Using Autocorrelation to identify a representative cycle in a larger dataset with Python

Loughborough University - Wolfson School of Engineering

Caterpillar Inc. – Research department

Tomás Apolónia

April 2022

Contents

Chapter 1

Introduction

1.1 Background and context to the problem

Finding useful information in vasts amount of data is as difficult as 'finding a needle in a haystack'. Using **automated** statistical tools to analyse these large datasets can provide valuable information to assist decision making.
Finding useful information in a vast amount of data can prove to be a difficult and overwhelming task - comparable to finding a needle in a haystack. Using automated statistical tools to analyse these large data sets can provide valuable information which can assist decision-making.
When a CAT machine finishes field testing, it has recorded over 100 channels of data for each test run. An example of a test run would be having the machine lift a bucket of sand and unload it onto a truck. This action is repeated multiple times by an operator throughout the test. Due to the repeated action, a pattern will consequently emerge in every data channel, which provides the basis of the report. If the operator did not repeat any actions, the script would not be applicable. Using the author's tool, engineers are now able to efficiently analyse large volumes of data.

1.1.1 Cycle

A snippet of time is extracted from the larger dataset. In the report's case, the extracted data is roughly 30-120 seconds in length - which is the duration of the repeated task. The extracted data will be referred to as a "CYCLE" for the duration of this report.

1.2 What sort of data is suitable?

Suitable data for the autocorrelation script can range from the strain force of a boom arm on an excavator to a week's worth of foot traffic on a busy high street at peak times. In an ideal scenario, you would have time series data with low noise and randomness but high repeatability. To improve the filtering and selection process referenced in 4.2, data that has 10+ cycles should be used. The tool is unaffected by the length of the dataset, but is highly affected by the total repeated cycles. Cycle count limitations will be talked about in more depth in section 4.3.

Dataset	**A**	**B**	**C**
Description	Displacement Tilt cylinder	Displacement Steering cylinder	Force Strain gauge steering cylinder
Length of dataset	20 mins	20 mins	20 mins

Figure 1.1: Datasets

Row nº	**Magnitude**	**Resulting Time stamp**
0	-1.243801	0/500 = 0
1	-1.103403	1/500 = 0.002
2	-0.9631792	2/500 = 0.004
3	-0.9158725	3/500 = 0.006
...	...	...
13750	-1.122155	13750/500 = 27.5

$$f = \frac{rev}{t}$$

Figure 1.2: Frequency table Dataset C, 500hz

The datasets in this report include three separate files of CAT Machine data with varying levels of magnitude, randomness, and noise. This can be seen in Figure 1.1.

1.2.1 Time series data vs non-time series data?

Time series data are sequential (ordered by timestamp) and have constant time intervals between data points. In other words, data collected by an event-based data collection system are not time series unless events happen deterministically at constant time intervals.
A rule of thumb to identify the received data is to check if the data comes with timestamps alongside magnitude values in each time series. If data only has one column but includes the frequency, it does not need to come with timestamps, as the timestamp from single column data can be calculated using the formula shown in Figure 1.2.

Chapter 2

Background literature review

2.1 How does Autocorrelation work?

Autocorrelation is matching a signal compared to a delayed version of itself. This delay between the original signal and the delay is called ‘lag’.
“The autocorrelation [17] function can be used for the following two purposes:

- To detect non-randomness in data.
- To identify an appropriate time series model if the data are not random.

Given measurements, Y1, Y2, ..., YN at time X1, X2, ..., XN, the lag k autocorrelation function is defined as
Although the time variable, X, is not used in the formula for Autocorrelation, the assumption is that the observations are equispaced.
Autocorrelation is a correlation coefficient. However, instead of a correlation between two different variables, the correlation is between two values of the same variable at times Xi and Xi+k.
In our case we use Autocorrelation to identify an appropriate time series model, hence we plot with many lags.” [4]

2.1.1 How does this apply?

The Autocorrelation coefficient has now been explained mathematically - but how can this be shown graphically? When looking at Figure 2.2 and starting on the Y axis, you can see the scale goes from -1 to 1. 1 equates to a perfect match, whilst -1 is an anti-phase cycle match. 0 means there is a 0% correlation. The X axis is the length of time of the dataset, but is not shown as a conventional measure of time (ie seconds or minutes). Instead, it is shown as rows on our dataset (as seen in the table shown in Figure 1.1). If the dataset is 10,000 datapoints long, the X axis Autocorrelation coefficient graph will therefore show 10,000 as the maximum X value. On the graph, peaks will be marked with red crosses. These are what the script looks for when identifying representative cycles.

$$r_k = \frac{\sum_{i=1}^{N-k}(Y_i-\bar{Y})(Y_{i+k}-\bar{Y})}{\sum_{i=1}^{N}(Y_i-\bar{Y})^2}$$

Figure 2.1: Autocorrelation Formula

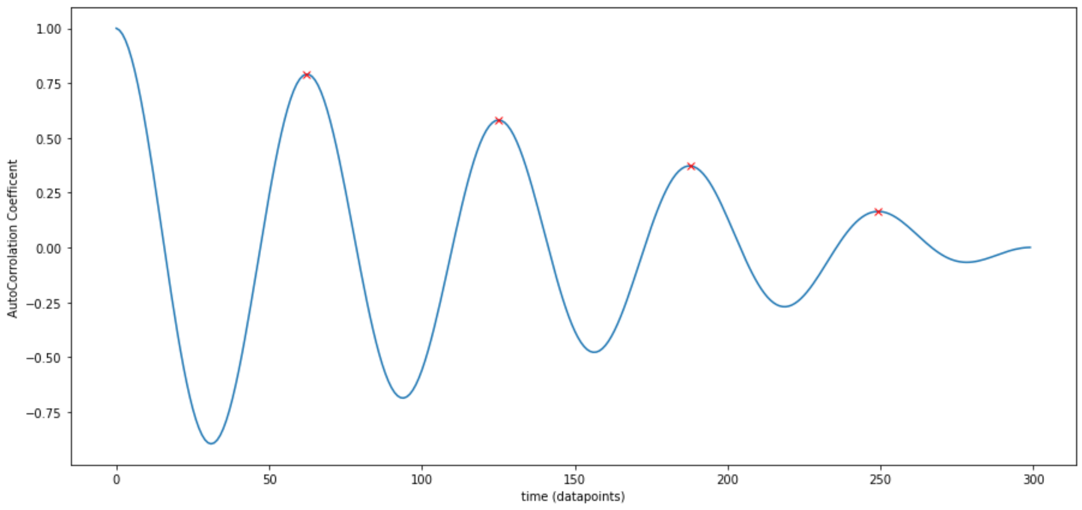


Figure 2.2: ACF tutorial

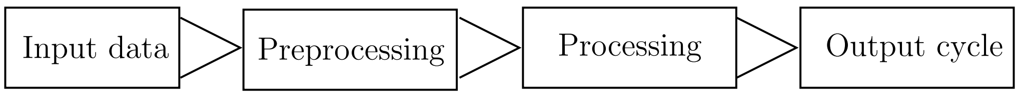


Figure 2.3

2.2 Other methods of pattern recognition

All methods and approaches of pattern recognition need to follow the approach shown in Figure 2.3. Using the database produced by Julien Siebert, Janek Groß, and Christof Schro [6], a table was made for time series data; this allows users to filter any information they need for the required task. In our case, we need to filter by pattern recognition. However, there are other classification methods available such as forecasting, classification, clustering, anomaly detection etc. Below is a list of all the libraries that fit the report's needs for pattern recognition.

2.2.1 Stumpy

On 17 Jul 2019, the Stumpy library was created. This library was born from the essential research created by the University of New Mexico, Riverside and the University of California. It "described an exact method called STOMP for computing the matrix profile for any time series with a computational complexity of O(n2)!" [13]

Matrix profile

The STUMPY library efficiently computes a matrix profile, a vector that stores the z-normalized Euclidean distance in between subsequences in a time series and its nearest neighbour. Target market is academics, data scientists, and developers. Open-sourced STUMPY, is a strong and scalable library that efficiently computes the matrixprofile according to this published study [13]. If the reader wishes to dive deeper into the technicalities of z-normalized Euclidean distance, they can read the best paper award winner from ICDM 2017.[14].

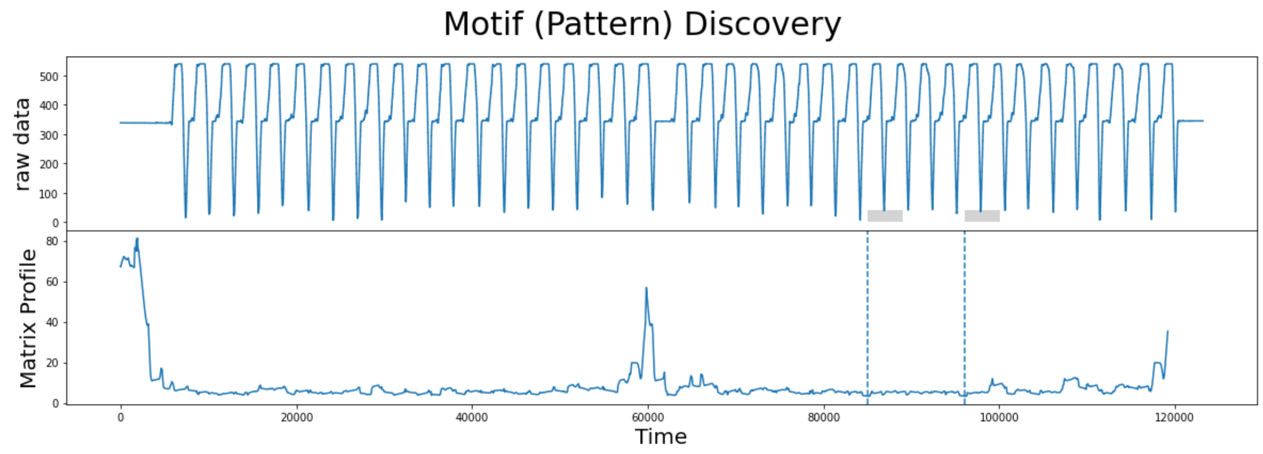


Figure 2.4: Motif (Pattern) Discovery

How can Stumpy be implemented for pattern recognition?

Utilizing dataset A with STUMP function, the graphs in Figure 2.4 were produced. The Y axis is the z-normalized Euclidean distance shown on the bottom plot. The higher the value, the lower the "correlation". This is illustrated by the best cycles selected, starting at the lowest points on the Matrix profile.
To automatically select the lowest point on the Matrix profile, a NumPy sorting algorithm is used to select the first point.

```
motif_idx = np.argsort(mp[:, 0])[0]
print(f"The motif is located at index {motif_idx}")
```

The motif is located at index 85062
According to the documentation [13], to select a complete cycle the user must select the lowest point in the Matrix profile, and then add the rough length of the pattern to create the start and endpoint of the cycle.
Requiring the knowledge of the cycle length prior to running the program highlighted a major problem with stumpy. This problem means that the stumpy library cannot be used if the engineer does not know the cycle length they are after.

```
length = 4000
mp = stumpy.stump(datasetA['mm'], length)
```

2.2.2 pyts

"pyts" is a Python library specialized in classifying time series. It seeks to make time series classification more accessible by offering pre-processing and utility tools, as well as implementations of multiple time series classification algorithms. The package includes several unit tests, and continuous integration ensures backward compatibility and code integration. The package is released under the BSD licence with three clauses. [8]
Unfortunately, after researching and testing the tool to see its application, it has been discovered that it does not fit the requirements. The 'pyst' library uses machine learning that requires training samples before processing the actual time series data, and under the methodology of the report, no training samples are provided. The function also has to find 'a representative cycle' with only 1 dataset,

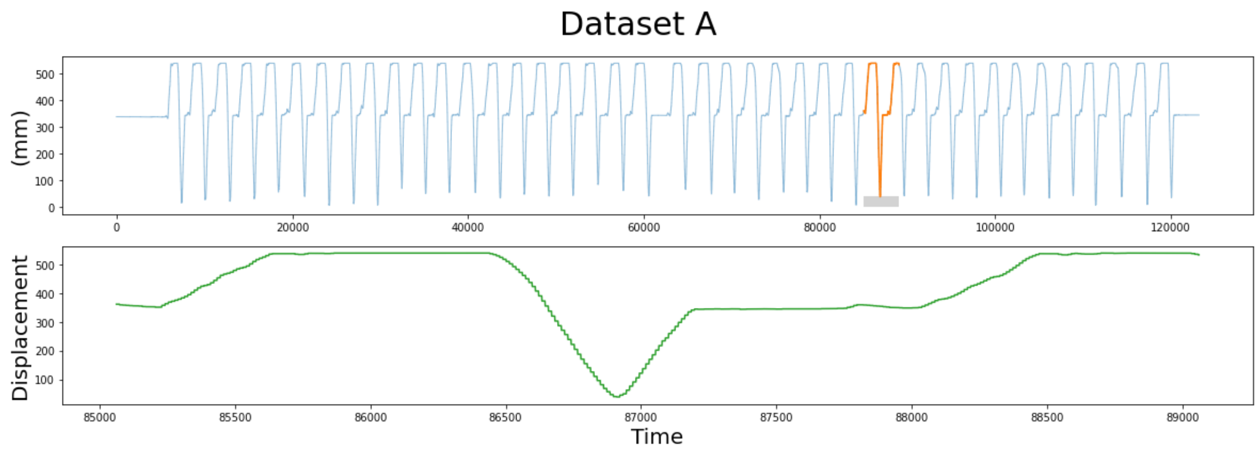


Figure 2.5: Motif (Pattern) Discovery

and even though it markets itself as pattern recognition (confirmed in database paper [6]), the'pyst' library fails to do so.

2.2.3 MatrixProfile

Here is another library based on the z-normalised Euclidean distance papers [14]. MatrixProfile is a Python 3 library produced explicitly by the MatrixProfile Foundation for timeseries doing data mining. The MatrixProfile is a novel data structure with accompanying algorithms (regimes, stomp, motifs etc.) created by the research groups of Keogh and Mueen at UC-Riverside and the University of New Mexico. The objective of MatrixProfile is to make these algorithms accessible to both beginners and experts. This can be done by standardising basic principles and good documentation [7].
Unfortunately, at the time this report was taken, Python 3.10 was not supported. This stopped the author from using the CAT machine datasets for this library. However, article [7] provides Figure 2.6 with the results of a taxi dataset conducted in New York City. These results seem promising for the report, and have an interesting automatic Discord identifier. In this context, Discord is defined as "A time series discord indicates a subsequence with the maximum distance to its neighbor in the given time series data, which means abnormal or unusual data trends."[15]

2.2.4 tslearn

"Tslearn" is another library based on the Scikit-learn platform, and is a trendy Python package for machine learning. Unfortunately, this library has the same issue as MatrixProfile; it relies on training data for the AI to practice before running through the main dataset. No training data is allowed in the report's case, rendering the "tslearn" library unsuitable.

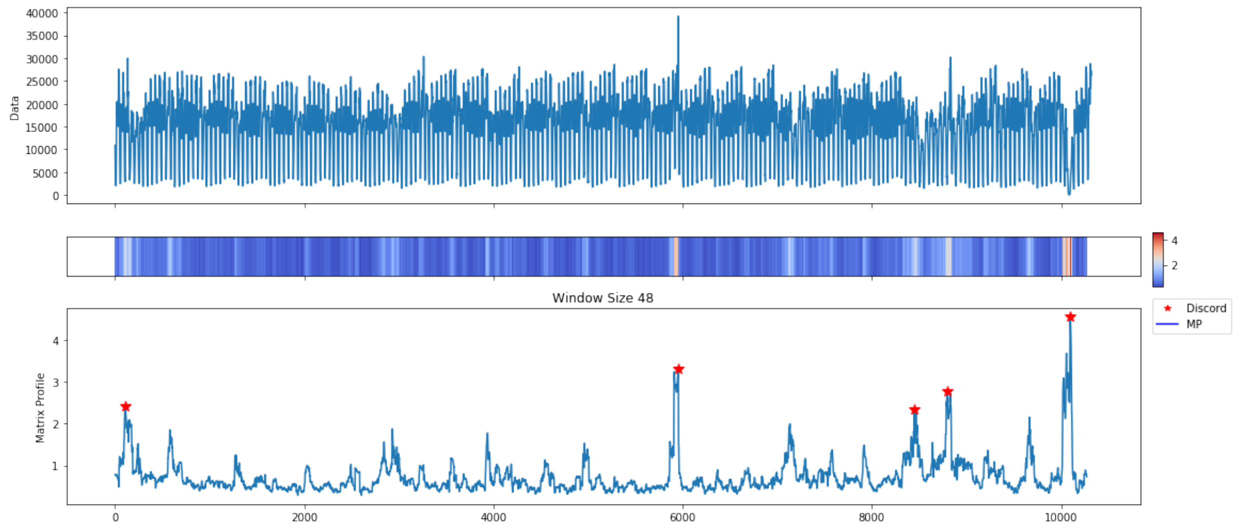


Figure 2.6: matrixTable

Chapter 3

Methodology

3.1 Software Considerations

In the process of software selection, multiple evaluations need to take place: access to licensing, software features, flexibility, pricing, and the time required to have a proficient understanding of the software.

Matlab

Matlab is the default company pick for a mathematical and graphical tool. However, due to licensing issues and multiple versions being available, the process was difficult and very time-consuming. The prominent issue is the presence of a very steep learning curve - this requires several hours of work, and the benefits given are not proportional to the amount of time it would take to become proficient in this software.

Python

Python was also closely considered for the mathematical and graphical tool. Here, the learning curve was more manageable, and due to the extensive range of tutorials and online documentation, Python became the program of choice. Another crucial selling point was the importing of libraries using “statsmodel”. This allowed for the integration of Autocorrelation and other statistical features without having to manually code them yourself. Due to all these benefits, the user could quickly get their desired outcome.

Diadem

The last software in consideration was Diadem, which was used to help engineers accelerate the post-processing of measurement data. Diadem proved to be an excellent choice as it was designed for large datasets. The only reason Python remains the software of choice is its superior flexibility and open-source which allows this skill to be applied in other areas.

3.2 Libraries used in final python script

Matplotlib

"Matplotlib" is a comprehensive library used for creating static, animated, and interactive visualisations in Python. "Matplotlib makes easy things easy and hard things possible." (2) This library was originally produced by John D. Hunder, an American microbiologist, to provide a MATLAB-like interface. This library is the backbone of the author's script. [2]
This library was originally produced by John D. Hunder American microbiologist to provide a a MATLAB-like interface. This library is the backbone of the authors script. All data visualisation graphs are produced using this library. For reference code refers to matplotlib as plt in script.

NumPy

NumPy is another famous open source library created in 2006 which was used in various fields of STEM, from engineering to science and mathematics."It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems." [12]
Used by beginner coders to advanced engineers, the API is used in depth with most data science and specific Python packages. Most importantly, for the autocorrelation script, Matplotlib and Pandas library also use Numpy.
The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
For this report, NumPy is used to create NumPy arrays, mathematical function sum [12], average and functions. These Numpy arrays are used instead of standard Python lists as they perform better in speed and are more compact.

Pandas

"Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more" [9]
Pandas in the code are referred to as the standardised name of pd. The sole purpose of this library in the report is to import and read the excel data using "pd.read_csv".

Scipy

SciPy (pronounced "Sigh Pie") is a library used strictly for up-sampling or down-sampling of the csv data. Example:

```
signal = scipy.signal.resample(signal,123200)
```

SciPy is an mathematics, science, and engineering software that is open-source. It includes modules for integration, linear algebra, statistics, ODE solvers, Fourier transforms, signal and image processing, and more. [10].
In this script it is only used to assist in down sampling the signal data.

Statsmodels

Statsmodels is a Python module that includes functions inside classes for a wide variety of statistical models, in addition to the execution of statistical tests and the study of statistical data. For each estimator, a full list of result statistics is accessible. In order to validate the accuracy of the findings, the results are compared to various pre-existing statistical analysis programs.[13] Statsmodel library serves as the foundation for this technical report because it can calculate the mathematical formulas and statistics for autocorrelation in a relatively small number of lines of code.

3.3 Script structure

Below is a high level breakdown of the autocorrelation script produced by the author. The diagram 3.1 is in the style of a flow chart with zero code. Zero code is used to simplify this process for those who are not familiar with Python.

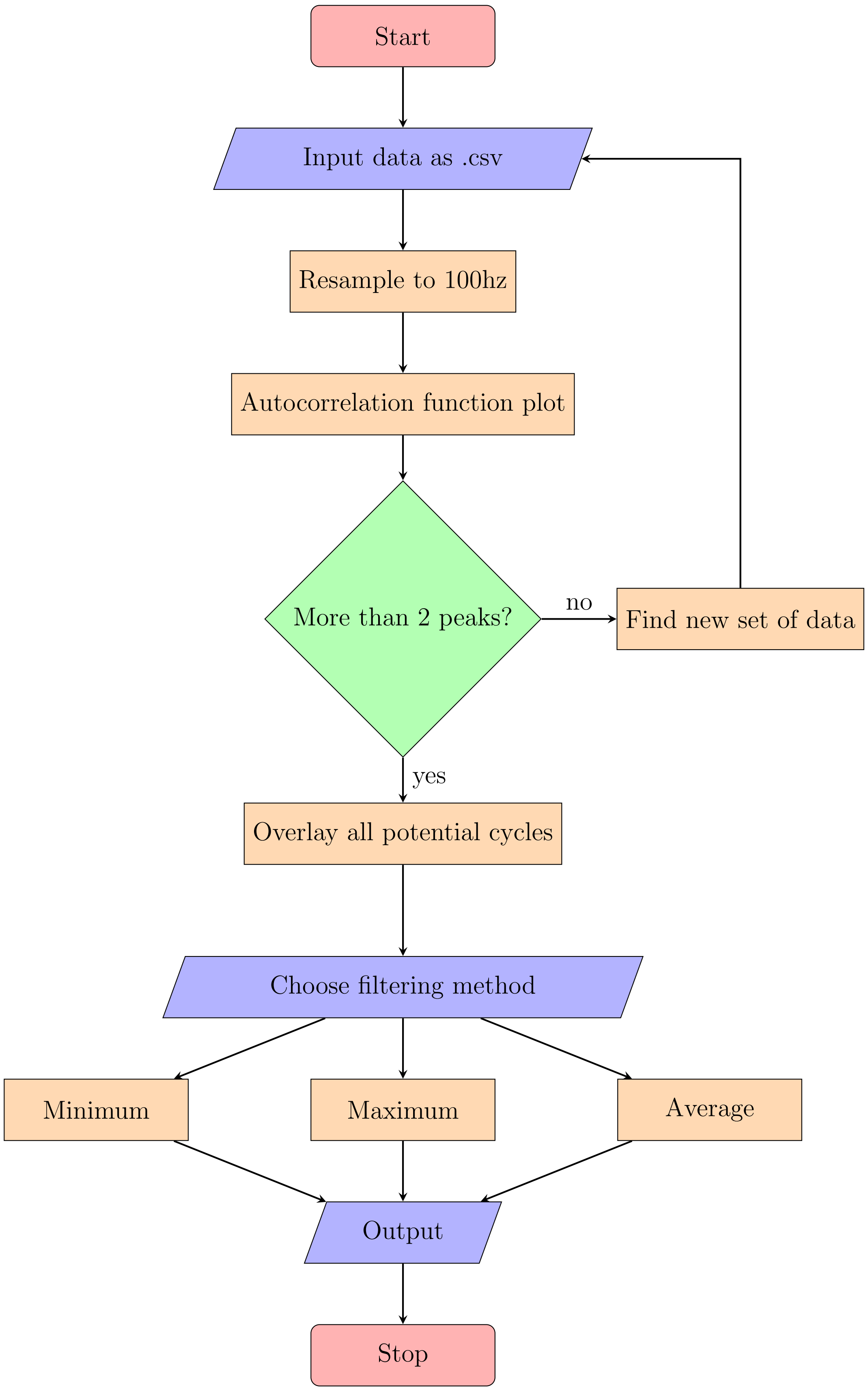


Figure 3.1: Flowchart

Chapter 4

Results and Analysis

4.1 Autocorrelation implementation

The Author's code starts here. How do we implement Autocorrelation using statsmodel library? Firstly import acf function from statsmodel library insuring it is previously downloaded onto the computer [3]. Code and print results seen on Figure 4.1 explain how to import the library as well as calling the function with our dataset (variable 'signal').

```
from statsmodels.tsa.stattools import acf
#Autocorrelate the signal and plot
#signal is the variable with all the raw data
acorr = acf(signal, nlags=(len(signal)))
plt.plot(acorr)
```

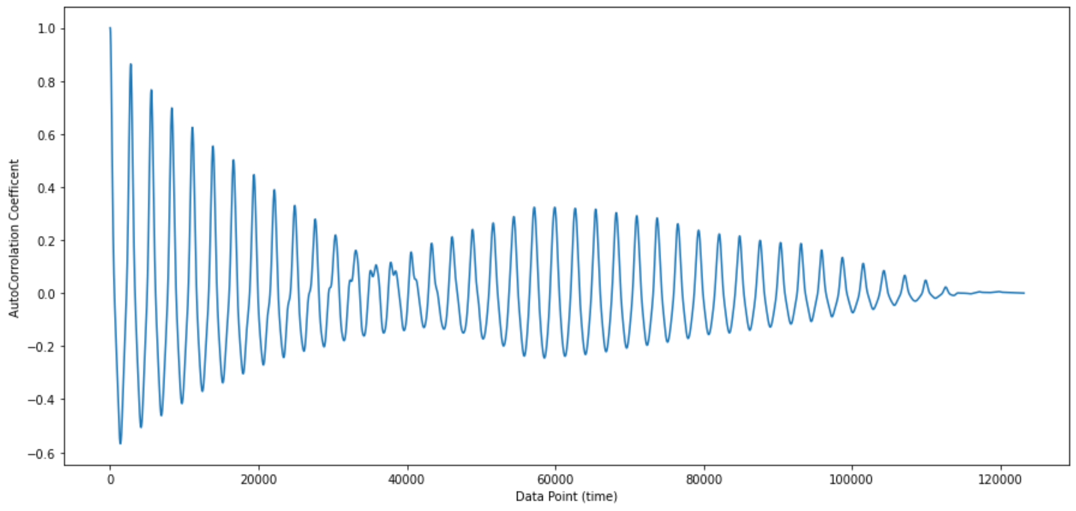


Figure 4.1: Output

The number of lags is equal to the total number of datapoints in plot 4.1. Every peak is a point where the function found a correlation i.e pattern. This explains why at datapoint zero the coefficient of autocorrelation is 1 (100%) as the script has not had enough time to shift itself by n lags. The next step is to extract the sections that have relatively high autocorrelation coefficient. This is done by recording all datapoints 'peaks'.The process of peaks in the next step can create limitations we will talk about in section 4.3.

```python
peak_points = scipy.signal.find_peaks(acorr, height=0.1,prominence=
                                      0.2)
plt.ylabel('AutoCorrolation Coefficent')
plt.xlabel('time (datapoints)')
plt.plot(acorr)
plt.plot(peak_points[0],peak_points[1]['peak_heights'],'rx')
#rx = crosses
```

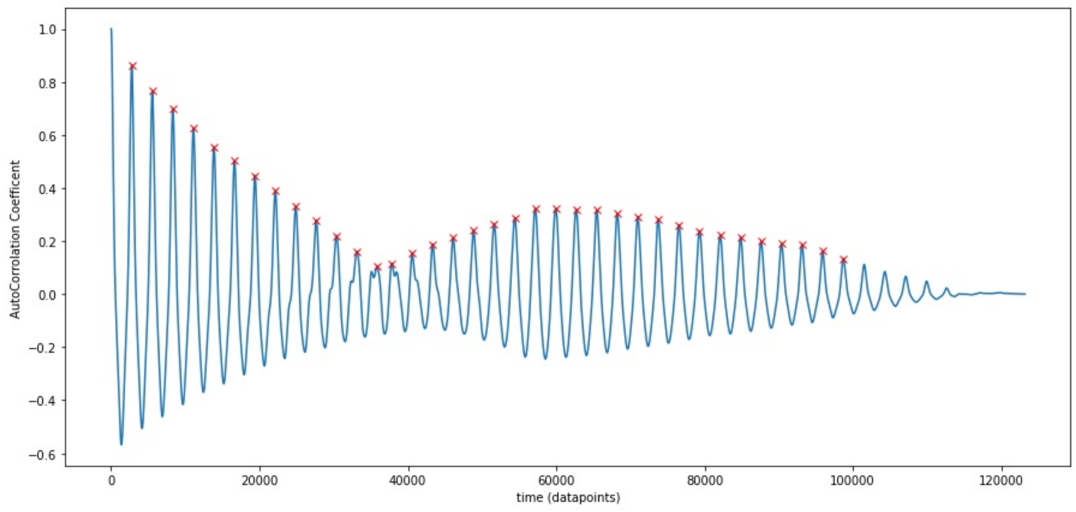
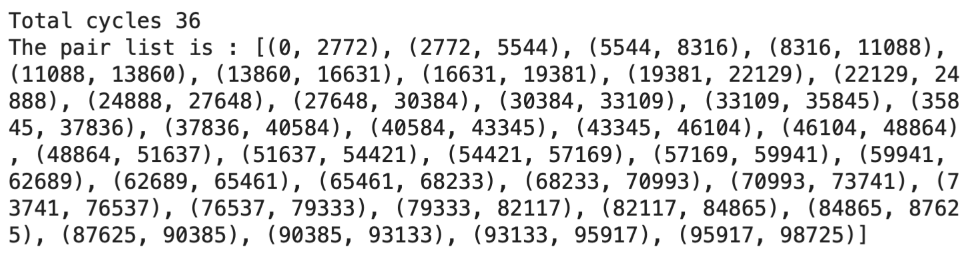


Figure 4.2: Output

“peak_points” array produces a list where in the 123,200 total datapoints the correlation is taking place. In other words, each value in ‘peaks’ array represents a pattern. The next step is to join 2 adjacent peaks to create a start and end of a cycle. This was done using python’s built in zip() function. The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.[5]. Code in 4.3 sampled from [11].

```python
"""Creating the timestamps from the peaks"""
#using zip() + list slicing
#to perform pair iteration in list
timestamps = list(zip(peaks, peaks[1:] + peaks[:1]))
print("Total cycles",len(timestamps))
# printing result
print ("The pair list is : " + str(timestamps))
```



```
Total cycles 36
The pair list is : [(0, 2772), (2772, 5544), (5544, 8316), (8316, 11088),
(11088, 13860), (13860, 16631), (16631, 19381), (19381, 22129), (22129, 24
888), (24888, 27648), (27648, 30384), (30384, 33109), (33109, 35845), (358
45, 37836), (37836, 40584), (40584, 43345), (43345, 46104), (46104, 48864)
, (48864, 51637), (51637, 54421), (54421, 57169), (57169, 59941), (59941,
62689), (62689, 65461), (65461, 68233), (68233, 70993), (70993, 73741), (7
3741, 76537), (76537, 79333), (79333, 82117), (82117, 84865), (84865, 8762
5), (87625, 90385), (90385, 93133), (93133, 95917), (95917, 98725)]
```

Figure 4.3: Paired List

As you can see in figure 4.3, we have converted the 37 ‘crosses’ into 36 cycles with start and end timestamps. List ‘timestamps’ plotted shown on fig 4.4.

4.2 Filtering

Back to the initial objective, “Identify a Representative cycle within the larger data set”. Next step is deciding which cycle we should pick from the 36 cycles identified.

```
for cycle in cycles_plot:
    plt.plot(cycle)
```

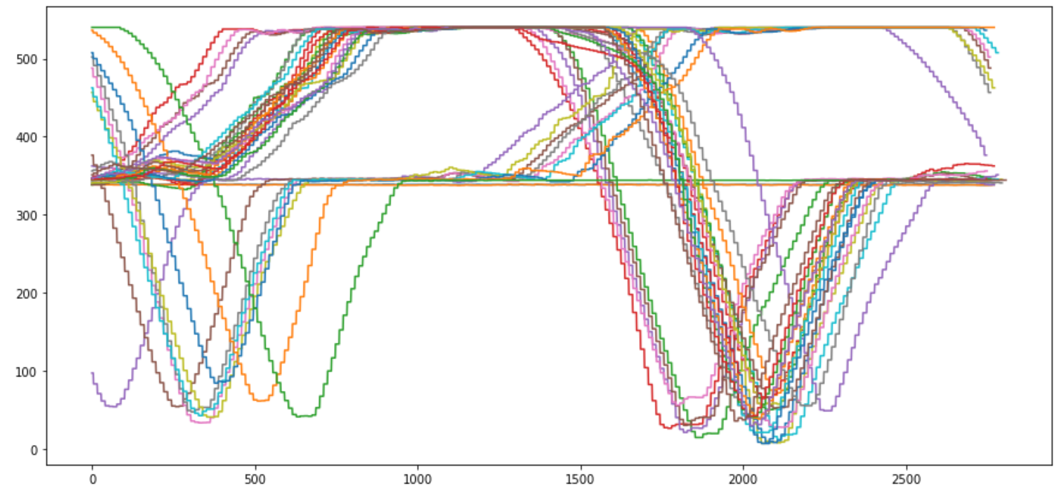


Figure 4.4: OverlayedCycles

4.2.1 Initial observations:

- Not all the cycles align.
- All the cycles seem to follow the large dip at time 2000.
- This dataset seems to have a cap at magnitude of roughly 500.
- Autocorrolation did a good job with this set of data.
- Some cycles seem longer than others.
- The cycles don’t all start at the same magnitude
- Visually the cycles are very similar.

4.2.2 Maximum peak

Approach A: filter with maximum peak, lowest trough, largest range
This approach would give the most extreme cycles. While it is a ”...Representative cycle...”. It isn’t the closest to the ”average”

```
listforMax=[]

for i in cycles_plot:
    listforMax.append(max(i))

maxIndex = listforMax.index(max(listforMax))
```

The 'for loop' finds the maximum value for each cycle in cycles_plot then adds it to 'listforMax'. Once the loop finishes, it has 36 values, one maximum value for each cycle. If we want to select the cycle with the largest value, variable 'maxIndex', holds the index value of this cycle. In our case it is index nº 4.

4.2.3 Lowest trough

```
listforMin=[]

for i in cycles_plot:
    listforMin.append(min(i))

minIndex = listforMin.index(min(listforMin))
```

The 'for loop' finds the minimum value for each cycle in 'cycles_plot' and then adds it to 'listforMin'. Once the loop finishes it has 36 values, one minimum value for each cycle. If we want to select the cycle with the lowest value, variable 'minIndex' holds the index value of this cycle. In our case it is index nº 30.

4.2.4 Average

After using the mathematical average approach to find the average peak of all the magnitudes of each cycle, we must find the mean of the 36 values. For the purposes of comprehension, assume it gives us a value of 340.2, and then use Python built in method index() to find the cycle number. Unfortunately the mean or average value calculated might not exist in the actual data, giving the script an error. Instead we have to alter this approach and find the closest real value to the theoretical mean. This is an approach:

```
meanList=[]
for i in cycles_plot:

    meanList.append(np.sum(i)/len(i))
```

Here we have created a list with 36 values, the overall mean magnitude per cycle, giving the length of this list equal to the total number of cycles found from the Autocorrolation function. Next is finding the index and closest real value.

```
closest_real_value = min(meanList, key=lambda x:abs(x-np.average(
                                    meanList)))
closest_to_av_index = meanList.index(closest_real_value)

print(closest_to_av_index)
```

Using Python's built in min function, it finds the minimum distance from the ´meanList' to the real value "np.average(meanList)". [16]

The ‘meanList.Index()’ function finds the index of our ´closest_real_value’ Giving an output of cycle nº 13.

4.2.5 Filtering limitations

Average, minimum and maximum work in the authors case but everyone’s use case is different. Exploring other options like Standard deviations, most common cycle, 25 quarterly could be explored.

4.3 Autocorrelation limitation

When the signal data has a greater Randomness the magnitude of the Autocorrelation coefficient is reduced drastically.
The lower the amount of cycles found, the higher the probability that the cycle selected is not “...Representative...”.

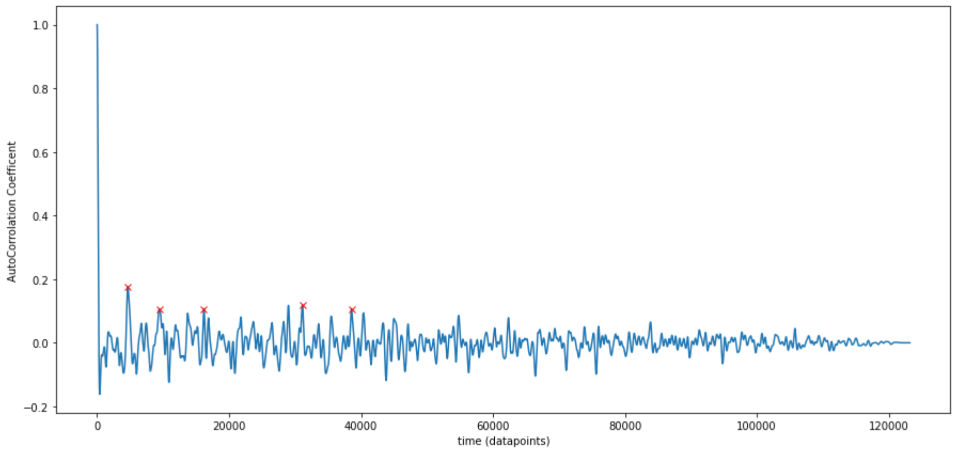


Figure 4.5: Limit Testing A

Stress testing ’acf’ function with dataset B, the displacement steering cylinder dataset has a higher randomness compared to the dataset A used in the rest of the report. This translates to a decrease in total cycles found. The amount of cycles identified went from 36 to 5, 86% decrease. This can be justified as the steering cylinder has a lot of feedback from the machine’s steer over rocks and uneven terrain unlike the lift or tilt cylinder.
The heaviest stress testing was with dataset C, ref 4.6.This data came from a strain gauge force dataset attached to a CAT machine. Here the function failed to correlate a single Dataset. Factors for this result could be 500hz sensor data causing too much noise. Forces by nature are going to be harder to correlate compared to displacement.

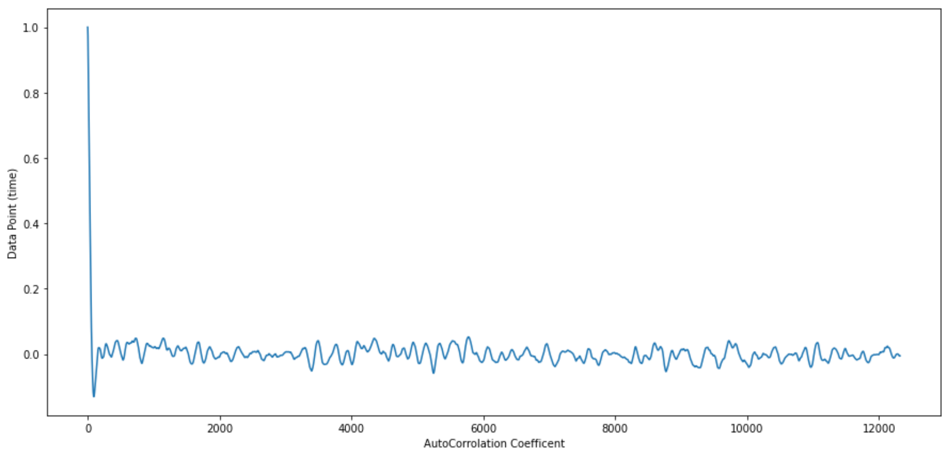


Figure 4.6: Limit Testing B

Chapter 5

Conclusions

5.1 Technical/Commercial Content

In this day and age, we are receiving massive amounts of data. Unfortunately, raw data is meaningless without the skills and tools to analyse it. The commercial benefits of autocorrelation are that it saves the company resources by reducing the time needed to filter through hours of machine data manually. As the engineers only have to analyse one cycle instead of hundreds, they can identify micro details in a cycle which would have otherwise drowned in hours of data. Data science tools are increasingly popular, especially within engineering teams to help assist in large decisions. The companies using data science tools range from financial services to AMG Mercedes F1 team in solving their ‘porpoising’ issue [1].

5.1.1 Reducing cost per dataset

Filtering this much data takes on average 1-2 days. The average engineer salary in the United Kingdom is £40,000 per year, or £20 per hour. If an engineer is on a salary of £20 an hour, this amount of work would cost the company £300 per dataset (7.5 hours x 20 x 2). Using the autocorrelation tool, the time taken to find the representative cycle would decrease from 2 days to 1 hour, and would cost £20 per dataset instead. This would mean a £280 decrease for the company per engineer.

5.1.2 Key findings

The script produced with the Statsmodels ACF function shows promising results for a method of cycle identification in python. Successfully working with 2 out of the 3 datasets tested against it and only failing with dataset C, this failure could be attributed to the 500hz frequency and high noise related to strain gauges. Filtering was another hurdle in producing a fully functioning tool using autocorrelation. Three methods were produced with the most complex one being the ‘mean’ function. Despite the effort that went into producing them, they are a weak link in the report. More advanced filtering methods could be investigated when narrowing down the group of cycles to one representative cycle.

5.1.3 Recommendations

For cycle identification the findings suggests that the reader try out both Statsmodel and Stumpy to see which method fits their specific needs better.

Autocorrelation

To carry out pattern recognition, you need the cycle length. However, if this is not available, the autocorrelation tool can be used instead. If large changes need to be made to the original script, the author recommends extracting the acf function from Statsmodel library instead of editing the whole script. An idea that could be investigated to improve the tools performance with higher randomness data like Dataset C would be lowering the height constant and changing the prominence value in the peaks function.

```
#Find peak points
peak_points = scipy.signal.find_peaks(acorr, height=0.1,prominence=
                                      0.2)
```

Stumpy

There are two choices for pattern recognition: the autocorrelation tool or Stumpy library (which is shown in the literature review 2.2.1). If the length of the rough cycle is known, it is recommended to use Stumpy - otherwise, the autocorrelation tool should be used.
If the length of a cycle is know, STUMP function would even work with dataset C at 500hz, which the autocorrelation script failed in.
If Stumpy is chosen and the reader is having issues with processing time required, Stumpy library offers an accelerated version through a Nvidia GPU version of the STUMP function. Due to limited access to a specific brand GPU, the author was unable to test this version.

```
import stumpy
mp = stumpy.gpu_stump(df['value'], m=m)
# Note that you'll need a properly configured NVIDIA GPU for this
```

Due to the open source nature of the library, other tutorials and examples can be updated at any moment, so it is recommended to check the Github page and the documentation at regular intervals [13]

Appendix A

Complete code

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf

#Autocorrelate the signal and plot
plt.figure(figsize=(15,7))
plt.ylabel('AutoCorrolation Coefficent')
plt.xlabel('Data Point (time)')
acorr = acf(signal, nlags=(len(signal)/10))
plt.plot(acorr)

file="STR_ABS_AX_F_COMB_HAND_BANK.CSV"

rawData = pd.read_csv (file, header=11)

signal = rawData["kN"].to_numpy()
signal = scipy.signal.resample(signal,123200)

plt.figure(figsize=(15,4))
plt.plot(signal)

#Autocorrelate the signal and plot
plt.figure(figsize=(15,7))
plt.ylabel('AutoCorrolation Coefficent')
plt.xlabel('Data Point (time)')
acorr = acf(signal, nlags=(len(signal)/10))
plt.plot(acorr)

#Find peak points
peak_points = scipy.signal.find_peaks(acorr, height=0.1,prominence=
                                  0.2)

# peak_points 4D array, 1D = timestamp, , 2D = peak_heights, 3D =
                                  left_based, 4D = right_based,

plt.figure(figsize=(15,7))
plt.ylabel('AutoCorrolation Coefficent')
plt.xlabel('time (datapoints)')
plt.plot(acorr)
plt.plot(peak_points[0],peak_points[1]['peak_heights'],'rx') #rx =
                                        crosses

peaks = np.append([0],peak_points[0])
len(peaks)

"""Creating the timestamps from the peaks"""
#using zip() + list slicing
#to perform pair iteration in list
timestamps = list(zip(peaks, peaks[1:] + peaks[:1]))
print("Total cycles",len(timestamps))
# printing result
print ("The pair list is : " + str(timestamps))

#Plot all cycles overlayed

cycles_plot= []
for a,b in zip(peaks[:-1],peaks[1:]):
    cycles_plot.append(signal[a:b])
cycles_plot # a 2D list, each element is 1 all the points in 1
                                        cycle
print(type(cycles_plot))
print("There's", len(cycles_plot), "cycles plotted" )

plt.figure(figsize=(15,7))
for cycle in cycles_plot:
    plt.plot(cycle)


listforMax=[]

for i in cycles_plot:
    listforMax.append(max(i))
maxIndex = listforMax.index(max(listforMax))

print("max value from list is",max(listforMax))
print("for function max, cycle index is",maxIndex)


listforMin=[]

for i in cycles_plot:
    listforMin.append(min(i))

minIndex = listforMin.index(min(listforMin))

print("min value from list is",min(listforMin))
print("for function max, cycle index is",minIndex)

meanList=[]
for i in cycles_plot:

    meanList.append(np.sum(i)/len(i))

closest_real_value = min(meanList, key=lambda x:abs(x-np.average(
                                        meanList)))
print(closest_real_value)
```

```
plt.plot(cycles_plot[13])
closest_to_av_index = meanList.index(closest_real_value)
print(closest_to_av_index)
```

Bibliography

[1] Autosport. *Mercedes still in "trial and error" mode with porpoising F1 updates*. 2022. URL: `https://www.autosport.com/f1/news/mercedes-still-in-trial-and-error-mode-on-porpoising-f1-updates/10295468/`.

[2] matplotlib. *Matplotlib: Visualization with Python*. 2022. URL: `https://matplotlib.org`.

[3] Josef Perktold et al. *statsmodels.tsa.stattools.acf*. 2022. URL: `https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.acf.html`.

[4] National Institute of Standards and Technology. *Autocorrelation*. 2022. URL: `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35c.html`.

[5] w3schools. *Python zip() Function*. 2022. URL: `https://www.w3schools.com/python/ref_func_zip.asp`.

[6] Julien Siebert, Janek Groß, and Christof Schroth. "A systematic review of Python packages for time series analysis". In: *CoRR* abs/2104.07406 (2021). arXiv: `2104.07406`. URL: `https://arxiv.org/abs/2104.07406`.

[7] Andrew Van Benschoten et al. "MPA: a novel cross-language API for time series analysis". In: *Journal of Open Source Software* 5.49 (2020), p. 2179. DOI: `10.21105/joss.02179`. URL: `https://doi.org/10.21105/joss.02179`.

[8] Johann Faouzi and Hicham Janati. "pyts: A Python Package for Time Series Classification". In: *Journal of Machine Learning Research* 21.46 (2020), pp. 1–6. URL: `http://jmlr.org/papers/v21/19-763.html`.

[9] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

[10] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[11] geeksforgeeks. *Python — Pair iteration in list*. 2019. URL: `https://www.geeksforgeeks.org/python-pair-iteration-in-list/`.

[12] Marten van Kerkwijk. *numpy*. 2019. URL: `https://github.com/numpy/numpy/blob/v1.22.0/numpy/core/fromnumeric.py#L2160-L2297`.

[13] Sean M. Law. "STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining". In: *The Journal of Open Source Software* 4.39 (2019), p. 1504.

[14] Yan Zhu et al. "Matrix profile VII: Time Series Chains: A new primitive for time series Data Mining (Best Student Paper Award)". In: *2017 IEEE International Conference on Data Mining (ICDM)* (2017). DOI: `https://www.cs.ucr.edu/%7Eeamonn/chains_ICDM.pdf`.

[15] Diane Myung-kyung Woodbridge et al. "Time series discord detection in medical data using a parallel relational database". In: *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (2015). DOI: `10.1109/bibm.2015.7359885`.

[16] kennytm. *From list of integers, get number closest to a given value*. 2013. URL: `https://stackoverflow.com/questions/12141150/from-list-of-integers-get-number-closest-to-a-given-value`.

[17] Jenkins et al. *Time Series Analysis: Forecasting and Control*. Revised Edition, 1976.