Beat detection

November 23, 2021

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py as h5
import os
import scipy.io
```

[2]:
```
from sys import platform
print(platform)
if platform == "linux" or platform == "linux2":  # linux
    volume = '/media/julian/Volume/'
elif platform == "win32": # Windows.
    volume = 'G:/'
```

```
linux
```

0.0.1 PTBXL

[3]:
```
p = os.path.join(volume, 'data/ECG/
↪ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
↪generated/1000/normalized-labels/train/00006_hr.h5')
full_data = None
with h5.File(p, 'r') as f:
    full_data = np.copy(f['data'])
data=full_data
print(data.shape)
```

```
(10000, 12)
```

[4]:
```
full_data = None
with h5.File('/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/
↪generated/normalized/test/patient007-s0078lre.h5', 'r') as f:
    full_data = np.copy(f['data'])
data=full_data
print(data.shape)
```

```
(115200, 12)
```

0.0.2 Challenge

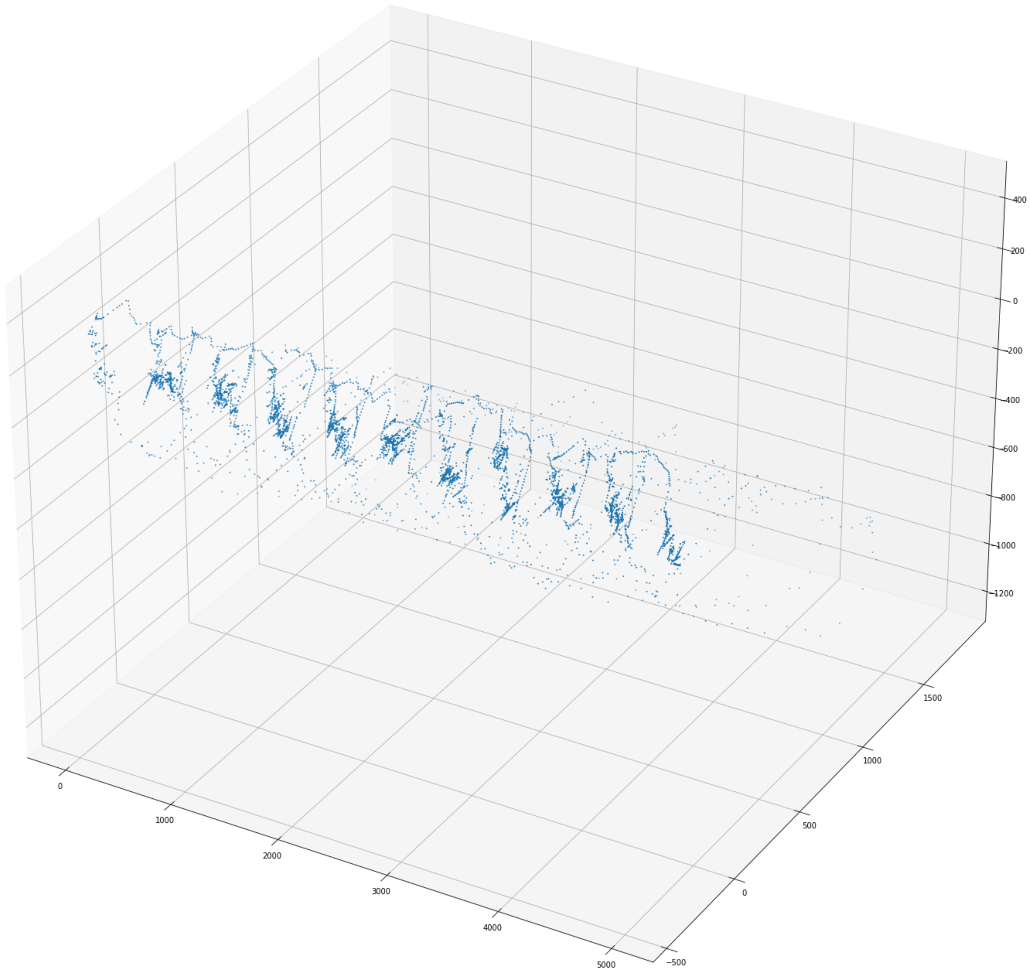
[20]:
```
data = np.array(scipy.io.loadmat('/media/julian/data/data/ECG/ptbxl_challenge/
↪HR21380.mat')['val']).T
full_data = data
data.shape
```

[20]:
```
(5000, 12)
```

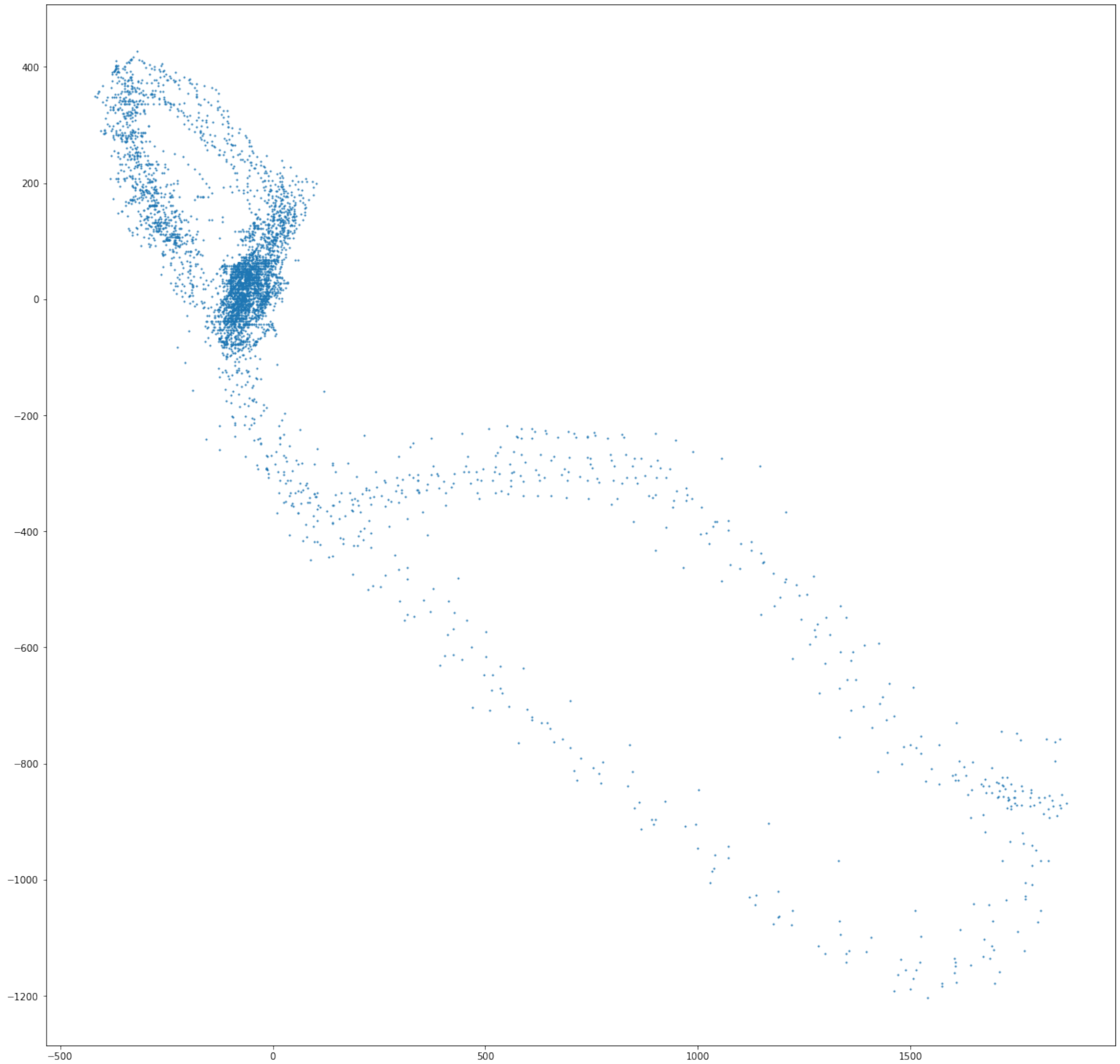
[69]:
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random
data = full_data - np.mean(data, axis=0)

fig = plt.figure(figsize=(20, 20))
ax = Axes3D(fig, auto_add_to_figure=False)

ax.scatter(np.arange(len(data)), data[:, 0], data[:, 1], s = 1)
fig.add_axes(ax)
plt.show()
```

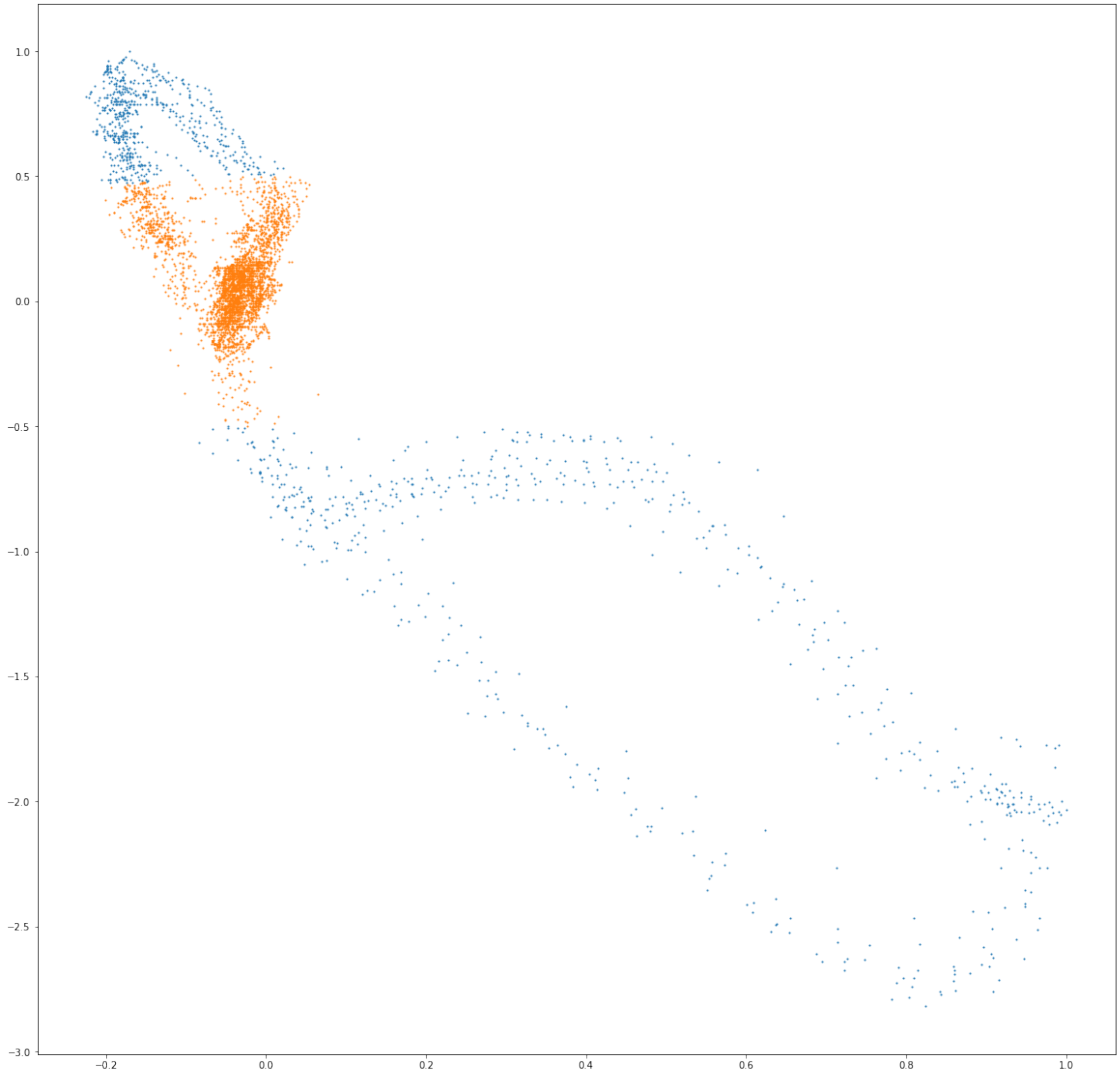


[23]:
```
plt.figure(figsize=(20, 20))
plt.scatter(data[:, 0], data[:, 1], s=1)
plt.show()
```



[25]:
```
data = (data-np.mean(data, axis=0))/(np.max(data, axis=0) - np.mean(data,␣
↪axis=0))
dists = np.sqrt(np.square(data[:,0])+np.square(data[:,1]))
print(dists)
medi = np.median(dists, axis=0)
print(medi)
mean = np.mean(dists, axis=0)
tresh = 0.5
x1 = data[:, 0]
x2 = data[:, 1]
plt.figure(figsize=(20, 20))
plt.scatter(x1[dists>tresh], x2[dists>tresh], s=1)
plt.scatter(x1[dists<=tresh], x2[dists<=tresh], s=1)
plt.show()
```

```
[0.68630151 0.68630151 0.68630151 … 0.17265697 0.17265697 0.17265697]
0.1637620449912468
```



[6]:
```
np.min(data, axis=0)
```

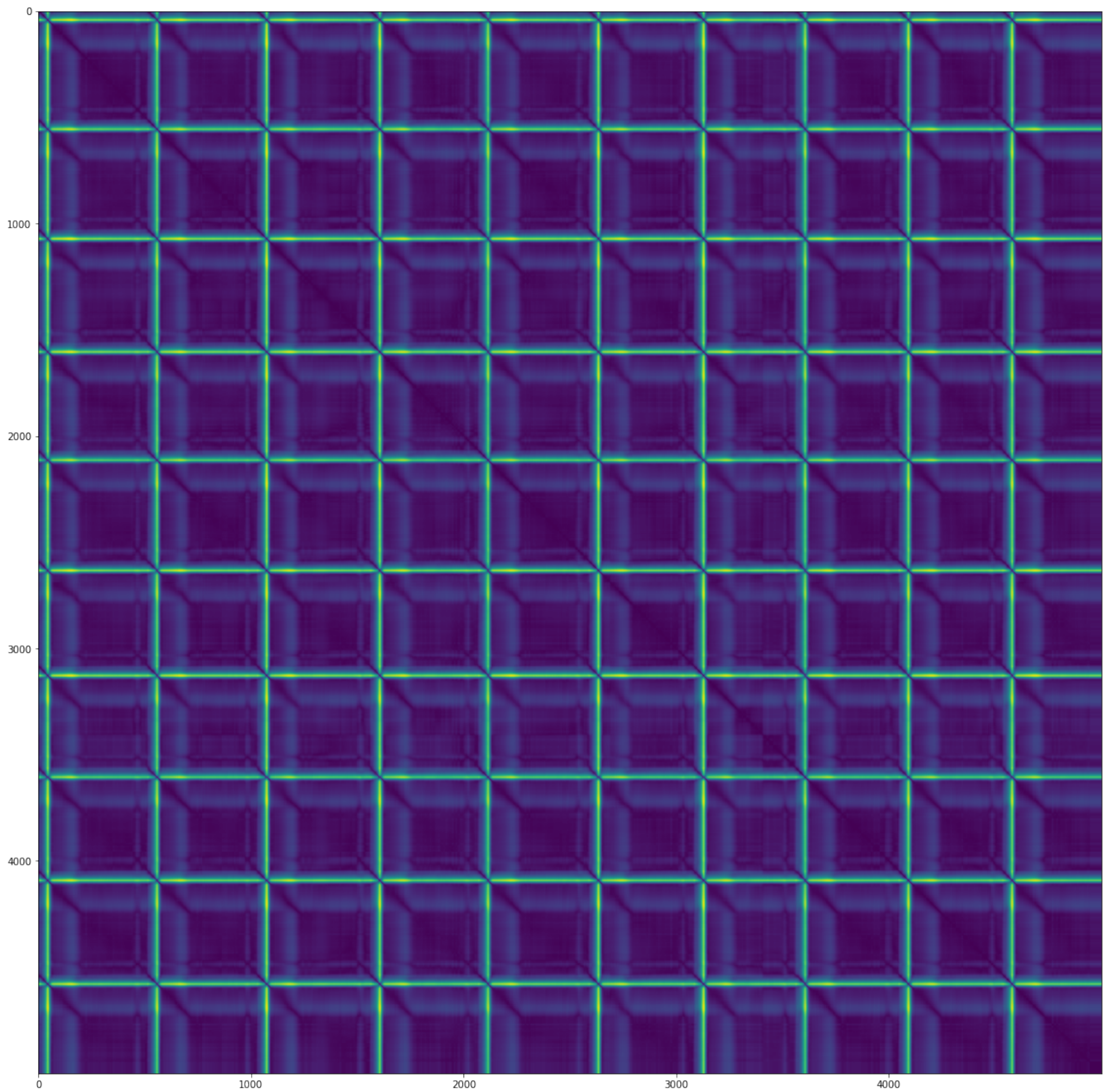
[6]:
```
array([-1.06313741, -0.60005059])
```

0.1 Calculating distance matrix between points

[27]:
```
from scipy.spatial import distance as d
#data = full_data[0:10000, :]
distv = d.pdist(data)
distm = d.squareform(distv)
print(data.shape)
```

```
(5000, 12)
```

[28]:
```
plt.figure(figsize=(20,20))
plt.imshow(distm)
plt.show()
```



0.2 Summing columns of distance matrix

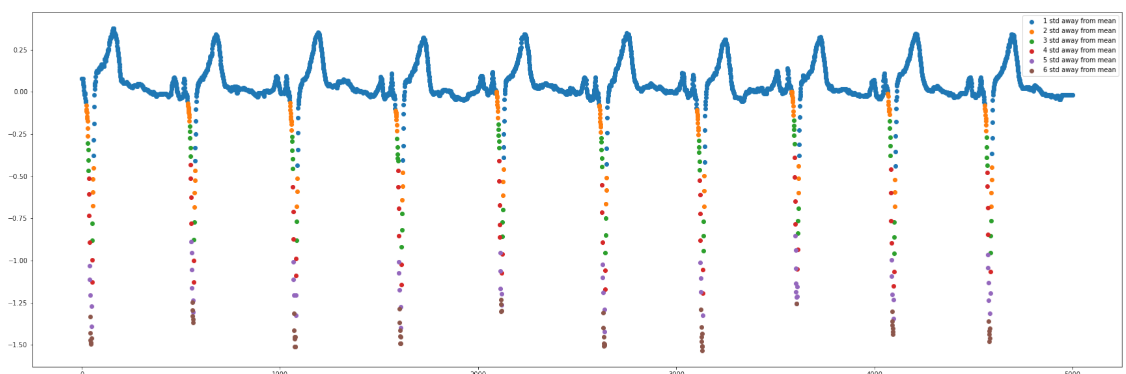
- Sum data in each column up
- Calculate mean and variance
- Divide data into bins where points are more than {0, 1, 2, …} variances away from mean.

[29]:
```python
sums = np.sum(distm, axis=1)/len(distm)
sums_inv = np.exp(-sums)
m = np.mean(sums)
s = np.std(sums)
print('mean', m, 'standard deviation', s)
devs = []
for i in range(1, 100):
    devs += [np.argwhere((np.abs(sums-m) >= (i-1)*s) & (np.abs(sums-m)<i*s))]
    if len(devs[-1]) == 0:
        devs.pop()
        break
print('elements in bins:', *map(len, devs))
```

```
mean 1.5651479328019686 standard deviation 1.192915638873555
elements in bins: 4626 127 75 58 55 59
```

0.3 Plot data (mean over all channels) with colors depending on bin

[30]:
```python
plt.figure(figsize=(30, 10))
for i,d in enumerate(devs):
    plt.scatter(np.arange(len(sums))[d], np.mean(data, axis=1)[d],␣
 ↪label=str(i+1)+' std away from mean')
#plt.plot(sums_inv, label='inverted sums (exp(-sums))')
plt.legend()
plt.show()
```

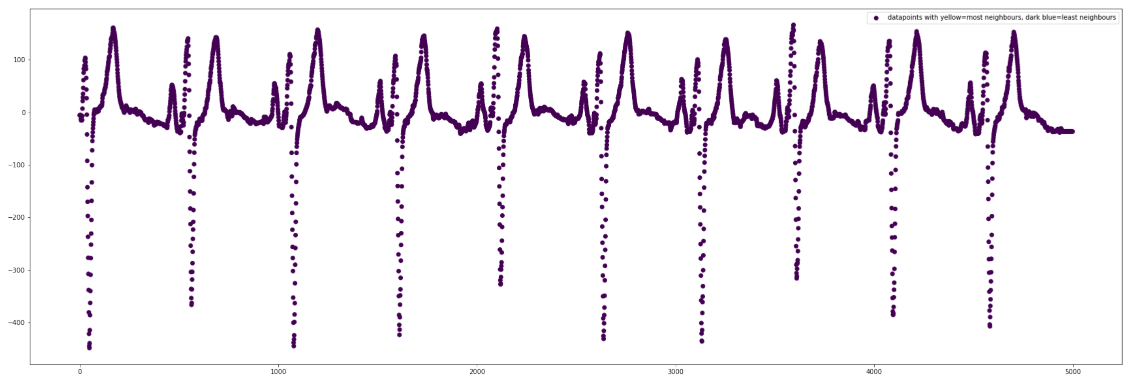


[34]:
```python
from scipy.spatial import distance as d
#data = full_data[0:10000, :]
distv = d.pdist(data)
distm = d.squareform(distv)
print(data.shape)
```

```
(5000, 12)
```

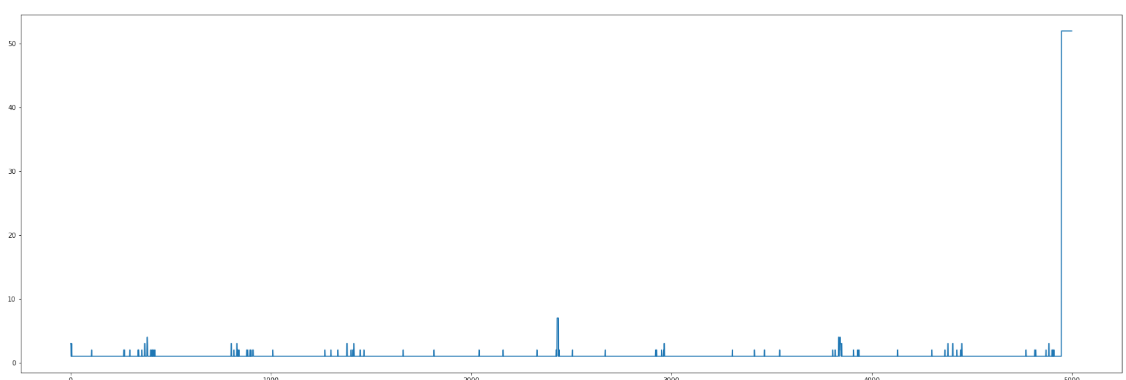
[35]:
```python
epsilon = np.var(distm)
neighbours = []
for i in range(distm.shape[1]):
    column = distm[:, i]
    neighbours += [np.argwhere(column<=epsilon)]
neighbour_counts = np.array(list(map(len,neighbours)))
```

[38]:
```python
plt.figure(figsize=(30, 10))
plt.scatter(np.arange(len(data)), np.mean(data, axis=1), c=neighbour_counts/max(neighbour_counts), label='datapoints with yellow=most neighbours, dark blue=least neighbours')
#plt.plot(neighbour_counts/max(neighbour_counts), label='neighbour counts (scaled down)')
plt.legend()
plt.show()
```



[39]:
```python
plt.figure(figsize=(30, 10))
plt.plot(np.sum(distm<=s, axis=1))
```

[39]: [<matplotlib.lines.Line2D at 0x7f12611ddad0>]

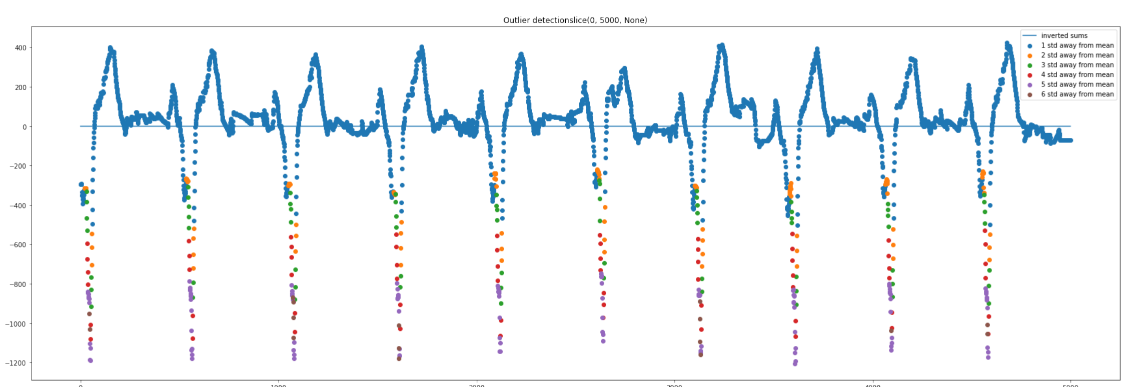


[40]:
```python
def _calc_and_plot(full_data, interval):
    data = full_data[interval]
    sums = np.sum(distm, axis=1)/len(distm)
    sums_inv = np.exp(-sums)
    m = np.mean(sums)
    s = np.std(sums)
    print('mean', m, 'standard deviation', s)
    devs = []
    for i in range(1, 100):
        devs += [np.argwhere((np.abs(sums-m) >= (i-1)*s) & (np.
↪abs(sums-m)<i*s))]
        if len(devs[-1]) == 0:
            devs.pop()
            break
    print('elements in bins:', *map(len, devs))
    plt.figure(figsize=(30, 10))
    plt.title("Outlier detection" + str(interval))
    for i,d in enumerate(devs):
        plt.scatter(np.arange(len(sums))[d], data[d, 1], label=str(i+1)+' std␣
↪away from mean')
    plt.plot(sums_inv, label='inverted sums')
    plt.legend()
    plt.show()
```

[41]:
```python
def calc_and_plot(full_data):
    base_size = 10000
    n_windows = (len(full_data)-3)/base_size
    window_size = int(base_size*(n_windows-int(n_windows)))
    for i in range(int(n_windows)):
        _calc_and_plot(full_data, slice(i*window_size, (i+1)*window_size))
```

[48]:
```python
_calc_and_plot(full_data, slice(0, len(full_data)))
```

```
mean 954.6658028454606 standard deviation 806.4016268791196
elements in bins: 4608 130 68 61 117 16
```

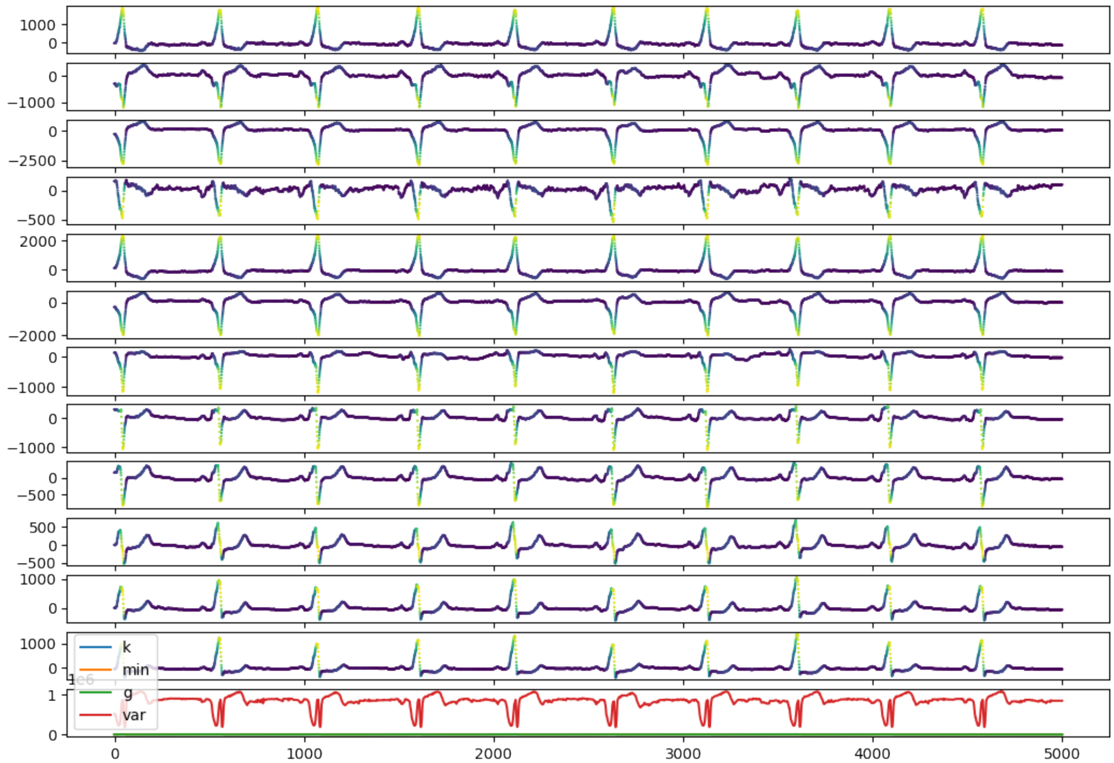


[49]:
```
def outlier_detection_nearest_epsilon(sorted_cols, k):
    l = []
    for col in sorted_cols:
        nei = col[0:k]
        kappa = nei[-1]
        min_dist = nei[0]
        gamma = np.mean(nei)
        var = np.var(nei)
        l.append([kappa, min_dist, gamma, var])
    return map(np.array, zip(*l))
```

[53]:
```
def plot_metrics(data, kappas, min_dists, gammas, var, save=None):
    x = np.arange(len(data))
    fig, axs = plt.subplots(data.shape[1]+1, 1, figsize=(len(data)/4/100, data.
 ↪shape[1]*300/4/100), dpi=100, sharex=True)
    for i in range(len(axs)-1):
        axs[i].scatter(x, data[:, i], c=kappas, s=0.5)
    axs[-1].plot(kappas, label='k')
    axs[-1].plot(min_dists, label='min')
    axs[-1].plot(gammas, label='g')
    axs[-1].plot(var, label='var')
    if save:
        plt.savefig(save, dpi=100)
    else:
        plt.legend()
        plt.show()
    plt.close()
```

[62]:
```
kappas, min_dists, gammas, var = map(np.array,␣
 ↪outlier_detection_nearest_epsilon(distm, 10000))
```

[63]:
```
plot_metrics(data, kappas, min_dists, gammas, var, save=None)
```




```
[ ]: sorted_cols = np.sort(distm, axis=1)
     for i in range(1, len(distm)//2, 2):
         kappas, min_dists, gammas, var = ␣
      ↪outlier_detection_nearest_epsilon(sorted_cols, k=i)
         plot_metrics(data, kappas, min_dists, gammas, var, save=None)
         print('Completed {} of {}'.format(i, len(distm)//2), flush=True)
```

```
[ ]:
```