xtest_matrix_standartize, xtest_fpca10, xtest_der_matrix_standartize))
xtest_der_new.shape
```

Out[459]:

```
(2511, 42)
```

## Test new features

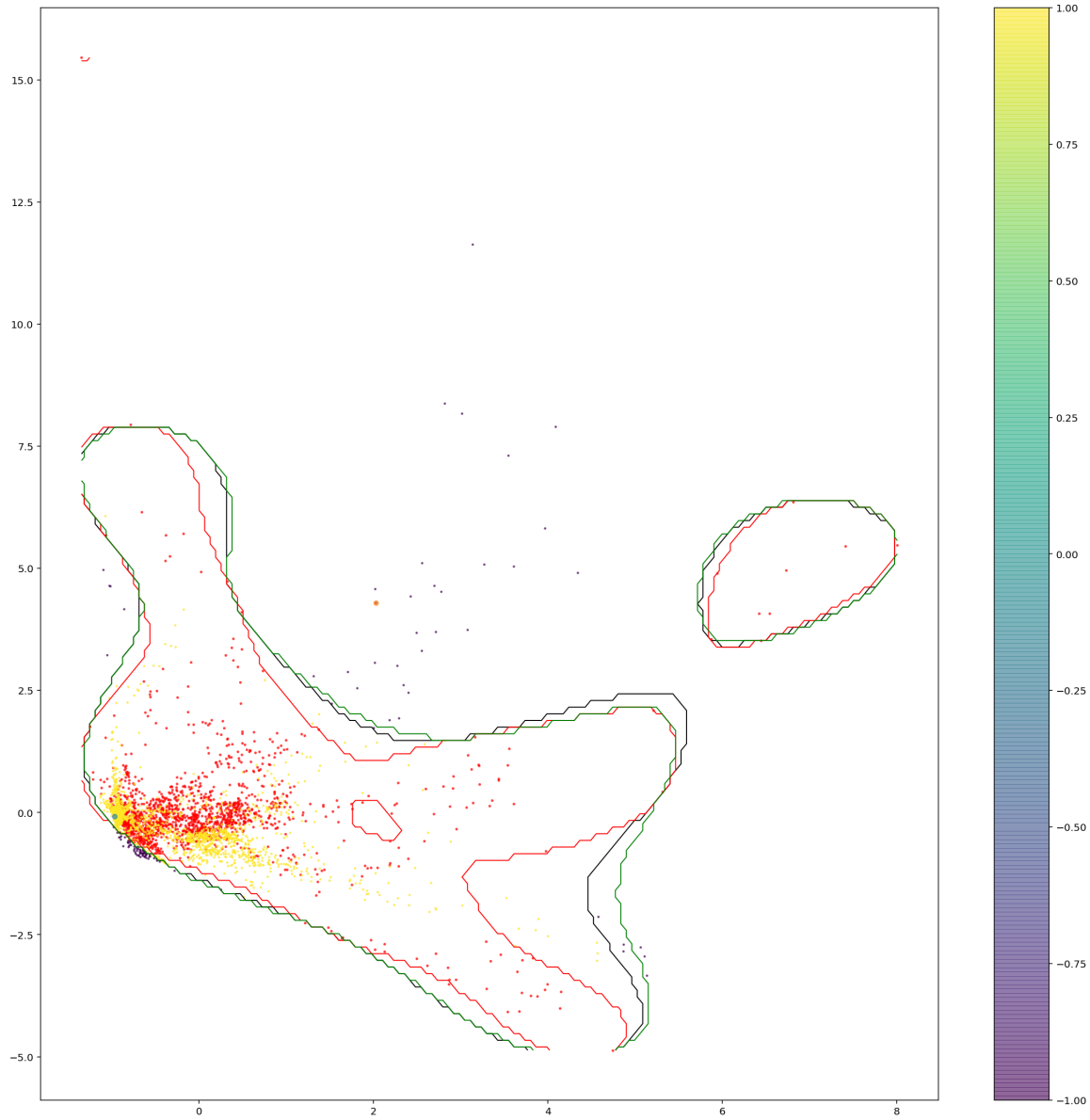
In [461]:

```
#0.770208387408 xtrain_new(29 feautres)
#0.753602902097 (34 feautres)
#0.654370141026 (46 feautres, missed,only 2 PCA)
#0.667249125539 (54 feautres 10 PCA)
#0.671207373898 (52 feautres 10 PCA)
#0.666995748205 (50 feautres 10 PCA)
#0.718394267492 (43 feautres 10 PCA gamma=0.1)
#0.704070723176 (43 feautres 10 PCA gamma=0.2)
#0.746682919896 (40 feautres 10 PCA gamma=0.1)
#0.756674329786 (34 feautres 10 PCA gamma=0.1)
#0.677039687542  (42 feautres 8 new derv gamma=0.1)
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_der_new[:,:])
sscore1 = -clf.score_samples(xtest_der_new[:,:])
print(xtrain_der_new.shape)
print(sscore1[:20])
np.savetxt('ytest_challenge_studentOCSVM01.csv', sscore1, fmt = '%1.6f', delimit
er=',')
```

```
(1677, 42)
[-0.00443653 -0.00942951 -0.00928733 -0.00760859 -0.00838195 -0.0075
135
 -0.00323282 -0.0082989  -0.00876187 -0.00782155 -0.00966054 -0.0065
1277
 -0.00759175 -0.00591305 -0.00864579 -0.00760524 -0.00847112 -0.0065
7944
 -0.00686223 -0.00805184]
```

In [399]:

```
#PCA
pca1 = PCA(n_components=2, whiten=True)
xtrain_fpca = pca1.fit_transform(xtrain_der_new)
xtest_fpca = pca1.transform(xtest_der_new)

from sklearn.svm import OneClassSVM
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_fpca)

c=clf.predict(xtest_fpca)
# Compare given classifiers under given settings
xmin = np.min([xtest_fpca[:, 0].min(), xtrain_fpca[:, 0].min()])
xmax = np.max([xtest_fpca[:, 0].max(), xtrain_fpca[:, 0].max()])
ymin = np.min([xtest_fpca[:, 1].min(), xtrain_fpca[:, 1].min()])
ymax = np.max([xtest_fpca[:, 1].max(), xtrain_fpca[:, 1].max()])
xx, yy = np.meshgrid(np.linspace(xmin, xmax, 150),
                     np.linspace(ymin, ymax, 150))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(20, 20))
#plt.xlim(-8,8)
#plt.ylim(-8,8)
plt.scatter(xtrain_fpca[:, 0], xtrain_fpca[:, 1],alpha=0.6,s=3,c='red')
plt.scatter(xtest_fpca[:, 0], xtest_fpca[:, 1],alpha=0.6,s=2,c=c)
plt.colorbar()
plt.scatter(xtest_fpca[2019, 0], xtest_fpca[2019, 1],alpha=0.6,s=20)
plt.scatter(xtest_fpca[2433, 0], xtest_fpca[2433, 1],alpha=0.6,s=20)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black')

clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.2).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='red')

clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.09).fit(xtrain_fpca)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='green')
plt.show()
```




In [405]:

```
#0.729426996428  (52 feautres 10 PCA)
#0.72833932787  (50 feautres 10 PCA)
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0 #, warm_start=True
                     ).fit(xtrain_der_new)

sscoreIF = -clf.score_samples(xtest_der_new)
print(sscore.var())
print(sscore[:20])
np.savetxt('ytest_challenge_studentIF3.csv', sscoreIF, fmt = '%1.6f', delimiter=
',')
```

```
1.0
[1.58843269 2.29438097 1.42773769 2.81452536 2.08281717 1.97398349
 1.13369299 1.18520263 2.00912182 2.84728323 1.12927474 1.99792463
 2.36259798 3.78273376 1.9965887  1.84444096 2.77184613 2.64075713
 2.19955593 1.90537194]
```

In [493]:

```
xtrain_FFT=np.abs(np.fft.fft(xtrain)[:,:30270])
xtest_FFT=np.abs(np.fft.fft(xtest)[:,:30270])
xtrain_FFT.shape
```

Out[493]:

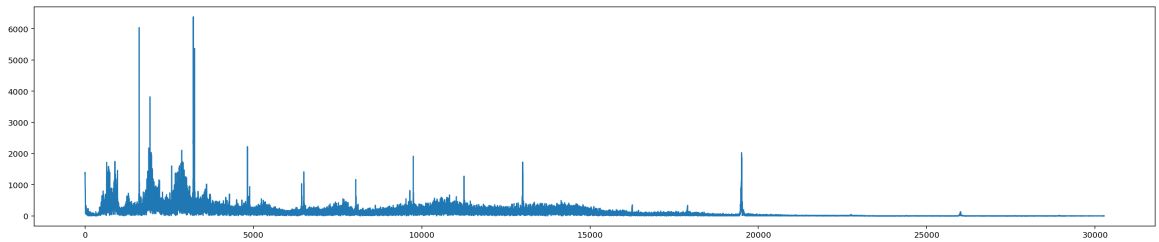
```
(1677, 30270)
```

In [494]:

```
pca10FFT = PCA(n_components=10, whiten=True)
xtrain_fpca10FFT = pca10.fit_transform(xtrain_FFT)
xtest_fpca10FFT = pca10.fit_transform(xtest_FFT)
```

In [481]:

```
bins = np.array([0.0, 1000, 2000, 3000, 4000, 5000, 10000, 15000, 20000,25000,30
270])
X = np.abs(fftpack.fft(xtrain[239])[:30270])
freqs = fftpack.fftfreq(len(x)) * 1024
plt.figure(figsize=(25,5))
plt.plot(X)
plt.show()
```




In [496]:

```
#0.677039687542 xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_stan
dartize
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_der_new[:,:])
sscore1 = -clf.score_samples(xtest_der_new[:,:])
print(xtrain_der_new.shape)
print(sscore1[:20])
np.savetxt('ytest_challenge_studentOCSVMFFTcombine01.csv', sscore1, fmt = '%1.6f
', delimiter=',')
```

```
(1677, 42)
[-0.00443653 -0.00942951 -0.00928733 -0.00760859 -0.00838195 -0.0075
135
 -0.00323282 -0.0082989  -0.00876187 -0.00782155 -0.00966054 -0.0065
1277
 -0.00759175 -0.00591305 -0.00864579 -0.00760524 -0.00847112 -0.0065
7944
 -0.00686223 -0.00805184]
```

In [539]:

```
#0.677039687542 only FFT
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_der_new[:,:])
sscore1 = -clf.score_samples(xtest_der_new[:,:])
print(xtrain_der_new.shape)
print(sscore1[:20])
np.savetxt('ytest_challenge_studentOCSVMFFTcombine01.csv', sscore1, fmt = '%1.6f
', delimiter=',')
```

```
(1677, 42)
[-0.00443653 -0.00942951 -0.00928733 -0.00760859 -0.00838195 -0.0075
135
 -0.00323282 -0.0082989  -0.00876187 -0.00782155 -0.00966054 -0.0065
1277
 -0.00759175 -0.00591305 -0.00864579 -0.00760524 -0.00847112 -0.0065
7944
 -0.00686223 -0.00805184]
```

In [498]:

```
xtrain_fpca10FFT.var(axis=0)
```

Out[498]:

```
array([0.9994037, 0.9994037, 0.9994037, 0.9994037, 0.9994037, 0.9994
037,
       0.9994037, 0.9994037, 0.9994037, 0.9994037])
```

In [503]:

```
xtrain_new3 = np.hstack((xtrain_matrix_standartize, xtrain_fpca10, xtrain_fpca10
FFT))
    #(xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_standartize))
xtest_new3 = np.hstack(
    (xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT))
print(xtest_new3.shape)

#0.648729405367  xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_new3[:,:])
sscore1 = -clf.score_samples(xtest_new3[:,:])
print(xtrain_new3.shape)
print(sscore1[:20])
np.savetxt('ytest_challenge_studentOCSVMFFTcombine01.csv', sscore1, fmt = '%1.6f
', delimiter=',')

#0.742655147268 xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0 #, warm_start=True
                     ).fit(xtrain_new3)

sscore = -clf.score_samples(xtest_new3)
print(sscore.var())
print(sscore[:20])
np.savetxt('ytest_challenge_studentIF3.csv', sscore, fmt = '%1.6f', delimiter=
',')
```

```
(2511, 44)
(1677, 44)
[-3.42859057e-03 -7.73254871e-03 -2.92579917e-03 -6.12241589e-03
 -6.81458834e-03 -6.02800120e-03 -8.73402396e-03 -3.53505438e-03
 -7.05659993e-03 -8.80331749e-05 -1.05319743e-02 -5.22921926e-03
 -6.11223602e-03 -4.78218429e-03 -7.04591246e-03 -6.11306075e-03
 -6.93136282e-03 -5.29006848e-03 -5.57460716e-03 -6.48786789e-03]
0.0011577922443019421
[0.48987058 0.4613875  0.452381   0.45822334 0.45372454 0.45100512
 0.43896832 0.44550176 0.45178198 0.49373922 0.43471634 0.45848828
 0.46129964 0.46679011 0.47779064 0.45381797 0.46435669 0.45068463
 0.45161717 0.45511166]
```

In [541]:

```
xtrain_new3 = np.hstack((xtrain_matrix_standartize, xtrain_fpca10,xtrain_der_mat
rix_standartize))
    #(xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_standartize))
xtest_new3 = np.hstack(
    (xtest_matrix_standartize, xtest_fpca10,xtest_der_matrix_standartize))
print(xtest_new3.shape)

#0.648729405367  xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
#0.756674329786 xtest_matrix_standartize, xtest_fpca10
#0.621091191121 xtest_matrix_standartize, xtest_fpca10,xtest_fpca10FFT,xtest_der
_matrix_standartize
#0.677039687542 xtest_matrix_standartize, xtest_fpca10,xtest_der_matrix_standart
ize
clf = OneClassSVM(nu=0.001,kernel="rbf", gamma=0.1).fit(xtrain_new3[:,:])
sscore3 = -clf.score_samples(xtest_new3[:,:])
print(xtrain_new3.shape)
print(sscore3[:20])
np.savetxt('ytest_challenge_studentOCSVMFFTcombine01.csv', sscore3, fmt = '%1.6f
', delimiter=',')

#0.742655147268 xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
#0.673853931057 xtest_matrix_standartize
# 0.564521611233 xtest_fpca10FFT
xtrain_new4 = np.hstack((xtrain_matrix_standartize, xtrain_fpca10,xtrain_fpca10F
FT))
    #(xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_standartize))
xtest_new4 = np.hstack(
    (xtest_matrix_standartize, xtest_fpca10,xtest_fpca10FFT))
print(xtest_new4.shape)

from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0 #, warm_start=True
                     ).fit(xtrain_new4)

sscore4 = -clf.score_samples(xtest_new4)
print(sscore4.var())
print(sscore4[:20])
np.savetxt('ytest_challenge_studentIF4.csv', sscore4, fmt = '%1.6f', delimiter=
',')

sscore3=sscore3/sscore3.std()
print(sscore3[:20])
sscore4=sscore4/sscore4.std()
print(sscore4[:20])
#stacking
#0.761058993659
sscoreS=sscore3+sscore4
print(sscoreS.var())
print(sscoreS[:20])
np.savetxt('ytest_challenge_student01.csv', sscoreS, fmt = '%1.6f', delimiter=
',')
```

```
(2511, 42)
(1677, 42)
[-0.00443653 -0.00942951 -0.00928733 -0.00760859 -0.00838195 -0.0075
135
 -0.00323282 -0.0082989  -0.00876187 -0.00782155 -0.00966054 -0.0065
1277
 -0.00759175 -0.00591305 -0.00864579 -0.00760524 -0.00847112 -0.0065
7944
 -0.00686223 -0.00805184]
(2511, 44)
0.0012234124563830706
[0.47560683 0.45902652 0.45199512 0.45920315 0.45021815 0.45441146
 0.43839565 0.44601991 0.44444467 0.50430138 0.43225311 0.45384431
 0.45615772 0.4658564  0.46200124 0.45324734 0.46187793 0.44704641
 0.45099632 0.45516687]
[-1.35790567 -2.88612644 -2.8426089  -2.32878989 -2.56549654 -2.2996
8373
 -0.9894807  -2.54007678 -2.68177846 -2.39397184 -2.95683668 -1.9933
8784
 -2.32363426 -1.80983068 -2.6462498  -2.32776433 -2.59278614 -2.0137
9382
 -2.1003474  -2.46445723]
[13.59758039 13.12354981 12.9225225  13.12859978 12.87171908 12.9916
0566
 12.53371418 12.7516915  12.70665539 14.41795627 12.35809919 12.9753
9084
 13.04153094 13.31881599 13.20859693 12.9583233  13.20507173 12.7810
392
 12.89396696 13.01320274]
3.343553952273305
[12.23967472 10.23742337 10.0799136  10.79980989 10.30622254 10.6919
2194
 11.54423347 10.21161472 10.02487693 12.02398443  9.40126251 10.9820
03
 10.71789668 11.5089853  10.56234713 10.63055897 10.61228559 10.7672
4538
 10.79361956 10.54874551]
```

In [518]:

```
#0.690006736129 xtrain_matrix_standartize, xtrain_fpca10,xtrain_fpca10FFT,xtrain
_der_matrix_standartize
xtrain_new4 = np.hstack((xtrain_matrix_standartize, xtrain_fpca10,xtrain_fpca10F
FT,xtrain_der_matrix_standartize))
    #(xtrain_matrix_standartize, xtrain_fpca10, xtrain_der_matrix_standartize))
xtest_new4 = np.hstack(
    (xtest_matrix_standartize, xtest_fpca10,xtest_fpca10FFT,xtest_der_matrix_sta
ndartize))
print(xtest_new4.shape)

from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0 #, warm_start=True
                     ).fit(xtrain_new4)
sscore4 = -clf.score_samples(xtest_new4)
print(sscore4.var())
print(sscore4[:20])
np.savetxt('ytest_challenge_student01.csv', sscore4, fmt = '%1.6f', delimiter=
',')
```

```
(2511, 52)
0.001155850564077948
[0.47323116 0.44913356 0.45905627 0.45849546 0.4380647  0.44531605
 0.44811877 0.44790441 0.44655052 0.48725525 0.43799479 0.44965965
 0.45353957 0.45934348 0.45936796 0.44748463 0.45235398 0.43458311
 0.44543828 0.4469312 ]
```

In [ ]:

```
sscore3=sscoreSVM34/sscoreSVM34.std()
print(sscore3[:20])
sscore4=sscoreS/sscoreS.std()
print(sscore4[:20])
#stacking
#0.741657087767
sscoreS2=sscore3+sscore4
print(sscoreS2.var())
print(sscoreS2[:20])
np.savetxt('ytest_challenge_studentStacking01.csv', sscoreS2, fmt = '%1.6f', del
imiter=',')
```

In [536]:

```
xtrain_FFT2 = np.zeros((len(xtrain_FFT), int(30270/30)))
xtest_FFT2 = np.zeros((len(xtest_FFT), int(30270/30)))
for i in range(len(xtrain_FFT)):
    for j in range(int(30270/30)):
        xtrain_FFT2[i, j] = xtrain_FFT[i,j*1009:(j+1)*1009].sum()

pca10FFT = PCA(n_components=10, whiten=True)
xtrain_fpca10FFT = pca10FFT.fit_transform(xtrain_FFT)
xtest_fpca10FFT = pca10FFT.fit_transform(xtest_FFT)

pca10FFT2 = PCA(n_components=10, whiten=True)
xtrain_fpca10FFT2 = pca10FFT2.fit_transform(xtrain_FFT2)
xtest_fpca10FFT2 = pca10FFT2.fit_transform(xtest_FFT2)
```

```
/Users/yang/anaconda3/lib/python3.7/site-packages/sklearn/decomposit
ion/pca.py:535: RuntimeWarning: invalid value encountered in true_di
vide
  self.explained_variance_ / total_var.sum()
```

In [538]:

```
#0.567436995563
clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.1).fit(xtrain_fpca10FFT)
sscore3 = -clf.score_samples(xtest_fpca10FFT)
print(sscore3[:20])
np.savetxt('OCSVMFFT05_rawFFT1.csv',
           sscore3, fmt='%1.6f', delimiter=',')

#0.499144079004
clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.1).fit(xtrain_fpca10FFT2)
sscore3 = -clf.score_samples(xtest_fpca10FFT2)
print(sscore3[:20])
np.savetxt('OCSVMFFT05_rawFFT2.csv',
           sscore3, fmt='%1.6f', delimiter=',')
```

```
[-0.0557487  -0.05636784 -0.01729834 -0.05615818 -0.05618354 -0.0561
4959
 -0.04511921 -0.03698752 -0.05638849 -0.0047561  -0.04538377 -0.0563
4506
 -0.05609903 -0.05629726 -0.05602952 -0.0561985  -0.05629062 -0.0555
3469
 -0.0561769  -0.0563625 ]
[-0.         -0.         -0.         -0.         -0.         -0.
 -0.         -0.         -0.         -0.         -0.05441855 -0.0544
1855
 -0.05441855 -0.05441855 -0.05441855 -0.05441855 -0.05441855 -0.0544
1855
 -0.05441855 -0.05441855]
```

In [556]:

```
### double the datas
pca10 = PCA(n_components=10, whiten=True)
xtrain_fpca102 = pca10.fit_transform(-xtrain)
```

In [572]:

```
### double the datas, anther way to do PCA
pcaDouble = PCA(n_components=20, whiten=True)
#xtrain_double = np.vstack((xtrain, -xtrain))
xtrain_fpca102 = pcaDouble.fit_transform(xtrain_double)
xtest_fpca102 = pcaDouble.fit_transform(xtest)
```

In [569]:

```
xtrain_double = np.vstack(
    (xtrain, -xtrain))
xtrain_double.shape
```

Out[569]:

```
(3354, 61440)
```

In [565]:

```
xtrain_matrix_combined = np.vstack(
    (xtrain_matrix, xtrain_matrix2))
xtrain_matrix_standartize_combined=np.zeros(xtrain_matrix_combined.shape)
for i in range(len(xtrain_matrix2[1])):
     xtrain_matrix_standartize_combined[:,i]=xtrain_matrix_combined[:,i]/xtrain_
matrix_combined[:,i].std()

xtrain_fpca10_combined = np.vstack((xtrain_fpca10, xtrain_fpca102))
```

In [571]:

```
from sklearn.ensemble import IsolationForest
xtrain_new10 = np.hstack((xtrain_matrix_standartize_combined, xtrain_fpca102))
xtest_new10 = np.hstack((xtest_matrix_standartize, xtest_fpca102))
# 0.648729405367  xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
# 0.756674329786 xtest_matrix_standartize, xtest_fpca10
# 0.621091191121 xtest_matrix_standartize, xtest_fpca10,xtest_fpca10FFT,xtest_de
r_matrix_standartize
# 0.677039687542 xtest_matrix_standartize, xtest_fpca10,xtest_der_matrix_standar
tize
# 0.760356026055 doubled: xtest_matrix_standartize, xtest_fpca10  gamma=0.1
# 0.775255849308 doubled: xtest_matrix_standartize, xtest_fpca10 gamma=0.2
# 0.783485977728 doubled: xtest_matrix_standartize(arpès combine), xtest_fpca10
# 0.786086185373 doubled: xtest_matrix_standartize(arpès combine), xtest_fpca10
 (arpès combine)
clf = OneClassSVM(nu=0.001, kernel="rbf", gamma=0.2).fit(xtrain_new10)
sscore3 = -clf.score_samples(xtest_new10)
print(xtrain_new10.shape)
print(sscore3[:20])
np.savetxt('OCSVM10.csv', sscore3, fmt='%1.6f', delimiter=',')

# 0.742655147268 xtest_matrix_standartize, xtest_fpca10, xtest_fpca10FFT
# 0.673853931057 xtest_matrix_standartize
# 0.564521611233 xtest_fpca10FFT
# 0.771971213863 doubled: xtest_matrix_standartize, xtest_fpca10
# 0.773446673341 doubled: xtest_matrix_standartize(arpès combine), xtest_fpca10
# 0.754960942811 doubled: xtest_matrix_standartize(arpès combine), xtest_fpca10
 (arpès combine)
clf = IsolationForest(n_estimators=400, max_samples='auto', contamination=0.001,
                      max_features=1,
                      bootstrap=True, n_jobs=-1, behaviour='deprecated',
                      random_state=None, verbose=0  # , warm_start=True
                      ).fit(xtrain_new10)

sscore4 = -clf.score_samples(xtest_new10)
print(sscore4.var())
print(sscore4[:20])
np.savetxt('FI10.csv', sscore4, fmt='%1.6f', delimiter=',')

sscore3 = sscore3/sscore3.std()
print(sscore3[:20])
sscore4 = sscore4/sscore4.std()
print(sscore4[:20])
# stacking
# 0.761058993659 origine
# 0.765063591531 double
# 0.78822907783
# 0.779923245207
sscoreS = sscore3+sscore4
print(sscoreS.var())
print(sscoreS[:20])
np.savetxt('ytest_challenge_student01.stacking2.csv',
           sscoreS, fmt='%1.6f', delimiter=',')
```

```
(3354, 34)
[-0.00403056 -0.00935557 -0.01127361 -0.00667698 -0.00756247 -0.0067
0669
 -0.01371685 -0.01290284 -0.00868895 -0.00530194 -0.0150788  -0.0054
4682
 -0.00703978 -0.00334936 -0.00826893 -0.00677875 -0.00785601 -0.0041
3697
 -0.00574398 -0.00751094]
0.0015603034184112719
[0.52480724 0.47737802 0.43302367 0.5006259  0.45894364 0.4616681
 0.43915557 0.43086973 0.46644261 0.4723526  0.42953452 0.47415274
 0.46828578 0.4994096  0.49351377 0.4640147  0.47464292 0.46815484
 0.4682558  0.46007543]
[-1.06000384 -2.46043819 -2.96486461 -1.75599134 -1.98886738 -1.7638
0346
 -3.60741795 -3.39333847 -2.28512208 -1.39436669 -3.96559937 -1.4324
6739
 -1.85140307 -0.88085297 -2.17465944 -1.78275573 -2.06606431 -1.0879
8893
 -1.51061771 -1.97531539]
[13.28603187 12.0853125  10.96243689 12.67385646 11.6186275  11.6876
0003
 11.11767214 10.90790766 11.80847155 11.95808886 10.87410547 12.0036
6141
 11.85513336 12.64306459 12.49380551 11.74700661 12.01607086 11.8518
1836
 11.85437439 11.64727978]
3.4905374178640503
[12.22602803  9.62487431  7.99757228 10.91786512  9.62976013  9.9237
9657
  7.51025419  7.51456919  9.52334946 10.56372217  6.9085061  10.5711
9402
 10.00373029 11.76221162 10.31914608  9.96425088  9.95000656 10.7638
2943
 10.34375668  9.67196439]
```

# History

## Prepare a file for submission

In [5]:

```
# Save the anomaly scores to file
print(sscore.shape)
np.savetxt('ytest_challenge_student.csv', sscore, fmt = '%1.6f', delimiter=',')
```

```
(2511,)
```

## Best score:79% ensemble OCSVM+Isolation Forest

Features: 10 raw features (PCA) + 24 new statistiques features (high peaks, low peaks, average curve, intervalle betweens each peaks...)