demo_outlier

September 18, 2018

Demostration of the various functions of mod_sc.py

1 Single contextual anomalies

First identify Contextual anomalies (Talga et al 18 Figure 1a)

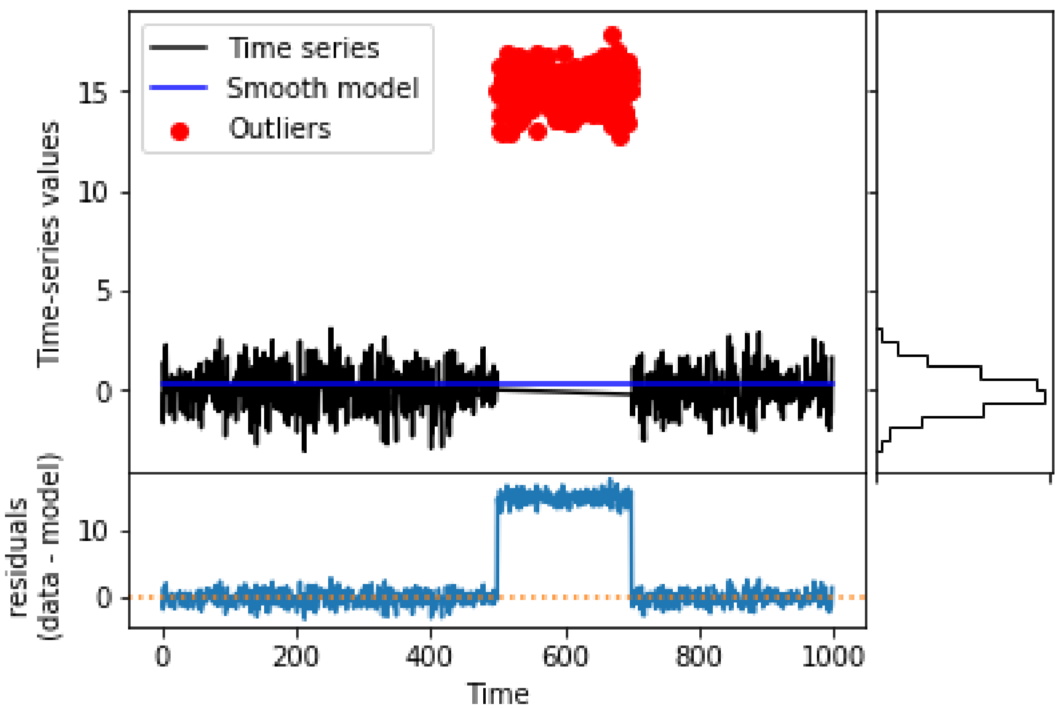
```
In [1]: import numpy as np
        import scipy
        import matplotlib.pylab as plt
        import scipy.optimize as mcf
        import scipy.signal as ssig
        import matplotlib.gridspec as gridspec
        import outlier_rejection as orej


        #generate some fake random data to test the code. Specify the parameters of the fake d
        sd_true = 1.0
        n_true  = 1000
        n_outlier = 20
        mean_outlier = 10.0
        sd_outlier = 1.0


        #make the fake data
        data_y = np.random.randn(n_true)*sd_true
        id_test = np.random.choice(np.arange(n_true), size=n_outlier, replace=False)
        data_y[id_test] = np.random.randn(n_outlier)*sd_outlier + mean_outlier
        idneg = np.random.choice(np.arange(n_outlier), size=n_outlier/2, replace=False)
        data_y[id_test[idneg]] = -1*data_y[id_test[idneg]]

        #Call the outlier rejection function defined above and test on the fake data.
        id_out = orej.outlier_smooth(data_y,sd_check=4,
        fname='running median',filter_size = 5,max_iteration=10,diagnostic_figure='show')
```

```
<Figure size 640x480 with 3 Axes>
```

2 Clusters of outliers

Now attempt to flag anomalous sub-sequences within a given series (Figure 2b Talagala et al 2018). Here a running median model is innapropriate as it would consider the outlying sequence as normal unless a very larger filter size is assumed. A global median is a better choice of smooth function here as the anomalous sub-sequence is still outnumbered by the abundance of well behaved data. A 'global median' fit is also better than a 'global mean' as it is robust to outliers and in this case completely disregards the outliying sequence.

```
In [2]: # define the random sub-sequence
        subsequence_size = 200
        data_y = np.random.randn(n_true)*sd_true
        id_test = np.arange(n_true/2,n_true/2+subsequence_size)
        data_y[id_test] = np.random.randn(subsequence_size)*sd_outlier + 1.5*mean_outlier


        #call to 'outlier_smooth'
        idx_outlier = orej.outlier_smooth(data_y,sd_check=1,fname='global median',runtype='ser
                                          filter_size = 5,max_iteration=1,diagnostic_figure='sl
```



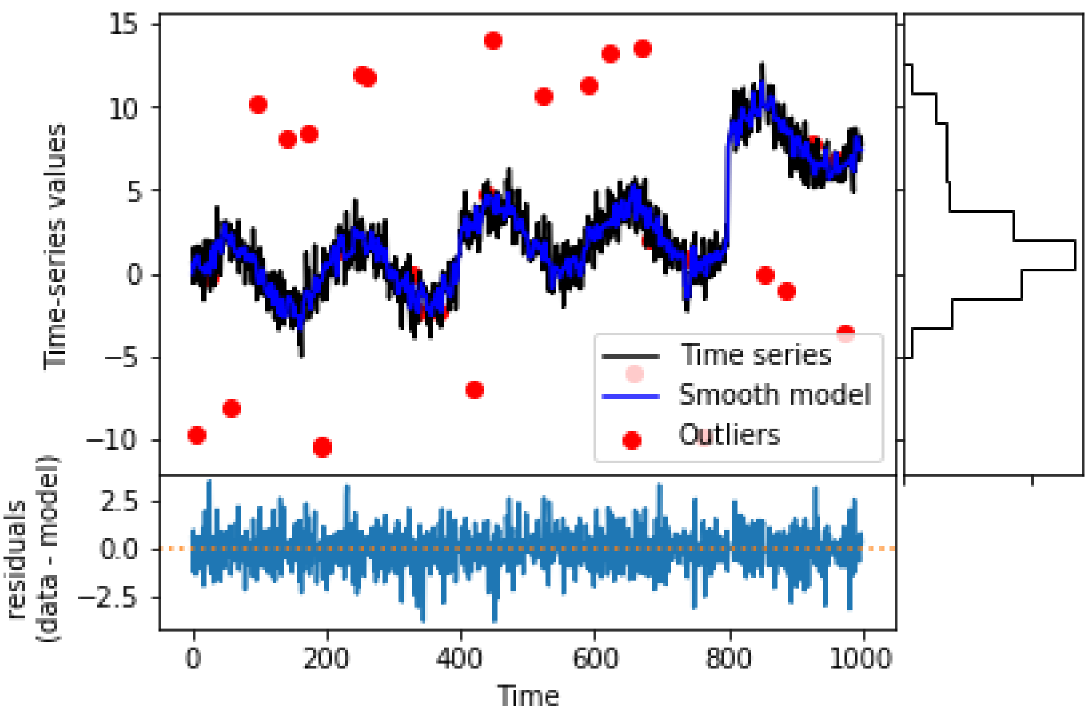

```
<Figure size 432x288 with 0 Axes>
```

3 Outliers in nonstationary time series

Now try a slightly more complex model. Simulate non-stationary time series (in this case some combination of periodic signal that increases with time as a polynomial).

```
In [3]: #make the fake data
        period = 200.0
        amplitude = 2
        t_ref = 400
        amp_poly = 2
        data_y = np.random.randn(n_true)*sd_true +  amplitude*np.sin(2*np.pi/period*np.arange(r
        id_test = np.random.choice(np.arange(n_true), size=n_outlier, replace=False)

        idneg = np.random.choice(np.arange(n_outlier), size=n_outlier/2, replace=False)
        mirror = np.ones(n_outlier)
        mirror[idneg] = -1.
        data_y[id_test] = amplitude*np.sin(2*np.pi/period*id_test) + amp_poly*(id_test/t_ref)**
```

```
In [4]: id_out = orej.outlier_smooth(data_y,sd_check=3,fname='running median',runtype='series'
```



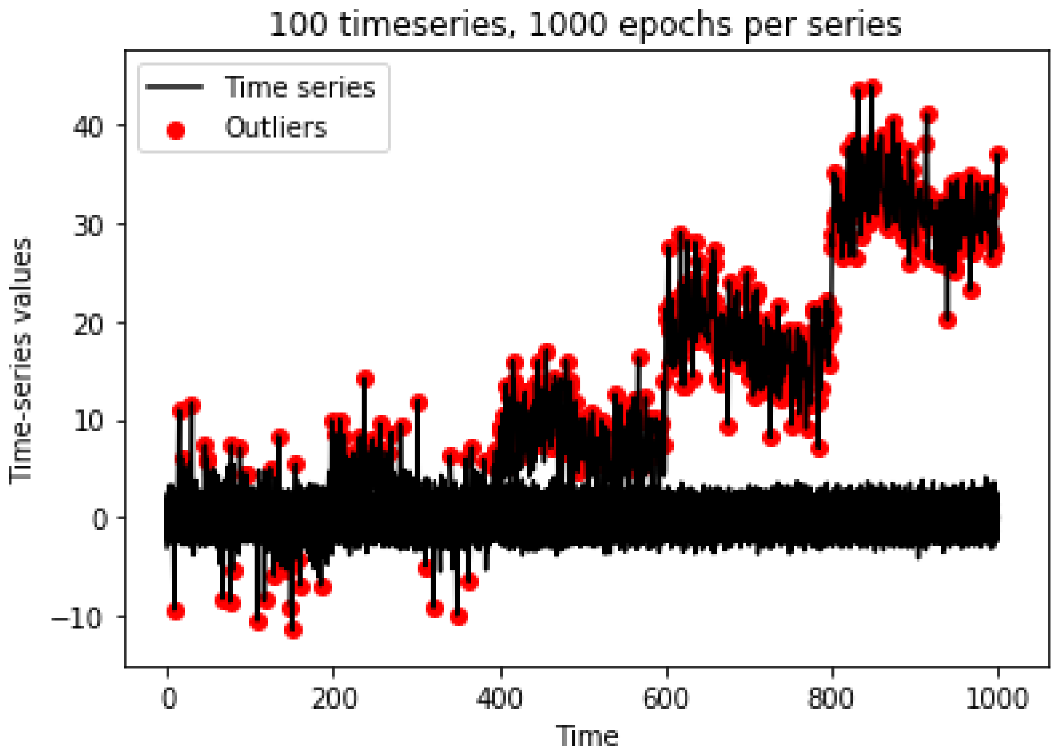

```
<Figure size 432x288 with 0 Axes>
```

The 'outlier_smooth' fits a smooth function to identify outliers inconsistent with the evolving time series. A standard sigma clip using just the mean and standard deviation would fail here as the distribution is now multimodal and non-stationary.

4 Identify outliers from multi-variate time series

The examples above are all looking for outliers from within a single time-series. Now introduce a set of multiple time series data with one entire anomalous time series. The objective is now to identify anomalies between multiple time series rather than within a single time series (Figure 2c from Talagala et al 2018).

```
In [5]: #make the fake data
        period = 200.0
        amplitude = 2
        t_ref = 200
        amp_poly = 2

        sd_background = 3.0
        n_epoch  = 1000
        n_timeseries = 100
        id_outlier = 23
        time_anomaly = 35
        grad_anomaly = 0.1
        diagnostic_figure = 'show'
        data_y = np.reshape( np.random.randn(n_epoch * n_timeseries), (n_epoch,n_timeseries) )

        data_y[:,time_anomaly] = np.random.randn(n_epoch)*sd_background +  amplitude*np.sin(2*n
        id_test = np.random.choice(np.arange(n_epoch), size=n_outlier, replace=False)

        idneg = np.random.choice(np.arange(n_outlier), size=n_outlier/2, replace=False)
        mirror = np.ones(n_outlier)
        mirror[idneg] = -1.
        data_y[id_test,time_anomaly] = amplitude*np.sin(2*np.pi/period*id_test) + amp_poly*(id_

In [6]: id_out = orej.outlier_smooth(data_y,sd_check=5,fname='running median',runtype='paralle
```



In the above figure we have the same increasing sinusoid as with the previous example, but we also have 99 well behaved stationary time series that oscilate around zero. 'outlier_smooth' now flags the entire series as 'bad' as the iterative-smooth-model fitting now takes place epoch-by-epoch across all the 100 time-series rather than across all the epochs for a single time series.

5 Feature space identification

In some cases outliers manifest gradualy and are correlated across adjacent time-series. Talgala et al 2018 gives an example of a gas pipe with a hole where not only the sensor nearest the hole exhibit an anomaly, but the damage also affects adjacent sensors. Rather than a single anomalous time series as above, we see several. An example is given below.

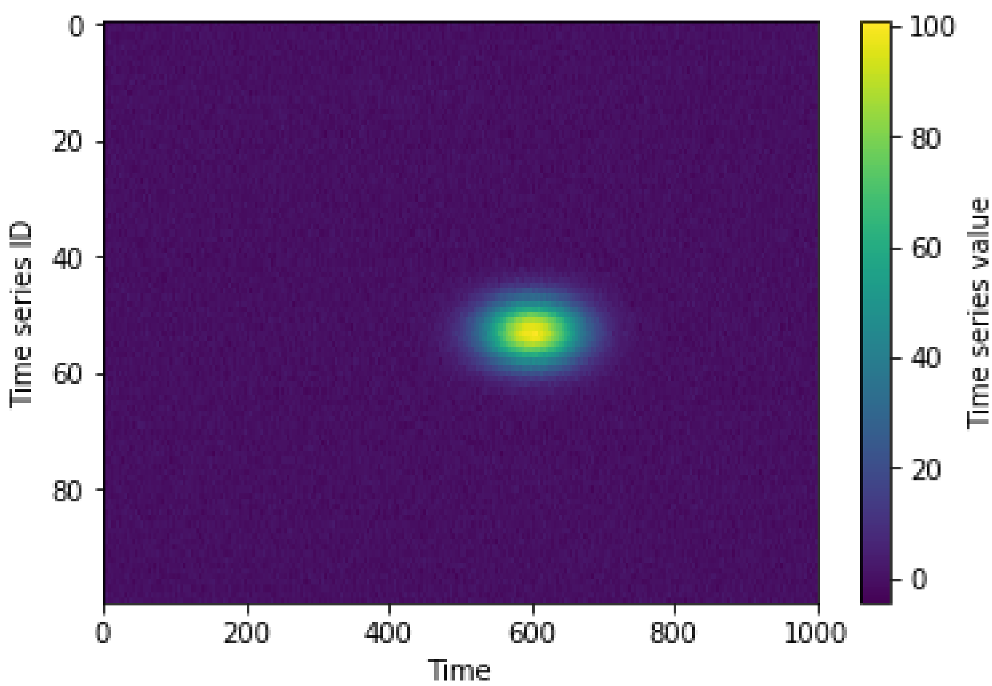
```
In [7]: #introduce some time-varying noise into the model
        sd_ts = 4.0
        sd_epoch = 50.0
        amp_ts_max = 100
        amp_epoch = 100.0
        mean_ts = 53.0
        mean_epoch = 600.0



        nts = 100
```

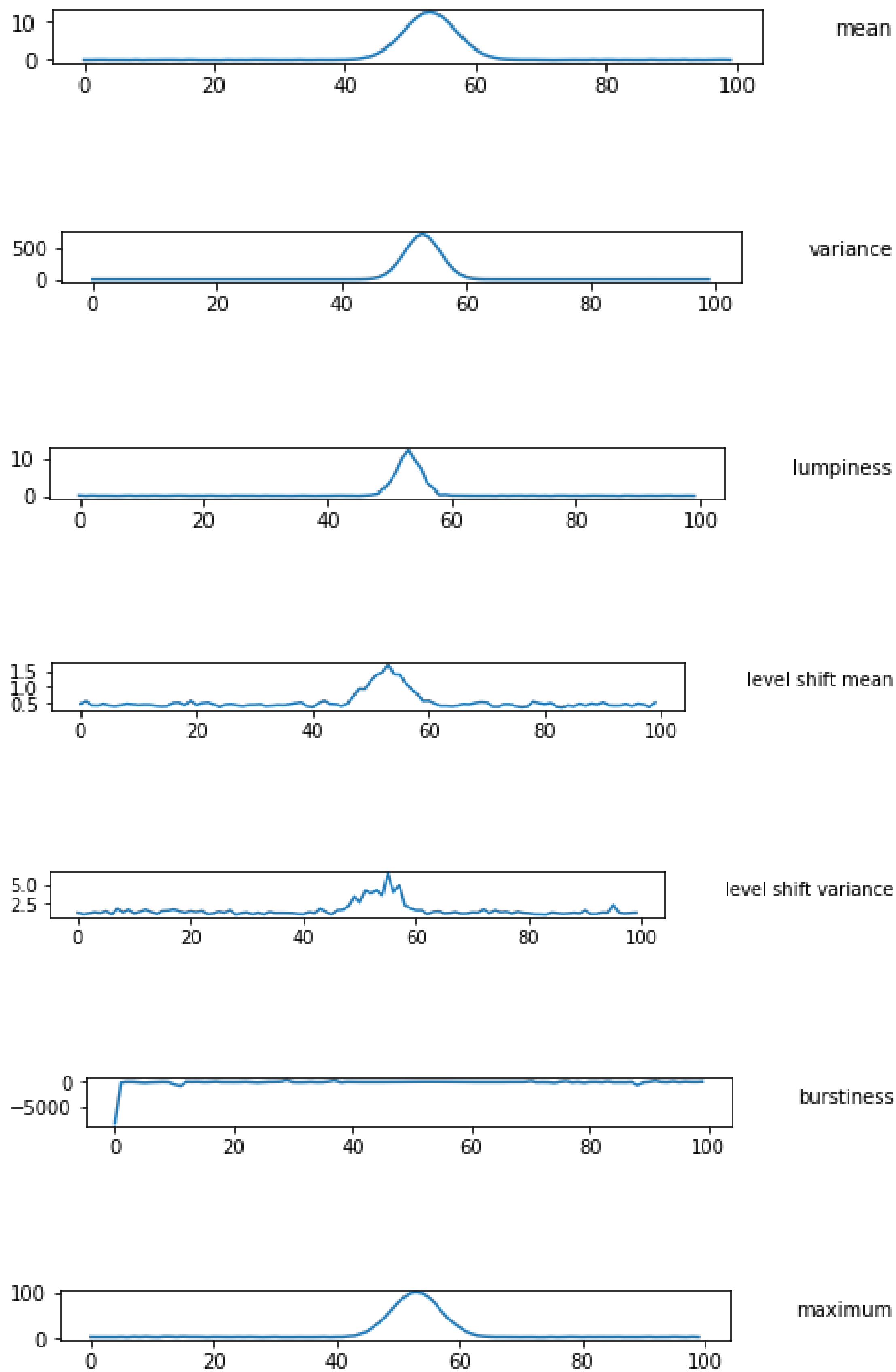
```
nepoch = 1000
dat = np.random.randn(nepoch*nts).reshape(nepoch,nts)
dat_train = np.array(dat)
for i in range(nts):
 amp_ts   = amp_ts_max * np.exp(-0.5*((i*1. - mean_ts)/sd_ts)**2)
 dat[:,i] = dat[:,i] + amp_ts* np.exp(-0.5*((np.arange(nepoch) - mean_epoch)/sd_epoch)
```

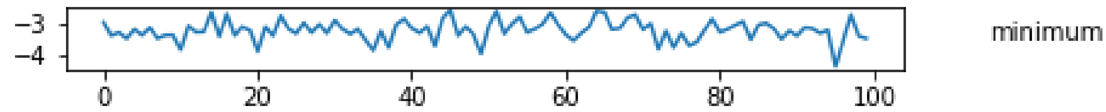
```
plt.clf()
fig = plt.figure()
ax1 = fig.add_subplot(111)
a = ax1.imshow(dat.T,aspect = 'auto')
plt.colorbar(a,label='Time series value')
ax1.set_xlabel('Time')
ax1.set_ylabel('Time series ID')
plt.show()
```

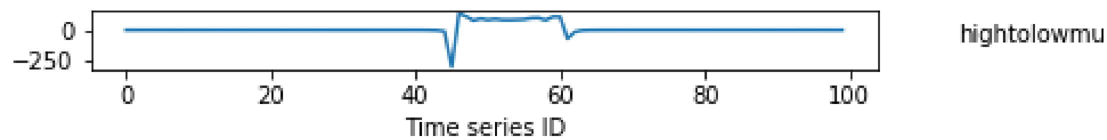
```
id_out = orej.outlier_smooth([dat_train,dat],diagnostic_figure='show')
```

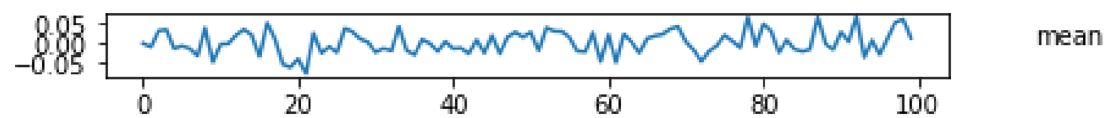
```
<Figure size 432x288 with 0 Axes>
```

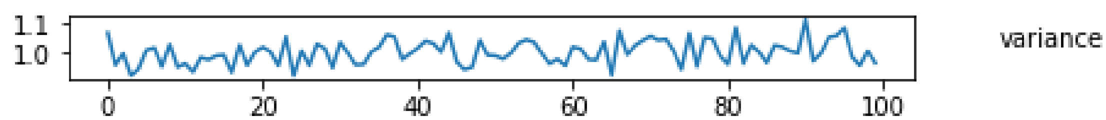


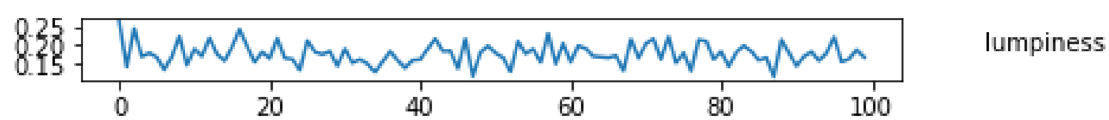

mean
variance
lumpiness
level shift mean
level shift variance
burstiness
maximum

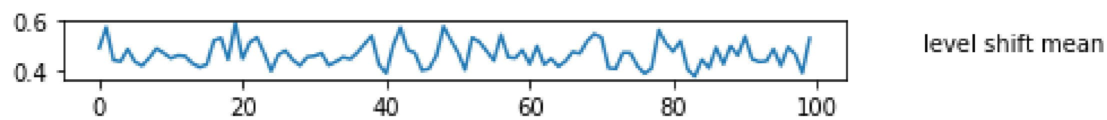

−3
−4
0
20
40
60
80
100
minimum

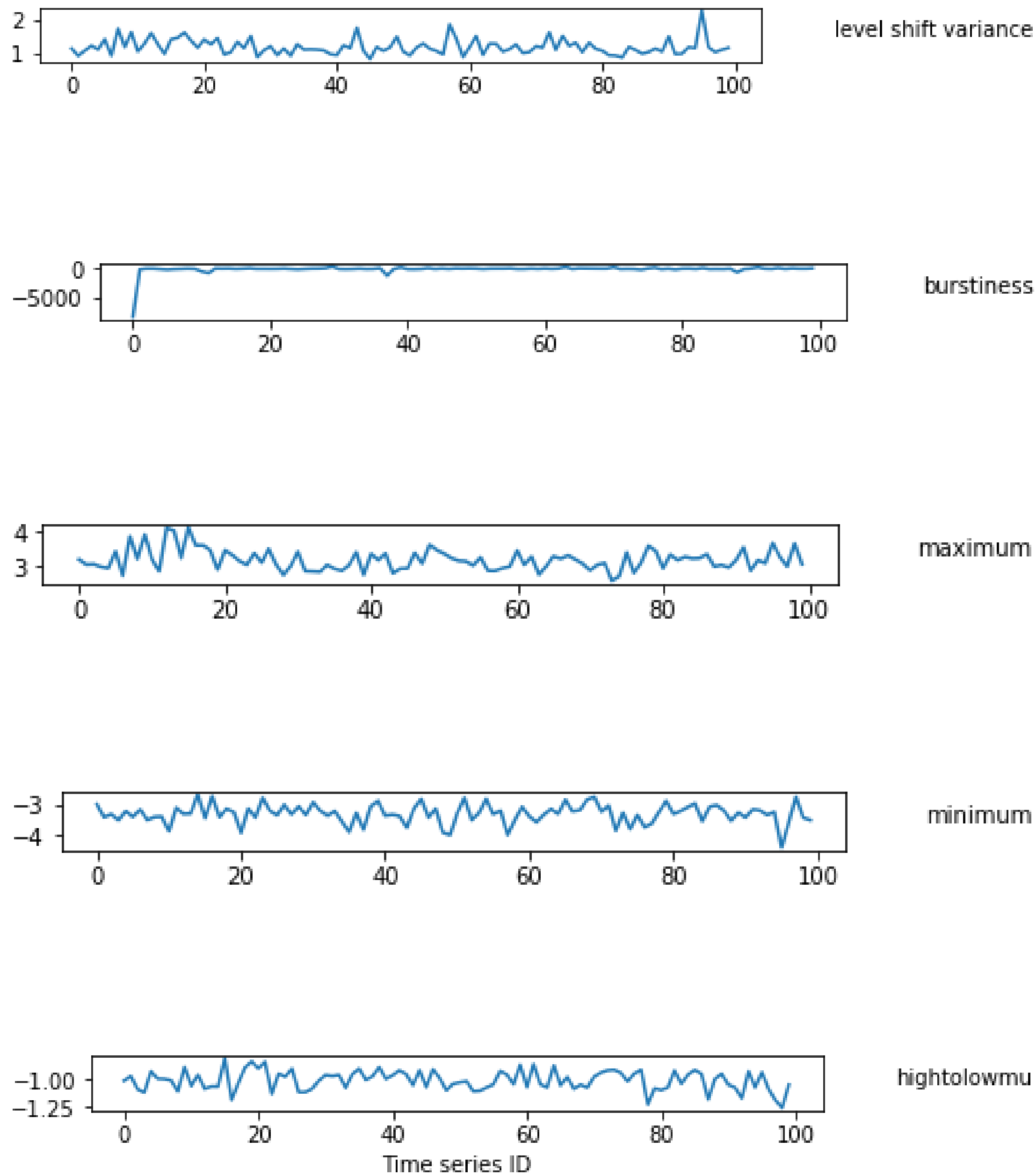

0
−250
0
20
40
60
80
100
Time series ID
hightolowmu

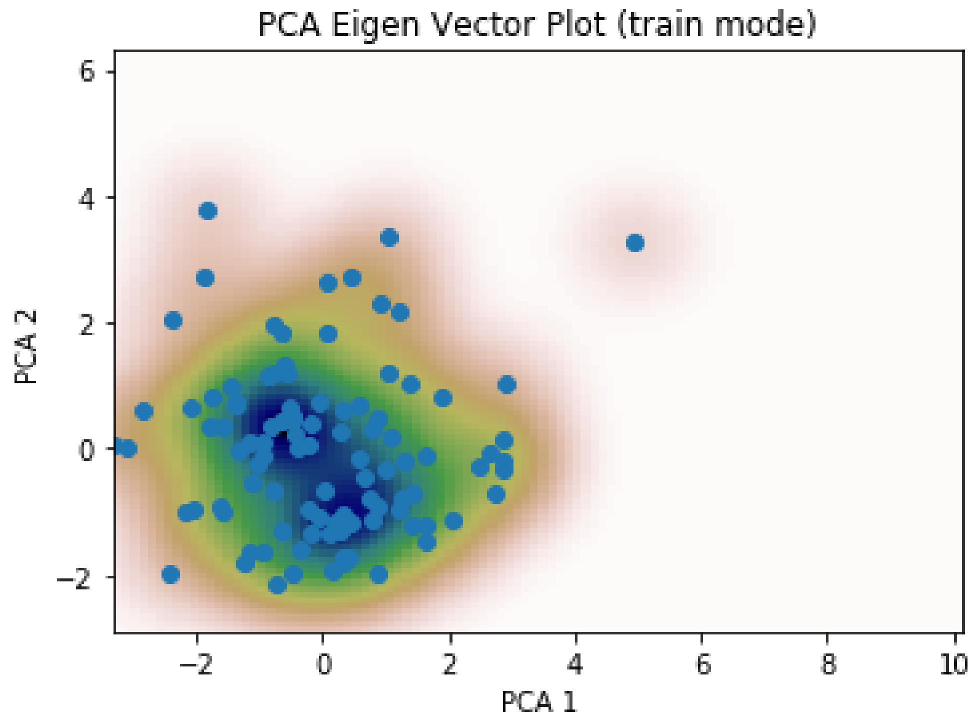

0.05
0.00
−0.05
0
20
40
60
80
100
mean

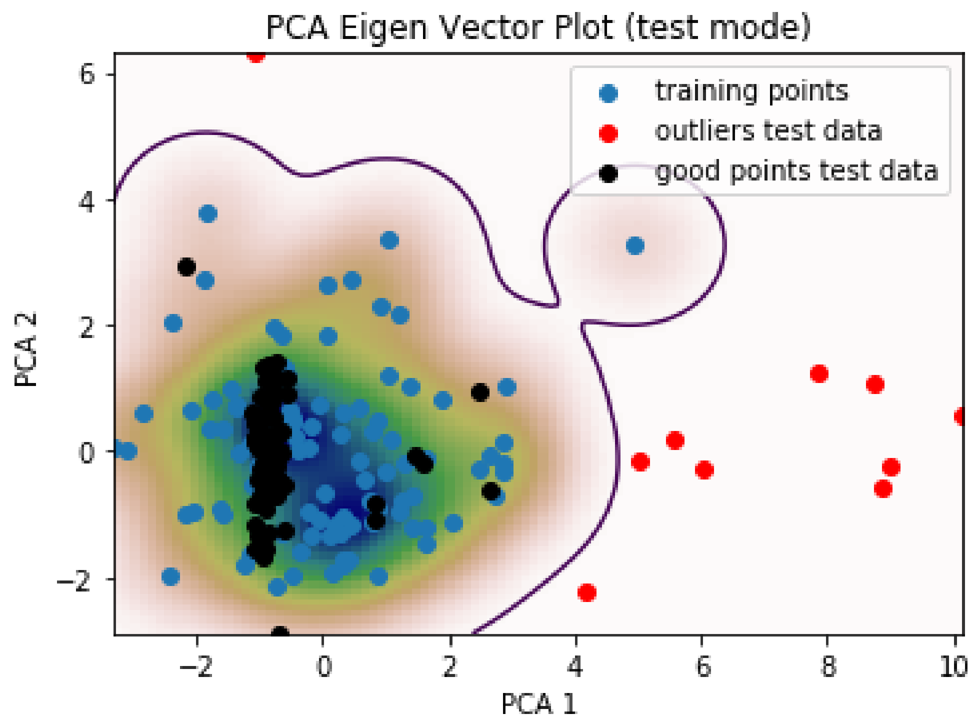

1.1
1.0
0
20
40
60
80
100
variance

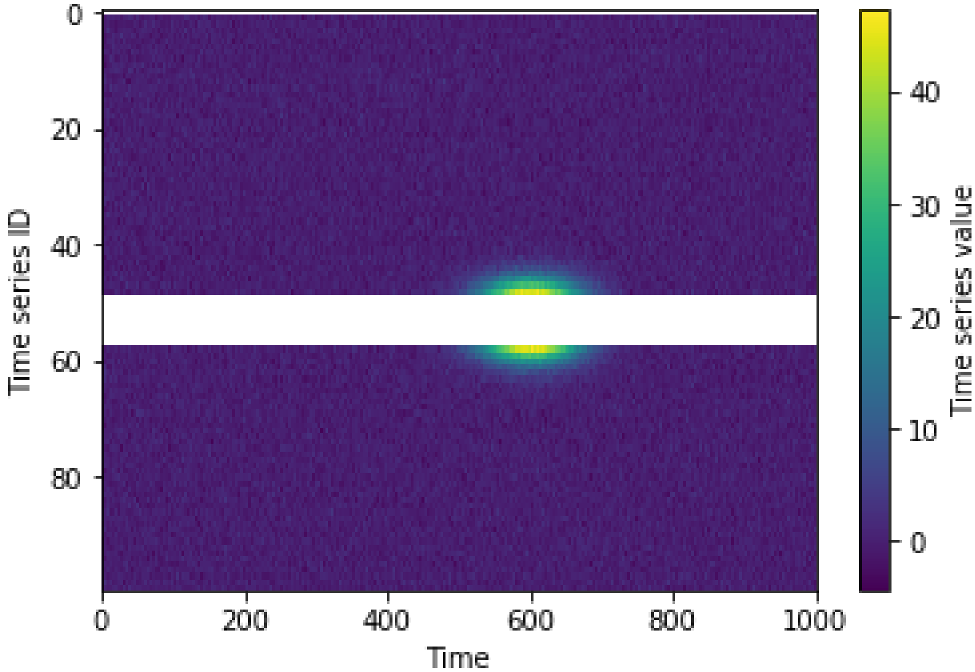

0.25
0.20
0.15
0
20
40
60
80
100
lumpiness


0.6
0.4
0
20
40
60
80
100
level shift mean


level shift variance
burstiness
maximum
minimum
hightolowmu
Time series ID


PCA Eigen Vector Plot (train mode)
PCA 2
PCA 1
6
4
2
0
−2
−2
0
2
4
6
8
10


PCA Eigen Vector Plot (test mode)
training points
outliers test data
good points test data
PCA 2
PCA 1
6
4
2
0
−2
−2
0
2
4
6
8
10



The above figure shows that the feature space representation of the timeseries data helps to identify the outlying 'hump' but we still have a tail either side of the anomaly where the algorithm could use further refinement.

TEST THE NON STATIONARITY CODE EVD_EVOLVE!!!!!

6 Non-stationarity

It is relatively easy to update this model to allow for non-stationarity. We just define a window of length 'w', input the most recent w epochs of known normal behaviour (a training data set) and the most recent w epochs. Using KDE, if > 0.5 the new observsations lie outside the confidence region of the training data, we update the KDE contours as the new 'normal behaviour'.