Imperial College London

BEng Individual Project

Department of Computing

Imperial College of Science, Technology and Medicine

# Exploring a New Time Series Prediction Algorithm

*Author:*
Samuel Brotherton

*Supervisor:*
Prof. William Knottenbelt
*Co-Supervisor:*
Ben Rogers
*Second Marker:*
Dr. Thomas Heinis

June 15, 2020

Submitted in partial fulfillment of the requirements for the BEng Computing of Imperial College London

## Abstract

The importance and implications of using past information to predict the future has shaped the global society in ways it would be difficult to overestimate. From Aristotle (384-322 BC) in his attempt to predict the weather, forecasting techniques have progressed from folklore and intuition towards methods founded on more rigorous models. Complex algorithms, requiring specialist training, huge computational resources, and vast quantities of data, dominate the forecasting market. Within this context, an inventor has created a novel Time Series forecasting algorithm, Alpex, following decades of work, that is characterised by zero configuration, constant space complexity and promises to not only be expeditious in terms of runtime but is capable of grasping complex classes of chaotic time series rapidly and accurately. Building from a single function console application, this project considers and answers the questions “How can we make Alpex accessible and deployable?”; and “For which classes of time series is Alpex most effective?”.

The first question is addressed by developing a robust and useable C++ API to stream data in and predictions out; we extend this to create Python wrappers, command line programs and a fully containerised, RESTful, micro-service Spring Boot web application. Providing the deployment infrastructure, results in a Time Series as a Service implementation which is ready to facilitate any eco-system. In developing a suite of Analysis tools, involving graphing functions and sample experiments, we empower data analysts in their investigation into Alpex.

In answering the second question, we conducted a thorough investigation into two types of time series: synthetic time series with a known ground truth, and real-world datasets with an unknown underlying pattern. In addressing the former, we find that on some classes of complex chaotic time series, Alpex is not only considerably faster at picking up on these patterns but also outperforms ARIMA in terms of accuracy. Investigating domains such as power consumption, renewable energy generation and changing water levels; we find that Alpex can detect underlying, periodic patterns on some real-world datasets, with accuracies that are competitive against (S)ARIMA. In terms of scalability, ease of operation and computational complexity Alpex has a clear and distinct advantage paving the way for further research into the algorithm.

## Acknowledgments

I would like to thank my supervisor and personal tutor, Prof. William Knottenbelt, who not only has supported me throughout this project, providing invaluable insight and knowledge around the subject matter, but has also offered guidance and encouragement throughout my time at Imperial College London.

I am also extremely grateful to Ben Rogers who, without his ingenious algorithm, this project would not have been possible. His constant enthusiasm, insight and contribution made this a successful, interesting learning experience.

Finally I would like to thank my family who have helped and supported me throughout my whole education.

# Contents

# Chapter 1

# Introduction

A Time Series is a series of data points, ordered by time, with data taken at equal intervals to shows how values change over a set time period. Time Series are everywhere. Anything that deals with measurable variation over time can be considered as a Time Series; from tracking a person's height over time, to energy consumption over a given period, to stock markets, to sales... It is hard to think of any industry or aspect of life that does not involve Time Series, even without realising it.

## 1.1 Motivation

As Time Series are so prevalent in our lives and in industry, being able to predict the output of them is one of the key problems facing Computer Scientists and Mathematicians. Time Series prediction is used everywhere, some examples are as follows:

- It is used to predict the stock market. In 2018, CNN reported[26] that between 50-60% of daily trades are executed automatically using predictions, which can increase to 90% on volatile days; and not least to mention algorithm assisted training which makes up much of the rest.

- It is used in cloud computing to predict workload of servers such that Software as a Service providers can offer high Quality of Service whilst keeping costs low[50].

- It is used in predicting the Wind Speed and Wind Power Density for wind Turbines, it is reported that [64] inaccuracies in Wind Power Density forecasts account for about a 10% loss in the total energy generated by wind power.

These are just a few examples of where Time Series forecasting can be used, however their implications are almost endless; therefore, it is of vital importance that we study them and try and find new and innovative means of predicting Time Series, so this is what we aim to study.

## 1.2 Problem Description

The aim of this project is to study a new and exciting Time Series Algorithm proposed by Ben Rogers; this algorithm takes an innovative approach to time series prediction which has never been seen before, providing itself with some distinct advantages over the competition.

However, like a rough diamond, this algorithm has many exceptional aspects, hidden away behind potential; it is these qualities that set it apart from the rest, without the right polish it is all just a concept. Through initial tests and reports, the algorithm shows the potential to revolutionise the way that we predict Chaotic Time Series. It is our job to refine this gem and unlock its potential.

Although the detailed implementation of the algorithm cannot be shared as they are protected under an NDA, we can discuss the properties that make this algorithm unique and the advantages and limitations these may incur.

It was seen, through limited experiments, that without much training Alpex can make pinpoint accurate predictions for classes of chaotic time series. One of the big problems with Chaotic Time Series is the quantity of data needed to start making predictions; this algorithm is able to learn these series with only a handful of data points.

The algorithm is extremely fast, to train and make accurate predictions, on 4000 data points can take just 528 milliseconds. Typical Artificial Intelligence systems require vast amounts of computation which takes a substantial amount of time, this algorithm does not have this problem.

Compared to other training solutions, Alpex has a constant space complexity. This allows for the algorithm to be scalable and operate without the risk of running out of resources. However, a limitation here is the fact that the base memory requirement is substantial which restricts the range of values the algorithm can predict for.

In conjunction with the promises that Alpex shows in terms of predictions and computational requirements, Alpex has one very distinct advantage over Regression models, such as ARIMA. Alpex is completely non-configurable, the algorithm just runs; this is vastly different to complex parametric models like ARIMA. ARIMA requires specialist training and manual interpretation of datasets, by expensive data analysts, before forecasts can take place. This ease of operation is a distinct advantage that Alpex has over the competition.

## 1.3 Current Time Series Prediction Algorithms

As Time Series are such an important research topic with many profitable applications, there is a lot of literature and strategies on ways to predict them. Some of these methods are outlined here.

- In smoothing algorithms, a prediction is expressed as a weighted sum of previous values, there are many different types of smoothing algorithms, such as exponential smoothing where as values get further away the weights become smaller.
- The most used methods involve averaging the data in some way, the simple implementation is a Simple Moving Average, where we predict the next value based on the previous values. Nowadays, we use algorithms such as Autoregressive Integrated Moving Average (ARIMA) which are effective at predicting one step of a time series.
- Neural Networks are extremely prominent in Time Series predictions; however, these are extremely computationally heavy, requiring multiple large linear algebra computations to be completed per prediction. This makes them generally quite slow, and they require vast amount of training data. Neural Networks come in many different shapes and sizes, some examples are Elman Neural Networks or Multilayer Perceptron networks.

## 1.4 Aims

The aim of this project is to refine this rough diamond by answering two main questions:

1. **How can we make Alpex accessible and deployable?**: Before the start of this project Alpex was a procedurally written, C, command line program, that had no arguments and was not cross platform. The program searched for a file called ‘idata.dat’ and outputted forecasts to a file ‘odata.dat’. This is not accessible or user friendly, therefore the first part of the project must create an implementation that is.
2. **For which classes of time series is Alpex most effective?**: Following on from past research, at the start of this project we had several Excel spreadsheets that contained a basic analysis of Alpex’s ability on a subset of Chaotic Time Series. Although these result were very exciting, we had no tangible analysis and were therefore limited in our conclusions. We aim to further, and robustly, analyse the ability of Alpex in this setting. Progressing from an investigation into Synthetic time series we explore the capabilities of Alpex on real-world datasets, and crucially aim to find a real-world context where this algorithm could be applicable.

## 1.5 Contributions

We have set out the aim of this project in the form of two distinct questions which form the different parts of this project.

### 1.5.1 How can we make Alpex accessible and deployable?

The first question forms the Software Engineering side of the project. To make Alpex accessible we must:

Develop an extensible, well tested, secure and portable, production ready, Object Orientated implementation of Alpex which supports the streaming of data into and forecasts out of a C++ API. We then extend this API with Python wrappers, a command line program and example implementations.

To supplement the API and address the substantial memory requirements of the algorithm, we extend this project by implementing a Time Series as a Service web application, forming the DevOps side of this project. This is done in a RESTful way to sustain scalability where interesting and novel architectural design elements were needed to maintain key properties of the algorithm.

Finally to aid our investigation into Alpex we develop a suite of Python tools that include sample experiments (in the form of Jupyter Notebooks), graphing functions, base line predictors, metric generators and much more.

### 1.5.2 For which classes of time series is Alpex most effective?

The later question will be answered in two parts. We will first conduct an analysis into Synthetic chaotic time series with a known chaotic structure. The former part of our investigation forms a pre-requisite into our investigation into the performance of Alpex on real-world datasets with an unkown underlying model.

#### Synthetic Analysis

In this project we continue research into the performance of Alpex in a Mathematical setting. We knew that Alpex appeared to be fast in terms of runtime and what had observed the capabilities of the algorithm in rapidly learning classes of chaotic time series. We perform a comprehensive investigation into these types of series, evaluating how Alpex compares to Auto Regressive models.

As part of our systematic investigation, we conduct an analysis into the noise tolerance of Alpex on a number of different time series. We then propose and test different methods for making predictions on noisy datasets.

During our experimentation we also explore emergent properties of the algorithm, such as spikes in our predictions.

**Real-World Analysis**

The final section of this project explores over one hundred real-world time series to determine a set of properties that these series should, or should not, have for Alpex to be able to make predictions.

We set out a series of tests and develop a novel metric specifically designed for Alpex, to determine if we are making proactive or reactive forecasts.

Finally we go through a case study relating to the hourly power load of southern Italy, using this to explain how and when we could use Alpex.

**Evaluation**

Our evaluation draws conclusions and direct comparisons between Alpex and a (S)ARIMA implementation. We find that in some classes of Chaotic Time Series Alpex is able to significantly outperform ARIMA, and in others it is able to competitively compete with this method. When comparing Alpex in real-world settings we see how it is able to detect underlying periodicity and patterns that produce forecasts as accurate as ARIMAs.

Taking into consideration factors such as runtime, memory complexity and the algorithms non-configurability, we see the clear and distinct advantage Alpex has against ARIMA; hypothesising how Alpex could be used in large scale, high fidelity systems where we are able to train and forecast on rapidly arriving and changing data.

# Chapter 2

# Time Series

## 2.1 What is a Chaotic Time Series?

Chaos in Mathematics is characterised by non-linear dynamical systems typically described by relatively simple equations, which are deterministic in nature, yet are capable of producing unpredictable and divergent behaviour [49]. Non-linearity in the context of a feedback process results in a system which displays sensitive dependence on initial conditions. This means that small changes in initial conditions may produce wildly different results. So Chaotic time series are known to be hard to predict, since what is true for 'initial conditions' also applies at each iteration of the evaluation of a Chaotic system.

Fractal is defined in mathematics by a value or series that is infinitely detailed and self-similar without ever repeating itself at the same scale. Mathematical chaos exhibits a Fractal nature, if you were to find the exact value then you can see it will never repeat itself; due to precision limitations in Computing, Chaos values will appear to repeat themselves.

Attractors are often talked about within Chaos, an Attractor defines where a series settles towards over time. As well as simple attractors, attractors that oscillate between many values also exist, this type of attractor is called a limit cycle. In Chaos, a never repeating limit cycle is observed.

## 2.2 Time Series Properties

### 2.2.1 Heteroskedasticity

A time series is said to be heteroskedastic[84] when the variance of values is not constant. In practice this can be seen as having periods of high volatility and periods of low volatility randomly distributed across a series. The opposite of heteroskedastic is homoskedastic and

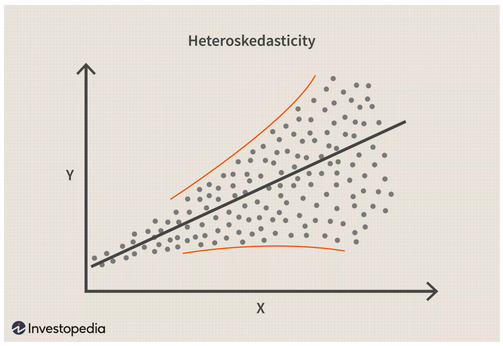


**Figure 2.1:** A time series with a non constant variance[24].

it means there is a constant variance throughout the series. Figure 2.1 is an example of a series with a non standard variance, as you can see the values along the y axis begin to span out.

### Detecting Heteroskedasticity

One way to detect this property is the Goldfel-Quandt[58][57] test, which checks for homoskedasticity. In this test we split the dataset into two sections and calculate a regression for both, and the corresponding Mean Squared Error, then if we take a fraction of these errors $\frac{MSE_A}{MSE_B}$ we can use this to reject a null hypothesis of homoskedasticity.

### Removing Heteroskedasticity

There is no way that always works for removing Heteroskedasticity from a time series, like we could with seasonality, however if we take logs of the time series, then we can often remove this property.

If we are creating a regression model, we know that the variance of the error term is non-constant, however if we take our regression:

$$Y_i = \beta_0 + \beta_1 * X_i + \epsilon_i$$

where the variance of $\epsilon_i$ is non-constant, and divide it by:

$$\frac{Y_i}{\sqrt{X_i}} = \frac{\beta_0}{\sqrt{X_i}} + \frac{\beta_1 * X_i}{\sqrt{X_i}} + \frac{\epsilon_i}{\sqrt{X_i}}$$

our new error term $\frac{\epsilon_i}{\sqrt{X_i}}$ will have a constant variance, therefore removing Heteroskedasticity from our regression model. This is known as the Weighted Least Squares[51].

### 2.2.2 Auto Correlation Function

The Auto Correlation of a series is the correlation of a signal with a delayed copy of itself [86]. This Auto Correlation is used to detect periodicity and patterns in noisy data.

The Auto correlation of a continuous time series with a delay of $\tau$ is defined as:

$$R_{ff}(\tau) = \int f(t+\tau)\bar{f(t)}$$

Where $\bar{f(t)}$ is defined as the complex conjugate of $f(t)$. However for real world time series we are usually interested in the discrete definition defined as:

$$R_{yy}(\tau) = \sum y(t+\tau)\bar{y(t)}$$

When we are looking at time series it is often useful to plot a Correlogram about the series [82]. This is also known as the autocorrelation plot as it is a graph of autocorrelation against lags for a series.

When we analyse a Correlogram, a value less than a lower threshold implies that there is no correlation at the specified lag. If we are unable to detect correlation at any lags then we can say that the data is most likely random. This is often used to determine if a forecasting algorithm is appropriate and the parameters to use.

ARMA uses this to determine how far the Moving Average should look back, as if there is no correlation past a lag of $\tau$ then there is no point including values greater than that in our moving average.

### 2.2.3 Partial Auto Correlation Function

Partial Auto Correlation of a time series [76], $z_t$, is the correlation between itself and a lagged version of itself, $z_{t+\tau}$, that is not accounted for by the lags between $1 \rightarrow \tau - 1$. The PACF is used to determine the autocorrelation factor in a time series.

### 2.2.4 Volatility

#### Standard Deviation

Standard Deviation of a series tells us the spread of numbers within the series around the mean. This can be important to check in time series as some algorithms perform better when values are grouped close together where as some are more tolerant to outliers. Furthermore as the standard deviation is the square root of the variance it can act as a good measure for the volatility of a series [58]. The standard deviation of the differences of a dataset can also be calculated to find out the variance in the change of values. Often a scaled standard deviation is more useful, this is calculated by dividing the standard deviation by the mean.

### 2.2.5 Seasonality

Seasonality is a predictable repeating pattern within a set unit of time, this unit of time can be a month, a week, or more commonly a year. However, seasonality is not usually seen as a repeating pattern within a quantity of time, for example within two weeks, as these are often unpredictable, and we call these cycles.

An example of a seasonal trend would be a hotel keeping track of the number of families within a year, it is likely that every year during school holidays the hotel has a lot more families than during term time.

Seasons are everywhere in time series, with different seasons often overlapping, for example, an ice cream shop may be busier during the summer, but also busier on the weekend. Therefore, it may be useful to look at different seasons separately and remove them one by one.

Dealing with seasonality often requires a vast amount of data over a long period of time, we are only able to start dealing with seasons after at least one full season has been observed. We can reason about the length of a season without seeing concrete trends in the data, for example an ice cream shop is intuitively going to be busier during the summer and therefore have a yearly seasonality.

An interesting problem to consider with seasonality is dealing with leap years.

There are several ways of dealing with seasonality,

- One of these methods is to transform the data by the value from the previous period. For example, we would transform a time series by

  $$z_t = x_t - x_{t-365}$$

  then we can just perform our predictions on the new time series z then perform a transformation back into the original space. This is usually the best way to remove seasonality for data, however it requires intuitive knowledge of the period to define this.

- Another strategy as outlined in this paper [56]) is to take the logarithm of the dataset such that the seasonality is now additive and can be predictable. This isn't as powerful as taking the difference, however we do not need any past data to implement this.

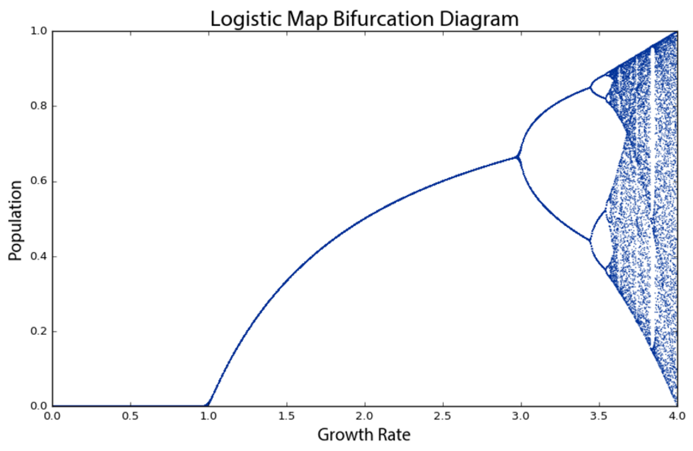


**Figure 2.2:** A graph showing the possible values for the population at each growth rate, assuming an initial population of 0.5. [49]

## 2.3 Chaotic Time Series

### 2.3.1 Logistic Function

The logisitic function, discovered in 1976 also known as the Logistic Map, is one of the most cited examples of chaos, with applications involving modelling population growth [80] and germination [77]. The algorithm is defined by:

$$x_{t+1} = r * x_t * (1 - x_t) \tag{2.1}$$

where r is the rate of growth and x is the population. This equation appears very simplistic therefore we would expect that the behaviour would be such; what we actually observe, for some values of r greater than 3.57, is Chaos. We see that for these values the series does not converge, occurring unpredictably to the point that it appears almost random.

Figure 2.2 shows the attractor for different growth rates for the Logistic map, what we see is that after around 3.57 there is no obvious attractor and that any value between 0 and 1 appears possible.

Figure 2.3 shows another interesting property of the logistic map and chaotic series in general, if you zoom into the graph, we observe the same values and the same pattern, this is because of the Fractal nature of Chaos [59].

Figure 2.4 shows us a snippet of the time series for a growth rate of 3.8 and an initial value of 0.7, there appears to be no pattern within the data. The graph appears random in nature, even though we know it is not.

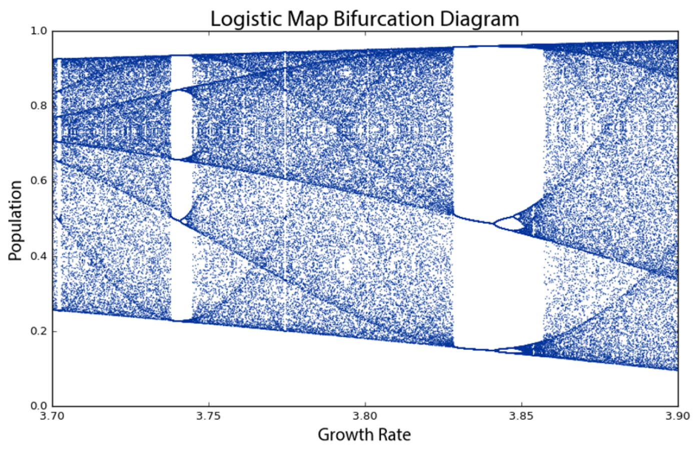


**Figure 2.3:** A zoomed in view of the graph in Figure 2.2. [49]

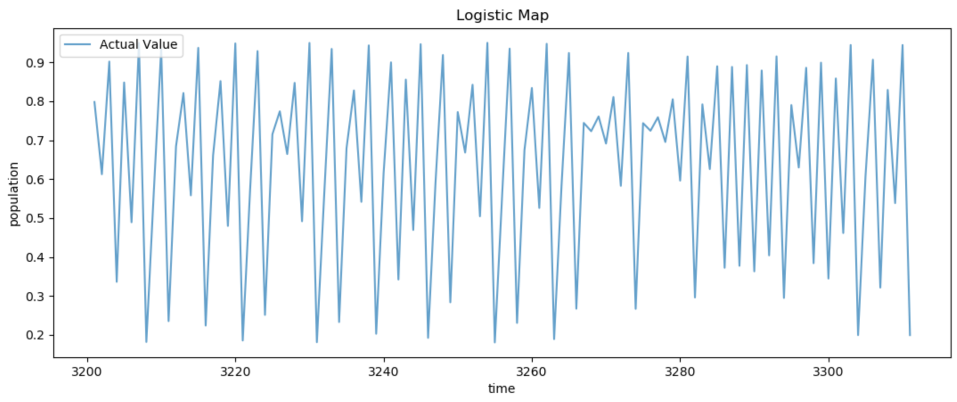


**Figure 2.4:** A graph of Population against Time for the Logistic map with a growth rate of 3.8 and initial value of 0.7.

## 2.3.2 Henon Map

An example of a multi-dimensional Chaotic Time Series is the Henon Map [78]. The Henon Map is defined as follows:

$$p_{n+1} = \begin{cases} x_{n+1} = 1 - a * x_n^2 + y_n \\ y_{n+1} = b * x_n \end{cases} \tag{2.2}$$

where a and b are constants. The Classical Henon Map is defined with parameters $a = 1.4$ and $b = 0.3$.

This function can be composed into one dimension as

$$x_{n+1} = 1 - a * x_n^2 + b * x_{n-1}. \tag{2.3}$$

We can see from this that the function being used to construct the series appears fairly simplistic, however when we look at the Bifuriation Diagram, Figure 2.5, we see this demonstrates

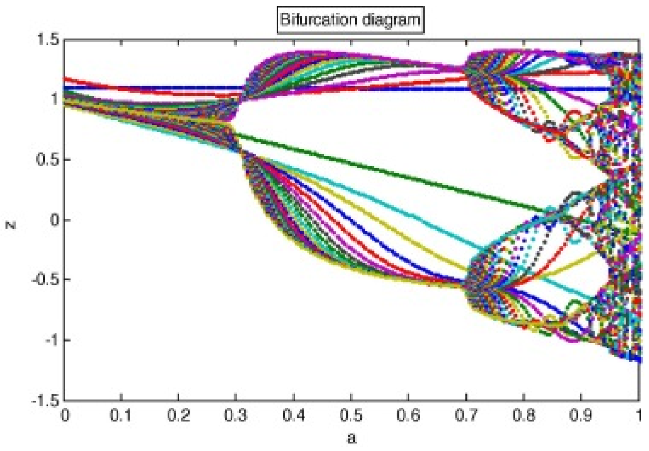


**Figure 2.5:** A graph showing the possible values the henon map is able to take on [78]

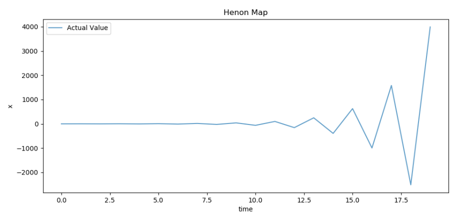


**Figure 2.6:** A graph showing the first 20 values of the henon map time series with a = 1.4 and b = 0.3.

chaos. This is also seen in Figure 2.6 where we can see a snippet of the first 20 values.

### 2.3.3 Mackey-Glass System

The Mackey-Glass System[71] is an example of a differential Time Series that can exhibit Chaos. The system is defined by the equation:

$$\frac{dx}{dt} = \beta * \frac{x_\tau}{1 + x_\tau^n} - \gamma * x. \tag{2.4}$$

### 2.3.4 Lorenz System

Although the Logistic Map is the most cited example of chaos, the Lorenz System is regarded as being the first example of chaos, having been discovered in 1963[70] as a mathematical representation of atmospheric convection. The Lorenz System is both a multi-dimensional and differential Chaotic Time Series. The Lorenz System is defined as:

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x * (-z) - y \\ \frac{dz}{dt} = x * y - \beta * z \end{cases} \tag{2.5}$$

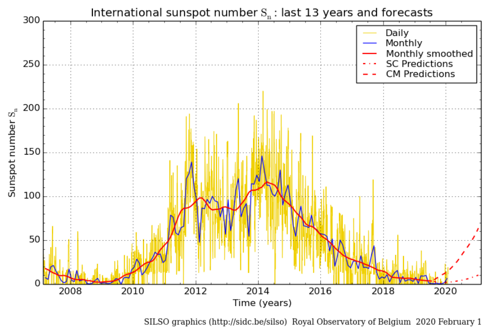


**Figure 2.7:** A graph showing the daily and monthly sunspots for the last 13 years [14]

## 2.4 Real-World Time Series

### 2.4.1 Sun Spot Prediction

We are able to measure the activity of the sun by counting the number of sunspots observed over a period of time, it is believed that the measure of these sunspots is related to weather on earth, and it is known that they are relevant in atmospheric density which effects satellite trajectory. Sunspot numbers appear to follow a cycle and we are therefore able to attempt at creating models that are able to predict the number of expected sunspots in a given month. Monthly sunspot data goes back all the way to 1749, therefore we have a vast amount of data to use. This paper [73] is one of the main papers about how sunspot cycles exhibit chaos and appear like a chaotic time series.

### 2.4.2 Open Power System Data

The Open Power System Data[15] project is a free collaborative project between researchers across Europe that aims to provide datasets regarding electrical systems for researchers to model. The variety and number of datasets available is astounding, as well as the number of data points available for these many reaching well over 30,000. Data is available for many regions and countries across Europe, some of the data available for these regions include: Offshore Wind power generation, solar power generation, energy consumption and the prices for these.

# Chapter 3

# Predicting Time Series

## 3.1 Time Series Prediction Methods

### 3.1.1 Naive Forecasting

$$a_t = x_{t-1}$$

### 3.1.2 Moving Average

#### Method Description

A Moving Average (MA) of a datapoint, is the average of the last n values in the series, it can be expressed mathematically as:

$$a_t = \frac{\sum_{i=0}^{n} x_{t-i}}{n} \tag{3.1}$$

Where $a_t$ is the average at time $t$, $x_t$ is the observed value at time $t$, and $n$ is the number of values to include in the moving average.

One type of moving average is a Weighted Moving Average (WMA) where values that are further away have less of an affect on the overall average:

$$a_t = \frac{\sum_{i=0}^{n} (n-i) * x_{t-i}}{n * \frac{n+1}{2}}. \tag{3.2}$$

Another type of weighted moving average is an Exponential Moving Average[61] (EMA), which is very similar to the linear weighted moving average, however instead of a linear reduction in weight, we see an exponential decrease:

$$a_t = (x_t - a_t) * \sigma + a_t, \text{ where } \sigma = \frac{2}{n+1} \text{and} a_{t-n} = x_{t-n} \tag{3.3}$$

Where $\sigma$ is a smoothing constant. This is often regarded as Exponential Smoothing.

Comparing the different types of Moving Averages, we can see that the EMA has weights that fall off much quicker than the simple WMA and the MA therefore it is much more responsive to changes in the values. However, this means that the averages are very susceptible to noise, if a noisy or outlier is seen, then it will affect the other values far more, this is what the smoothing constant aims to fix.

**Advantages**

An advantage of MA's is that they implement some sort of smoothing where noise will not immediately affect the value, therefore one random spike in data may not affect the average significantly.

Another advantage of MA's is that they are very fast, at a maximum the forecasting time has a complexity $O(N)$ where $N$ is the size of the window.

**Disadvantages**

A disadvantage of all types of MA is the concept of lag, as the average is based on prior data, it means that significant spikes in data will not necessarily be seen immediately.

Another disadvantage of MA's is the fact that they can only really be used for very short term predictions, after a couple of values (depending on the window size) the predictions will begin to normalise themselves.

### 3.1.3 Auto Regressive Models

Auto Regressive (AR) models are about predicting future values based on specific previous values. For example, for a shop, you may predict this month's demand based on the demands from previous months. This is different to MA as in MA you look at the average of the last few months, in this model, you look at the actual value from the last few months:

$$a_t = \beta_0 + \beta_1 x_{t-1} + \beta_2 x_{t-2} ... + \beta_n x_{t-n} + \epsilon_t = \beta_0 + \sum_{k=1}^{N} \beta_k x_{t-k} + \epsilon_t \tag{3.4}$$

What we see from the above equation is that each past value, relative to the current value, is given its own weight $\beta_k$. If we calculate the ACF of a dataset, seen in section2.2.2, then we can see the correlation of past values on the current value. We use this to determine our value for n.

#### Auto Regressive Moving Average Model

The Auto Regressive Moving Average Model (ARMA) model combines the two different techniques of the AR and MA models. The function for ARMA (1, 1) would be:

$$a_t = \beta_0 + \beta_1 x_{t-1} + \phi_1 \epsilon_{t-1} + \epsilon_t \tag{3.5}$$

Where $a_t$ is the actual value, this can be calculated if we know the error, where we use the last values error term to get $\phi_1 \epsilon_{t-1}$ (the MA part) and the actual last value $\beta_1 x_(t-1)$ (the AR part). We do not however know the value of this month's error, so we take a prediction as

$$\hat{a_t} = \beta_0 + \beta_1 x_{t-1} + \phi_1 * \epsilon_{t-1} \tag{3.6}$$

For the generic case of ARMA (p, d) we have

$$\hat{a_t} = \beta_0 + \sum_{n=1}^{p} \beta_n x_{t-n} + \sum_{n=1}^{d} \phi_n \epsilon_{t-n} \tag{3.7}$$

### 3.1.4 Auto Regressive Integrated Moving Average Model

The Auto Regressive Integrated Moving Average Model(ARIMA) model is similar to the ARMA model, however before we make a forecast we must first difference the data until it is stationary. The integrated variable determines the number of times that the dataset must be diffed by:

$$z_t = a_{t+1} - a_t \tag{3.8}$$

We can then fit our model on the differenced dataset, and return the predictions to the original domain by integrated our predictions as:

$$\hat{a_{t+1}} = \hat{z_t} + x_t \tag{3.9}$$

ARIMA is also configured, where ARIMA (p, q, d) has p representing the AR part, d representing the MA part, and q represents the number of times we transform the graph.

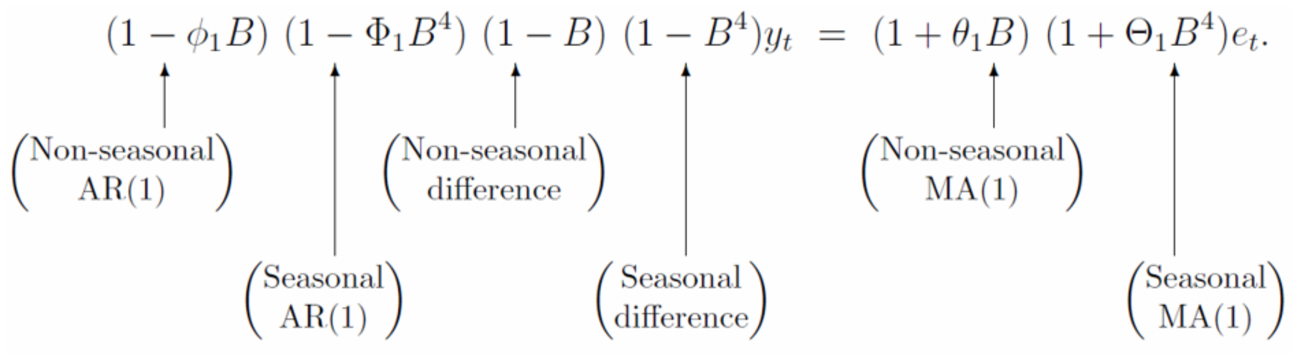


**Figure 3.1:** A definition of $SARIMA(1,1,1)(1,1,1)_4$ [43]

**Advantages**

Works on timeseries with a nonconstant mean, when it has an obvious linear (m = 1) trend upwards or downwards.

ARIMA is very good at forecasting t+1 predictions.

**Disadvantages**

Does not deal with seasonality in datasets.

ARIMA requires complex configuration to define the correct parameters for the optimal model. The current methodology for automating this configuration is through a naive, slow, grid search approach, an example of this being used is seen in [74]. Specialist training is often required for data analysts to make accurate ARIMA models.

### 3.1.5 Seasonal Auto Regressive Integrated Moving Average Model

The Seasonal ARIMA (SARIMA) model is like the ARIMA model, however we must remove seasonality from the dataset by subtracting the same value in the dataset from the previous season. This can be done by

$$s_t = a_t - a_{t-m} \tag{3.10}$$

Where m is the size of the period, for example a week would use $m = 7$.

Compared to ARIMA, SARIMA is far more complicated when it comes to parameters, we have $SARIMA(p, q, d)(P, Q, D)_m$ where p, q and d mean the same as they did before, and P, Q and D represent the seasonal variants, m is again the size of the period. $SARIMA(1, 1, 1)(1, 1, 1)_4$ is represented in Figure 3.1. Parameter tuning with SARIMA is like ARIMA where we look at ACF and PACF to find out what the different variables should be tuned towards.

#### Advantages

SARIMA shares all the benefits from the ARIMA model, however it also has the benefit of dealing with seasonality which almost every real-world time series must deal with.

#### Disadvantages

Although by looking at ACF and PACF graphs we can make good estimations on the parameters that we should tune the algorithm towards, it is hard to find the optimal combination of these. Often this process of tuning is extremely slow, taking a long time and the optimal configuration could change over time, even to the point where the configuration no longer yields helpful results.

Complexity is a barrier to adoption for SARIMA modeling, the equation is hard to understand and implement.

Another limiting factor with any algorithm that deals with seasonality is the fact that to remove seasonality, a vast amount of data must be observed. Only after a few periods of the time are we able to start to detect and account for seasonality, when this period is a year it means we need years of data before we can start to make predictions. This can be mitigated by intuitively calculating periodicity, for example we can reason that a hotel planning to model the number of guests that they will have in a particular week are likely to find a strong yearly seasonal trend. Although we can intuitively reason about this, computers are not able to reason in this way, although some work has been done in this field [60] it is not widely adopted.

### 3.1.6 Feed-Forward Neural Networks

#### Method Description

An Artificial Neural Network (ANN) is a series of nodes and weighted connections that allow for outputs to be generated by a set of input variables. The structure of an ANN attempts to mimic how a simple neuron passes signals inside the brain.

Each node inside an ANN takes weighted input from all nodes in the previous layer to generate an output value which is then fed forward to the next layer in the network. This process starts from the input nodes which take in normalised input values, feeding values forward until they reach the output layer. This process of feeding values through the layers of a network is where the name Feed-Forward Neural Network comes from. Figure 3.2 shows an example of an ANN and how weights are connected between different nodes.

Training of an ANN is about finding the right weights for each of the connections between nodes, such that they yield correct output values. There are a number of techniques used to train these weights.

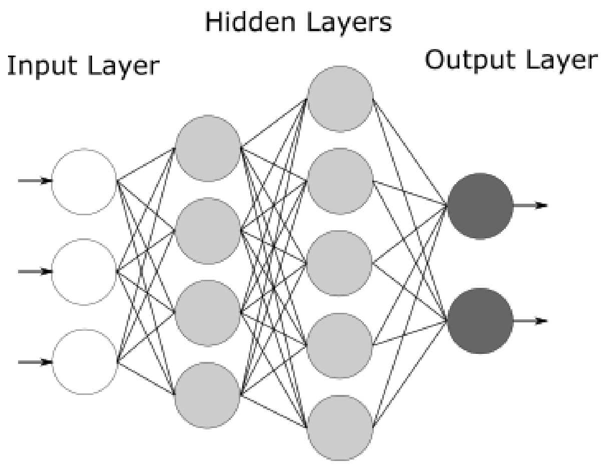


**Figure 3.2:** A Feed Forward neural network with 2 hidden layers

### Mathematical Description

We know that an Artificial Neuron takes in weighted inputs from every node in the previous layer, this can be represented by multiplying the value from the node and the weight of the connection between the nodes.

$$x_j = \sum_{i=0}^{nodes} W_{i,j} * x_i \tag{3.11}$$

To try and reduce overfitting, we want to give each node a bias

$$x_j = \sum_{i=0}^{nodes} W_{i,j} * x_i + b \tag{3.12}$$

Values inside a network should also be normalised, if we are summing and multiplying values then we will lose track of this normalisation, so the last thing we want to do is pass this output value through an Activation Function. It is important to note, this is often left out of derivations as it is implied. This leads to a final definition:

$$x_j = \sigma\big(\sum_{i=0}^{nodes} W_{i,j} * x_i + b\big) \tag{3.13}$$

An example of this is seen in Figure 3.3 where we have a hidden layer with two nodes, node 1 and node 2, these nodes feed into one output node, node 3. As we can see we have the output from nodes 1 and 2 as $x_i$ and we have values for the weight between the nodes and the output node as $w_{i,3}$ which is the weight from node i to node 3. We can then see that node 3 takes the sum of these values plus the biases,

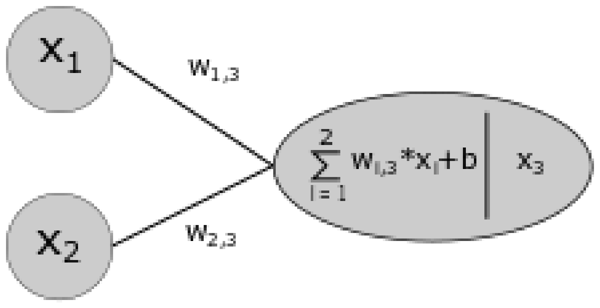


**Figure 3.3:** A single Neuron inside an ANN with 2 input nodes and one output node

$$x_3 = \sum_{i=1}^{2} W_{i,j} * x_i + b = w_{1,3} * x_1 + w_{2,3} * x_2 + b \tag{3.14}$$

passing these values through the activation function, denoted by the vertical line.

If we look at another mathematical representation of neural networks we can see that this can be represented using Linear Algebra, the input nodes can be represented as a vector where we have

$$inputLayer = \big(I_1, I_2, ..., I_i\big) \tag{3.15}$$

$$weightsBetweenLayers = \begin{pmatrix} w_{0,0} & \cdots & w_{0,j} \\ \vdots & \ddots & \vdots \\ w_{i,0} & \cdots & w_{i,j} \end{pmatrix} \tag{3.16}$$

then if we multiply these together, we get

$$\big(x_1, x_2, ..., x_j\big) = \big(I_1, I_2, ..., I_i\big) * \begin{pmatrix} w_{0,0} & \cdots & w_{0,j} \\ \vdots & \ddots & \vdots \\ w_{i,0} & \cdots & w_{i,j} \end{pmatrix} \tag{3.17}$$

which is a way of defining a layer in a neural network.

### Training Methods

Backpropagation is one of the most utilised methods for training a Regression neural network, this training algorithm aims to reduce the value of a cost function using gradient descent. For example, we could have a cost function which works out the difference between the actual and predicted value

$$C = \sum_{i=0}^{outputs} (actual_i - predicted_i)^2 \tag{3.18}$$

The algorithm will then use derivatives to try and reduce this value by changing the weights of the network [68].

Backpropagation by itself is usually very effective at predicting values with a good amount of accuracy, however there are many different optimisations (for example changing the cost function) that can be applied to either make the algorithm more accurate, reduce overfitting, or decrease training time.

Probabilistic models can also be used for the training of Neural Networks[47], where we apply a survival of the fittest approach to training a network. This is often not feasible in practice due to the complexity and memory requirement of maintaining so many networks needed for strong population diversity.

#### Hyperparameters

Hyperparameters are, generally, constant values inside a Neural Network used to configure the network. Examples of these are the structure of the network (number of input nodes, hidden nodes...), the bias in the network etc... These values can have a huge effect on the success of the Network, therefore optimisation of these parameters is extremely important. This paper [48], published by MIT, talks about the importance of hyperparameter optimisations in different scenarios and talks about the gradient technique for the optimisation of these.

This paper [72] takes an interesting look at training a network, it uses backpropagation as its main training mechanism, however it combines this with a Genetic Algorithm for hyperparameter optimisation. The paper was able to conclude that by using a GA instead of just trial and error it was able to get more accurate predictions.

#### Advantages

A neural network can map linear or non-linear functions to the input data, without knowing any prior information about the underlying data. This means that we don't need to know what the actual function is that made the data, however we are still able to make a reasonable estimation to what this could be.

If the network is not overfit, compared to a purely mathematical model, the ANN is better at tolerating noise in data. Mathematical models are often very rigid in their structure, ANN's are generally more flexible with the data they can handle.

#### Disadvantages

The structure of the network is extremely important, a too complex network and it will train to the input values, including noise, causing the network to become overfit; however a network that is not complex enough may fail to fully detect the correlation between the data and output. Therefore, a lot of configuration is needed to try and find the optimal structure, as ANN's are slow to train, this can be time consuming, and can change based on data.

As we know the definition of an ANN through matrices, we can calculate the time complexity of the network. Matrix multiplication of a $1 * i$ and a $i * j$ matrix has a complexity of $O(i * j)$ therefore if we have n layers we have the total complexity as

$$O\big(i * \sum_{i=0}^{n} nodes(j)\big) \approx O(i * n * j) \tag{3.19}$$

which means that the process of forecasting from a neural network can be extremely slow for large networks, and not scalable.

As well as this, the storage of all of these matrices uses up a lot of memory, practical networks have thousands of nodes and a few layers, this creates massive matrices that need to be stored in memory during calculations. As each node takes in values from all other nodes in the previous layer, these networks are often difficult to distribute over different computers.

Training methods such as backpropagation are extremely slow and computationally complex, for large networks training these can take hours and are therefore usually done overnight or in batches. This can make networks less responsive to changes, for example if something drastic occurs the network may not be able to respond until the next day.

#### Case Study

This paper [87] talks about using an Adaptive Neural Network implementation for time series prediction, comparing this new type of ANN against traditional methods such as AR models as well as the basic ANN we have discussed. The paper concludes that using this type of ANN is able to generate more accurate results for real time series than traditional Mathematical models. The paper also found that the traditional ANN was able to outperform AR models in most mathematical time series, however this difference was small.

### 3.1.7 Elman Neural Networks

#### Method Description

An Elman Neural Network differs from a simple Artificial Neural Network as it maintains memory inside the network itself. Figure 3.4 shows how a Feed Forward ANN would maintain memory about the last 3 values it has seen, taking these values as inputs into the network.

Figure 3.5 shows us how an Elman Neural Network, uses weighted context nodes to automatically feed past values into the network, forming a concept of memory[65]. This changes

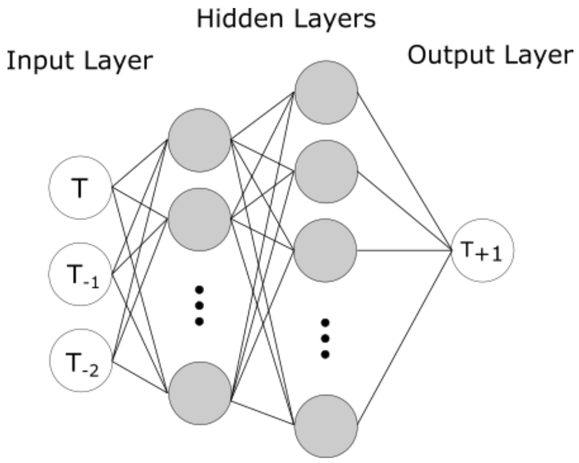


**Figure 3.4:** A Feed Forward Neural Network with 2 hidden layers implementing memory

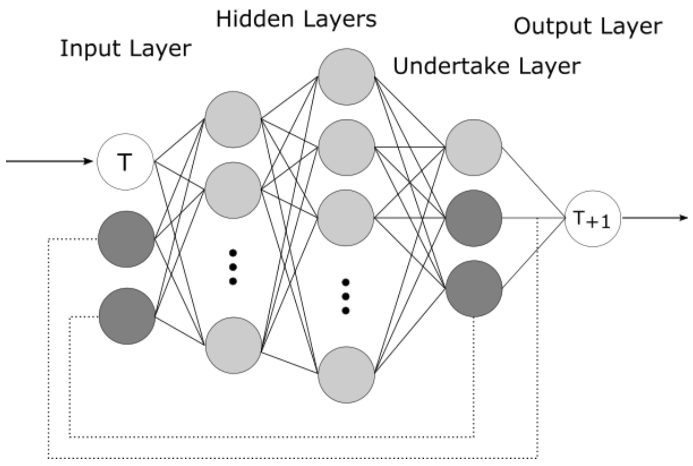


**Figure 3.5:** An Elman Neural Network with 2 hidden layers and 2 context nodes being fed in from the undertake layer, implementing memory

the network such that it takes in just one value, which is the observed value and then returns one value which is the predicted value. The predicted value is not fed into the network [66].

### Mathematical Description

The change in terms of a mathematical definition for an Elman Neural Network compared to a Feed Forward ANN is in the first hidden layer

$$x_j = \sum_{i=0}^{inputs} W_{i,j} * \sigma(I_i) + b \tag{3.20}$$

the first layer changes to use the context nodes as,

$$x_j = \sum_{i=0}^{inputs} W_{i,j} * \sigma(I_i) + \sum_{i=0}^{contextNodes} W_{inputNodes+i,j} * \sigma(C_i) + b \tag{3.21}$$

where $C_i$ is the value of the context node from the last iteration.

When working with small datasets, it is important to consider initial values of the Context nodes.

### Advantages

The ENN has a number of key advantages over other types of ANN's. ENN's have the network deal with the complexity of maintaining a memory of past values. This comes at a cost to flexibility. This is particularly useful within Time Series prediction algorithms, as we often want to consider past values.

It has been observed in [81] that as networks train on the context nodes (and other reasons), we are able to achieve higher accuracy in ENN's than regular Feed Forward Neural Networks in many situations.

### Disadvantages

ENN's maintaining their own memory limits clients to sequential predictions, for example we cannot have multiple power stations trying to predict their own power usage using the same Network. Even though they may have the same training set, and therefore a similar network, they will all have different histories, so we would need to copy the network multiple times for each station.

As we are not passing the real value back into the network as input, we may lose track of some values and noisy values may be treated in a way that we do not want them to.

Similar to how hyperparameter tuning is important in ANN's, similarly we see that the learning rate of ENN's is even more important and greatly effects the performance of a system. Most ENN's use a constant learning rate to train the network, however this causes values to converge slowly; there are ways to dynamically set the learning rate, as outlined in this paper [88], however these are complex to implement.

### Case Study

This paper [69] utilises Elman Neural Networks for controlling an air conditioning unit. VAV Air Conditioning units take in the current temperature and use this to change the power of the unit, however there is a lag between reading the temperature and adjusting the unit, therefore the paper discusses how an ENN can be used to predict the temperature and use the predicted value to adjust the unit. The system takes 5 minutes to adjust the temperature, and is able to read temperature in intervals of 1 minute, therefore the network makes a 5 step prediction. By using this method they were able to improve control stability within the system.

### 3.1.8 Distributed Neural Networks

As seen above, Neural Networks are very large and computationally slow to train and predict values, this is a limiting factor in the usefulness of ANN's in certain scenarios. To mitigate this, Distributed Neural Networks have been proposed. One approach outlined in this paper [83] uses Distributed Multistructure Neural Networks, where the network is split into smaller networks and each node in the system is trained on a different unit class. A combiner is then used to combine the different outputs into one single output. The paper also talks about using a synergistic multistructure network where within each unit class, we would distribute different networks that use different activation functions to train the data, then combine these; increasing the accuracy of the output.

This paper [79] discusses the advantages of Distributed Deep Neural Networks in reducing the memory usage of each layer of the network. The paper applies this in a scenario where early layers of the network are able to run on end devices, and then later layers run in the cloud, this has a number of unique advantages; such as an increase in data privacy as the data has already gone through one network.

## 3.2 Time Series as a Service

In today's world of Internet of Things, it seems every industry and every part of Computing is tending towards a Platform as a Service (PaaS) or Software as a Service (SaaS) infrastructure, and it appears Time Series prediction is no exception. Amazon and Microsoft offer a service for this, AWS Forecast and Azures Automated Machine Learning. These are not the only companies offering forecasting services, companies like IBM [21] offer them, as well as start ups like Causa Lens [8].

SaaS is the practice of a whole system being hosted on a server, clients interact with this network to perform operation. All data and the operations themselves are carried out on the server instead of the clients machine. This paper [52] talks about the rise of SaaS, the growth this experienced in the 2000's and how switching to a SaaS architecture affected the industries that adopted them. This article [18] quotes SaaS as accounting for 24% of enterprise workloads.

As systems become more and more complex, and more specialised, it became apparent that separating the data from the processing would have benefits to clients, this is where PaaS [67] comes in. Like SaaS, all operations happen on the host network, however the data and the application itself are managed by the client; allowing for users to manage their own data whilst maintaining the benefits of IOT and SaaS. In this presentation [63], KPMG talk about how IOT will change within the next 6 years, including how PaaS will rise and become more prominent as well as the financial importance of PaaS.

### 3.2.1 AWS Forecast

Amazon offer a PaaS implementation of time series predictions in the form of their AWS Forecast, this system boasts "Amazon Forecast provides forecasts that are up to 50% more accurate by using machine learning" as well as being able to "Reduce forecasting time from months to hours" [42].

AWS Forecast allows for clients to choose the model to train their predictor on [9], for example a client could choose ARIMA, Exponential Smoothing or Amazons DeepAR+[16] algorithm; however each of these algorithms requires the user to manually configure their hyper parameters.

AWS Forecast allows for a pre-packaged solution to many of the algorithms already discussed. Allowing anyone to be able to take advantage of modern Time Series methods.

### 3.2.2 Azure Machine Learning

Microsoft offers a suite of different PaaS tools that can be used to predict values from data, on top of these they have focused sections of their documentation [4] on how to use these services for Time Series predictions. Although the algorithms may be just as effective, Azure uses general ML tools compared to AWS which has specific Time Series prediction tools. Microsoft take the approach of using the tools they already have [5] but providing documentation to make them work in this scenario.

### 3.2.3 Software as a Service

One advantage the AWS and Microsoft PaaS solutions have, is integration with their respective cloud platforms. This article [89] talks about the importance of SaaS in Time Series; how using a time series specific database is able to provide unique opportunities and advantages. This is quite similar to how AWS works, AWS Timestream [1] is an extremely fast (up to 1000x) way of storing Time Series data; their AWS Forecast library is then able to use this directly, to be able to forecast much quicker.

## 3.3 Metrics for Predicting Time Series

### 3.3.1 Root Mean Square

Root Mean Square Error (RMSE) in Statistics is a measure of how far away from the actual value you are. It is the standard deviation of the prediction errors and is defined as:

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(\hat{a}_i - x_i)^2}{N}} \tag{3.22}$$

Where $\hat{a}_i$ is the predicted value and $x_i$ is the actual observed value. A low value for RMSE means that the prediction is good, and the predicted values are not too dissimilar.

One of the problems with RMSE is that it amplifies larger errors, for example we may have a very good predictor for most points, however one outlier will cause the RMSE to be much higher. Some systems may be less tolerant to occasional errors, and these should be weighted higher, in these cases RMSE will be a good indicator.

### 3.3.2 Mean Absolute Error

Mean Absolute Error (MAE) is the average error expected for each value:

$$MAE = \sum_{i=1}^{N} \frac{|\hat{a}_i - x_i|}{N} \tag{3.23}$$

This is useful as it tells us the average error in our predictions. This may not always be optimal as in some scenarios we want to amplifier outliers in the data. The smaller the MAE value, the better our predictions are.

### 3.3.3 Mean Absolute Percentage Error

Like the MAE, the Mean Absolute Percentage Error (MAPE) is a measurement of how much each predicted value is from the original:

$$MAPE = \frac{100\%}{N} \sum_{i=1}^{N} \frac{|\hat{a}_i - x_i|}{x_i} \tag{3.24}$$

MAPE has a number of drawbacks however, one of these being that when the predicted value is larger than the actual, we are likely to see higher MAPE scored. This can make MAPE readings unreliable in some scenarios.

A smaller value for MAPE means that our values are closer to the regression line.

### 3.3.4 Mean Percentage Error

The Mean Percentage Error (MPE) is different from MAPE in it does not take the absolute error, this allows us to measure the bias of predictions:

$$MPE = \frac{100\%}{N} \sum_{i=1}^{N} \frac{\hat{a}_i - x_i}{x_i} \tag{3.25}$$

This does not tell you how your algorithm performs overall as the positive and negative parts may cancel out. The smaller the absolute value, the less bias; a negative value tells us that predicted values are generally smaller than the actual, whereas a positive value tells us predictions are generally overestimation's.[44]

### 3.3.5 Comparing Algorithms

If we want to compare an algorithm to a baseline, we can calculate the R squared score for this value:

$$R^2 = 1 - \frac{MSE(model)}{MSE(base)} \tag{3.26}$$

The result of the $R^2$ can be analysed as follows:

$$R^2 \begin{cases} > 0 & \text{The model is better than the base} \\ = 0 & \text{The model is the same as the base} \\ < 0 & \text{The model is worse than the base} \end{cases} \tag{3.27}$$

This is useful as we can see if our new model adds anything compared to the original model. A problem with this however is that it does not penalise models that add no value, for this we can use an Adjusted $R^2$ model as follows:

$$\bar{R}^2 = 1 - (1 - R^2)\left[\frac{n-1}{n-(k+1)}\right]. \tag{3.28}$$

### 3.3.6 Conclusion

There is no best metric for every situation, it is dependent on the type of data we are predicting. If we care about outliers and reducing these, then RMSE should be used. If we want to detect bias in our models, MPE should be used. If we do not care about outliers, but we just want to know how well the algorithm performs in general we would use MAE, or MAPE if we want to compare this across different datasets.

# Part I

# How can we make Alpex accessible and deployable?

# Chapter 4

# Alpex C++ Library

**The main purpose of this algorithm is to allow clients to input a sequence of numbers and to return an accurate prediction based on that sequence.**

The original implementation of the algorithm achieves this aim by taking in a file called ‘idata.dat’, where values in a sequence are seperated by a new line. This file is processed and a new file ‘odata.dat‘ is created which contains the predictions made after each input value.

The main aim for the Software Engineering side of the project was to create an accessible C++ library from the procedural C code which is: secure, extendable, fast, documented, well-tested, portable and Object Orientated. The key to success with this library will be figuring out the most simplistic, yet expandable, definition. By creating this API we are making Alpex accessible for developers.

## 4.1 Design and Technical Specification

### 4.1.1 User Journey

A user must be able to predict a value based on a sequence of numbers, this involves the user being able to train a predictor and then make predictions on that.

As mentioned the success of this library is dependent on creating the most simplistic implementation, that means the library should be able to be extended to support different types of predictions (like t+n) without itself having to support it.

### 4.1.2 Security

The main focus of the security element of this library is to ensure that the Intellectual Property, the algorithm, is hidden once the library is created. This means that a client needs to be able to access the predictor without being able to find anything out about the implementation itself.

### 4.1.3 Testing

The implementation must be well tested such that we can be assured that any changes to the Algorithm's code will not negatively impact the performance. Testing must involve checking the ability of the algorithm to predict different sequences that we know the algorithm can learn. The implementation must also be tested with extreme and erroneous values.

There are a number of testing frameworks that we can use for C++, each with their own advantages and disadvantages. One of the most popular libraries is Google Test [22], this library aims to be a lightweight testing environment which allows for basic testing of C++ programs. Google Test has CMake extensions such that it is cross platform and can be added to a build pipeline.

An alternative to Google Test is Microsoft's Unit Testing Framework [27] this is extremely powerful and comes with Visual Studio installations therefore it is easy to install. The problem with this library is the fact that it is not portable and does not work on Linux machines as they cannot install Visual Studio.

Another widely used library is Boost.Test [6] this is a very in depth and useful framework that includes all of the capabilities you could want from a testing framework. Boost is cross platform, and it has been around for a long time therefore it is well documented. The problem with Boost is the amount of backwards compatibility it provides makes it a very heavy library. As we are aiming to make the library as simple as possible, our test suite should also be fairly simple, therefore the lightweight implementation of Google Test is the optimal choice over Boost.

### 4.1.4 Object Orientated

The client needs to be able to create a predictor, train the predictor and then make predictions using it. This involves the client being able to store this predictor as an object they can interact with. Creating the API as an object will be the best way to achieve this, therefore the solution must use good, and secure, Object Orientation techniques.

### 4.1.5 Extendable

Following on from the Object Orientation and the Testing section, the algorithm must be implemented in such a way that it can be expanded upon in the future. This involves extracting the API contract as a common interface with which the Algorithm implements, and then the current implementation of this being tested. This means that the implementation of the algorithm itself can be changed without the clients being affected.

### 4.1.6 Documentation

The API contract must be well documented such that clients know how to use the Algorithm, however the documentation needs to be done in such a way that the inner workings of the algorithm are not revealed.

### 4.1.7 Speed

One of the advantages of the algorithm is the speed it is able to run. Testing must be in place to ensure that the runtime of the algorithm is acceptable compared to the original implementation.

### 4.1.8 Portability

The original implementation of the Algorithm was done as a Visual Studio Console Application, this means that the project was limited to Windows or Visual Studio compatible Operating Systems. This is not optimal as we want our API to be usable from any device, therefore one of the requirements of the project is that the library must be portable.

There are not many alternative build systems to Visual Studio. The main alternative, and de-facto standard for C build systems, is CMake [10]. CMake is fast, portable and very powerful offering users any functionality they would need from a build system. However CMake can be very complicated to use, this is mitigated by the wide adoption and therefore vast amounts of documentation available for the library. For these reasons we will be using CMake as our build system.

## 4.2 API Contract

We can see an overview of the Alpex API Contract as a class reference in Appendix A.1. This documents the Alpex API interface which is how clients will be able to access the library. This reveals no information about the implementation of the algorithm except for how to interact with it, therefore this also works as the external documentation requirement of the algorithm.

## 4.3 Implementation

### 4.3.1 Version Control

Due to the security aspects of this project, it was decided that keeping the code off platforms such as github or gitlab would be in our best interests. That being said, it is not sensible to just have one copy of the code on one device. A number of solutions to this are available, one solution would be to periodically back up the code onto an external hard drive. Another solution is to periodically email an encrypted version to people involved in the project.

I decided to set up a local git server on a separate linux desktop. This involved turning the desktop into an SSH server, and then into a git server where I could initiate a git repository for the project. I could then connect to the git repository on my laptop by adding the desktop as a remote server, this gave me all the command line functionality of git. This technique is used for other projects within this report.

### 4.3.2 PIMPL

Pimpl[23][32] which stands for 'Point to Implementation' is a C++ abstraction for hiding the implementation of a class behind a pointer class acting as an interface.

A buffer exists between the implementation of the class (including the private methods and variables) and the class that is globally available. Separating the public and private methods like this breaks compile time dependencies, meaning that fewer '`#include ...`' are needed in the project reducing compile time. This also allows for the private code to be changed and updated without the client code being recompiled. Therefore, fewer compilations are needed and these compilations are faster.

The memory overhead of pimpl is an extra pointer, which may be a factor if we were constructing many small objects, however as we are likely to have 1 predictor, this is trivial.

The performance overhead is an extra pointer indirect per function call, due to the complexity of the algorithm adding this one extra indirect is indistinct.

The construction and destruction overhead is more noticeable, however as we are only likely to be creating one predictor, which we can reset, this is not a concern.

The closest alternative which achieves the separating of compile time dependencies is to use an OOP Factory, where users obtain a pointer to a lightweight abstract class with the implementations in the derived class. The trade offs between the two are negligible, however we have decided to use pimpl.

The way that our library will implement this idiom is seen in the header file, Listing 4.1, which shows the exposed class '`Alpex`' and the source file. The implementation of Alpex is defined in 'Alpex.cpp' and not visible from the header file. The '`Alpex`' object stores a '`unique_ptr (pimpl)`' to the implementation of the algorithm, then in the actual code 'dummy' functions are used that feed the input through the 'pimpl' pointer, returning the output.

Carefully chosen function names, constructor parameters and return variables within a pimpl idiom is how we have been able to achieve a secure, extendable, Object Orientated and fast library.

**Listing 4.1:** Code for the header file Alpex.h implementing the pimpl idiom.

```
// alpex.h

class Alpex {
public:
    void reset();
    int predict(int value);
    ...

    Alpex();
    Alpex(int initial);
    ...
private:
    class Alpex_Algorithm;
    std::unique_ptr<Alpex_Algorithm> pimpl;
}
```

### 4.3.3 Doxygen

To create a well documented API which is still extendable, we want a way to automatically document the API from our code. We can achieve this by using the Doxygen documentation tool[12]. We are able to use this documentation to generate the API Contract, however as this is a contract any changes must only be an extension to the document that was originally generated.

Doxygen searches through a given directory, in our case the 'includes' directory, and checks code for comments formatted in the Doxygen style to create pdf, latex and html documentation. This documentation can then be included into the project or shared publicly. If we only document the 'includes' directory then the documentation will not impact the security of the algorithm.

The full documentation of Alpex in this style is available in Appendix A.1.

### 4.3.4 Google Test

We are using Google Test to manage our test environment. Google test works by creating a test executable that we can use to run tests on our library. Inside our test source file, we define Google Tests in the format, `TEST(test_case, test_name) {...test code...} .`

Due to the deterministic nature of the algorithm there are a number of tests we have been able to implement:

- The simplest test case checks that the algorithm is able to learn a simple sequence [1, 2, 3], as well as the emergent property that the algorithm repeats itself for new values.
- The next tests check for incorrect input data and ensure that the library throws correct errors for these.
- Another type of test that we perform calculates metrics for predictions we generated with the original executable, and compares these predictions against our own ensuring our predictions are just as good if not better.
- We then have checks that can fail, these checks take sample input and output data from the original executable and compares the values like for like. The values should always be the same, however with the metrics check we allow this to fail to allow for future improvements.
- Finally we have tests to ensure that the algorithm is still fast, these can also fail as some machines may not have enough memory to be quick, the algorithm calculates the run-time on some sequences and checks that these are acceptable.

The tests allow us to ensure that the algorithm is still accurate in it's implementation as well as fast enough to make it compelling.

### 4.3.5 CMake

To ensure that the algorithm is portable we decided to use CMake as our build system.

Our `CMakeLists.txt` file defines how our library will be built. We can use the `add_library()` function from CMake to create our alpex library. CMake takes in configuration files which we use to inform users of the library which version they are currently using. This file also includes a description of the library as well as the requirements for it, which are none.

To build the project with CMake, all a user needs to create a build folder, call cmake on the project and then call cmake build. The commands are as follows:

```
mkdir build
cd build
cmake ..
cmake --build .
```

This will build the library inside a `bin` folder, users can then use this library using the Doxygen Documentation as a reference. Instructions on how to install the library are inside a

`ReadME.md` file.

On some windows machines using a Visual Studio compiler, the executable location can be `bin/{Debug/Release}`.

**C-Test**

CTest[13] is a feature of CMake which allows for us to combine a test environment inside our current build pipeline. We are able to add our Google Test executable to the build pipeline. As well as this we are able to copy over our test files from the test directory into the same directory as our test executable. We can then run our test executable to ensure our library passes our test cases.

I have also added support for the project to be built without testing, to do this the cmake command,

```
cmake ..  -Denable_test:BOOL=false,
```

should be used. This will often fix bugs with the google test library not being downloaded correctly.

## 4.4 Library Extensions

### 4.4.1 Supporting t+n Predictions

When the library was created great care was made to ensure the endpoints would support different prediction methods without actually implementing the methods themselves. This is seen with t+n predictions where we are able to write Pseudocode in the language of the API Contract to show how this can be achieved. Algorithm 1 shows the Pseudocode for making t+n predictions with the algorithm.

```
input  : An instance of Alpex, $alpex$
input  : A Value to predict from, $value$
input  : The number of predictions to make, $n$, where $n > 0$
output: A list of predictions from T+1 to T+$n$
begin
    $hst \leftarrow alpex$.train($value$) ;
    $arr \leftarrow$ List() ;
    while $n > 0$ do
        $p \leftarrow alpex$.predict_without_training($arr$, $hst$);
        $hst \leftarrow alpex$.add_value_to_history($hst$, $p$);
        $arr$.append($p$);
        $n \leftarrow n - 1$;
    end
    return $arr$;
end
```

**Algorithm 1:** Pseudocode for t+n predictions

## 4.4.2 Supporting Test Train Split Datasets

Often when evaluating a dataset we may want to seperate testing and training of the algorithm such that we do not train on the test set, this is not really how the algorithm is intended to work however it is another example of the expandable nature of the algorithm. Algorithm 2 shows the Pseudocode for how this could be implemented using the library.

```
input  : An instance of Alpex, $alpex$
input  : Dataset to train on, $dataset\_train$
input  : Dataset to test against, $dataset\_test$
output: A list of predictions for the test dataset
begin
    for $v \in dataset\_train$ :
        $hst \leftarrow alpex$.train($v$) ;
    $arr \leftarrow$ List() ;
    for $v \in dataset\_test$ :
        $p \leftarrow alpex$.predict_without_training($arr$, $hst$);
        $hst \leftarrow alpex$.add_value_to_history($hst$, $v$);
        $arr$.append($p$);
    return $arr$;
end
```

**Algorithm 2:** Pseudocode for a train test prediction split

### 4.4.3 Command Line Executable

As well as the library supporting its own simplistic User Journey, it is also important that the library also satisfies the original User Journey, this is what our Command Line Program 'alpex_executable' aims to achieve.

#### Design

To ensure backwards compatibility, with the original user journey, running the code without any parameters should make predictions on a file 'idata.dat', and output predictions to a file 'odata.dat'.

The user should be able to specify a specific file they want to predict for as well as the path of the output file.

Data should be available to be streamed into the program using standard input/output, this should support: entering all the data at the start, inputting the data one by one, and inputting the data after each forecast.

#### Implementation

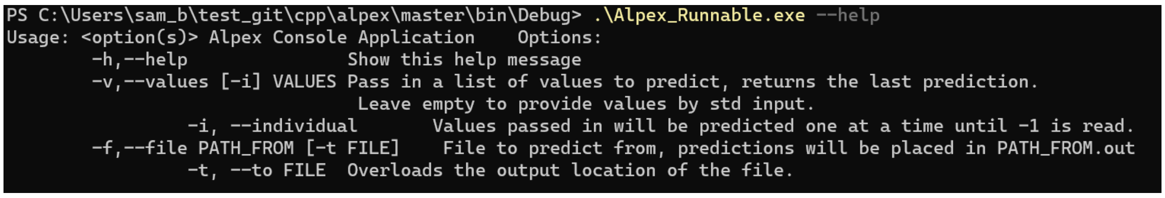
In C++ there are not many easy to use paramatisation libraries, therefore for simple parameters, it is easier to implement an argument parser ourselves.

The first argument we are going to use is the '–file', or '-f', argument which allows for users to specify the file that they want to predict from. This is accompanied by the '–to', or '-t', argument which writes the outputs to a specified path. These values are both defaulted to 'idata.dat' and 'odata.dat' respectively.

Instead of inputting data from a file, users should be able to input data directly from the command line. The '–values', or '-v', flag provides support for this. When the flag is followed by a list of values the algorithm will train on that list and return a prediction from the values.

If no values are provided then the program will listen for values from standard input. The algorithm will keep listening until '-1' is inputted and then return the forecast. If the '–individual', or '-i', flag follows the '–values' flag then the program will predict values as they are entered.

Figure 4.1 shows how to view the command line arguments by calling the help function on the executable.



```
PS C:\Users\sam_b\test_git\cpp\alpex\master\bin\Debug> .\Alpex_Runnable.exe --help
Usage: <option(s)> Alpex Console Application    Options:
        -h,--help               Show this help message
        -v,--values [-i] VALUES Pass in a list of values to predict, returns the last prediction.
                                 Leave empty to provide values by std input.
                -i, --individual        Values passed in will be predicted one at a time until -1 is read.
        -f,--file PATH_FROM [-t FILE]    File to predict from, predictions will be placed in PATH_FROM.out
                -t, --to FILE   Overloads the output location of the file.
```

**Figure 4.1:** An example of running the help command on the Alpex_Runnable executable.

### 4.4.4 Python Wrapper

Python is one of the most widely used languages for time series forecasting, this is partly due to the extensive suite of tools available. This makes Python a good choice for analysing the algorithm.

One way we could use Python with this algorithm would be to implement the algorithm in Python, which could be possible, however the algorithm takes advantage of C++ memory handling in its implementation which would need to be rethought, if we were writing it in Python. The solution to this is to create Python wrappers for the code which allow for us to directly call the C++ code from inside Python.

#### Design

There are two main ways that Python wrappers are made in C++ today [46], these are manually writing them or using a library to automatically generate them.

We are able to manually write Python compatible code inside C++, this involves writing Python callable functions that take in and return Python compatible data types. This has the advantage of being the easiest to setup, there is no external software needed to do this, however the trade-off is the complexity of the code needed.

One of the most popular libraries for automatically generating Python wrappers is Boost[7]. Boost is an extremely powerful tool that allows for C++ code to be called from Python and vice versa. It can generate wrappers for large C++ projects however it is extremely complex to download and implement.

Another library that we are able to use is pybind11 [35]. This library boasts itself as having similar capabilities and goals to Boost however being more simplistic and easier to use. The library does this by removing a lot of the tools and libraries that Boost depends on which are not needed in most situations or are only there to support legacy version. Boost has been around for a long time, therefore it does not take advantage of some newer C++ features that this software does.

For our project, the fact we are working with objects for our API means that manually creating the Python wrappers may be fairly difficult and will limit the expandability and maintainability of the code. However as we are only exposing one class in the API, Boost is probably 'overkill' for our requirements. Pybind11 strikes a nice balance and is therefore the library we will use for generating our Python wrapper.

**Implementation**

pybind11 works by users writing 'PYBIND11_MODULES' which act as a common interface for the C++ code. Due to our pimpl implementation, we are able to make a 'PYBIND11_MODULE' to mimic our 'Alpex.h' file. Appendix A.2 shows what our 'PYBIND11_MODULE' looks like. Inside our module we define our Alpex class, then our initialisers and finally the different functions available to interact with the object.

Another advantage of the pybind11 module is the cmake support available. Pybind11 has cmake extensions allowing us to add the wrapper to our cmake project, see Appendix A.3.

Python uses pip for its package manager, the cmake script is directly used by pip meaning that to install the Alpex Python library all we need to do is run 'pip install .' inside the project.

An example of using the Python library is seen in Listing 4.2.

**Listing 4.2:** Example Python code using the algorithm.

```python
import alpex_py

alpex = alpex_py.Alpex(1)

alpex.train(1)
alpex.train(2)
alpex.train(3)

assert alpex.predict(1) == 2
```

### 4.4.5 Project Architecture

```
./
├── alpex .................................................. Alpex Library Code
│   ├── include
│   │   └── alpex.h
│   ├── src
│   │   └── ...
│   ├── alpex.pc.in
│   ├── CMakeLists.txt
│   └── ReadMe.md
├── alpex_runnable............................Alpex Command Line Executable Code
│   ├── alpex.cpp
│   └── CMakeLists.txt
├── bin ............................................ Code Generated from CMake
│   ├── test_files
│   │   └── ...
│   ├── Alpex_Runnable.exe
│   └── Alpex_Test.exe
├── bindings
│   ├── pybind11.............................................PyBind11 Library
│   ├── Alpex.cpp
│   ├── CMakeLists.txt
│   └── setup.py
├── test_alpex.....................................Google Tests for the Library
│   ├── test_files
│   │   └── ...
│   ├── ...
│   └── CMakeLists.txt
├── CMakeLists.txt
└── ReadMe.Md
```

The complete architecture shows an overview of how the Alpex library project, including extensions, is structured. We can see that each extension has its own CMake file, this means that the different sub-projects can be ran independently of each other. We can also see that there is an overarching CMake file for the whole project, this combines all the different sub-projects such that you can work on the project as if it was all together. This removes the need to install the library or worry about dependencies.

# Chapter 5

# Time Series as a Service

Whilst conducting experiments into the noise tolerance of the algorithm, I was repeatedly running tests on large datasets and having to store the results of these predictions. Due to the memory requirements of the algorithm, offloading the processing to a server would have significant advantages, allowing future applications of the algorithm. This is one of the reasons for implementing Alpex as a Time Series as a Service. The TSaaS implementation aims to make Alpex deployable.

## 5.1 Technical Specification

This server needs to have all the functionality of the algorithm such that clients should not be aware they are using a server as opposed to the library directly. The main functionality we need to include is being able to create an instance of the algorithm that we can train on, and then make predictions with.

The server is aimed to relieve system resources on a clients machine therefore some performance overhead is deemed to be acceptable, however this must be carefully managed such that the algorithm is still fast enough to be useful.

The server must be scalable and follow modern Platform as a Service architecture patterns such as being RESTful and using micro-services.

The server should provide some of the higher level functionality of the algorithm, such as 't+n' predictions.

As well as being aimed as an aid to programmers, the server must also be usable on its own. Clients should be able to integrate the server and predictions into their own eco-systems, therefore it must be well-documented, secure, portable and expandable.

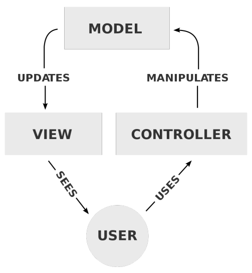


**Figure 5.1:** An example of how Model-View-Controller architecture works.

## 5.2 Design

### 5.2.1 RESTful Architecture

One of the requirements of modern day Platform as a Service implementations is that they need to be RESTful, this states that a web server:

1. Uses explicit HTTP methods
2. Is stateless
3. Exposes directory-like structure, URIs
4. Transfers only XML or JSON, common data structures

These are properties that our server must be required to maintain.

### 5.2.2 Model View Controller

Model View Controllers (MVC) is a design architecture for building micro-service systems that separates the view from the data. This is seen in Figure 5.1.

- The model manages the application data, logic and the rules of an application, this is known as business logic.
- The view is the presentation of the model, also known as UI Logic.
- The controller takes in the input and passes it to the model, carrying out validation and manipulation of the input, also known as input logic.

MVC separates the logic whilst maintaining loose coupling between them. Allowing for us to create self contained systems that can be “plugged into” any eco-system. For this reason we choose to use MVC within our project.

Modern implementations combine many different MVC architectures together into one. This is seen where the backend server acts as it’s own MVC having what is known as a ‘thin client’.

This has the advantage that all processing happens on the Server so that the user can access the data straight away. We decided to implement this functionality of the server acting as a thin client.

## 5.3 System Architecture

One of the key properties of the library that we must maintain is allowing users to instantiate an object, train using that object, and then make predictions. The problem with this is that the algorithm must maintain state to allow for clients to maintain this model. This directly contradicts our RESTful nature requirement.

There are a couple of solutions for this. One solution would be to send users the state of the algorithm and have them send it back every time they want to make a prediction. Considering the state is over 1.2GB, this is not feasible as well as not being secure.

Another solution is to have the state on the server backed up periodically meaning that although the server maintained state whilst it was operational, if it went down it could just restore it's state. This is not true statelessness and is not an ideal solution, the state is so large we cannot back it up that often without a significant performance overhead, therefore we are likely to lose data.

The solution that we came up with was to separate the server and the algorithm, such that the server maintained it's stateless property whilst the state of the algorithm was still maintained. This is done by creating a fully self contained web server for the algorithm that our main server would interact with. This means that if our main server were to go down then we could just boot up another one without their being any issue, however, if our Algorithm server were to go down then we would lose everything. This is a best fit solution as there is no real way for us to make it truly stateless, however in this way we are able to contain and manage our risk.

Figure 8.1 shows our system architecture with the separate C++ implementation and the main client facing server. From this diagram we can clearly see the MVC implementation, the client facing server acts as the controller, the model is the state machines and the view is returned to the user.

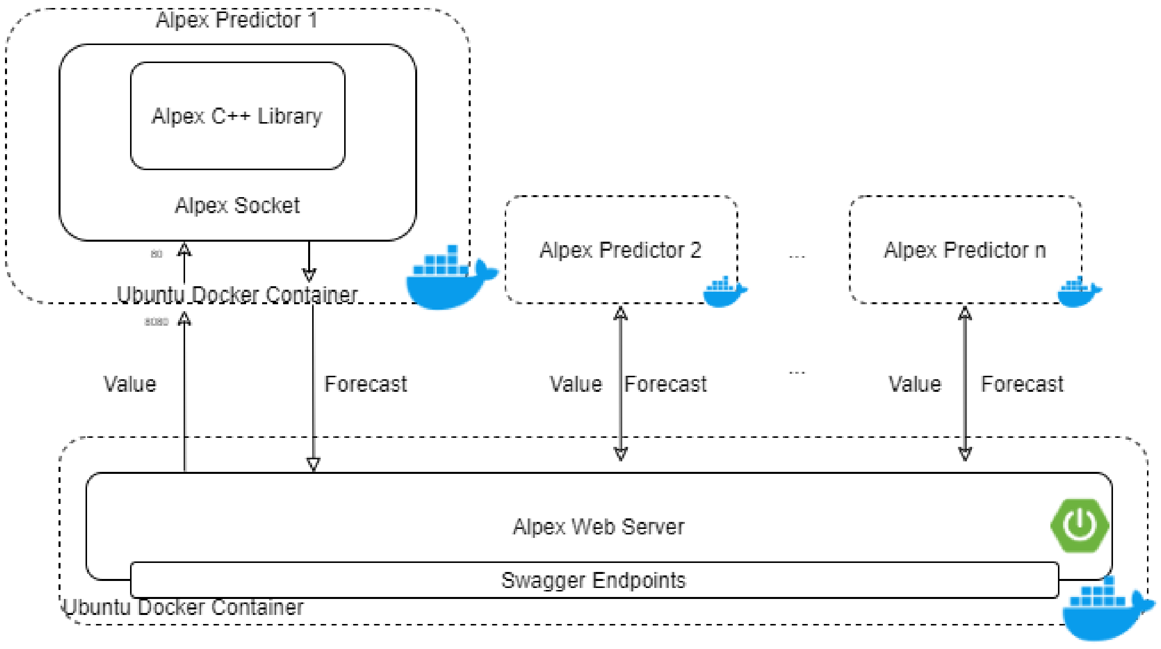


**Figure 5.2:** Alpex Time Series as a Service System Architecture.

## 5.4 C++ Sockets

As stated above, whilst trying to design a stateless Server, it became apparent that it would not be possible to run the algorithm on the server itself as we would want the algorithm to maintain state. Therefore we decided to implement a separate C++ Web Server that would listen for data from a socket and return predictions. This follows the micro-services architecture that is seen everywhere in modern computing, and is the motivation behind this section.

### 5.4.1 Technical Specification

The C++ Web Server needs to listen for input from a process and return a value to that process. As well as this it should maintain all the functionality that a client has by using the library directly. Like the C Library, this socket implementation should be as basic as possible, it should not include data validity or other higher level prediction methods; these should be handled by the client. The C++ Web Server should be seen as a way to interact with Alpex, and not a complete client facing server. This server must also be scalable and robust. With this being said, the C++ server implementation does need to be RESTful.

The server should be seen as a step toward distributing the algorithm therefore it must be cross platform and self-contained. The server should be easy to start up on different servers such that multiple instances can be maintained.

Finally although this implementation of the server will not handle data validity or security concerns, the server code should be secure enough to protect the Intellectual Property of the algorithm.

### 5.4.2 Design

#### Web Server Communication

As our Algorithm server is not aimed to be a Client facing server, we are able to look at some different methods for how the client facing server will interact with our C++ server.

The optimal method for large scale distributed projects, of this type, would be to implement Kafka event queues [2]. We would have a Kafka events queue separate from the servers where clients can post and listen to messages. We would have an implementation where our client facing server would post a value to predict for, our C++ server would listen for that value. Once the C++ server received the value, they would make a forecast and post that value to the events queue. The client facing server would then be listening for a response, once they have received the forecast they would return it to the user. This is very secure, expandable and would be the optimal way for a large scale project. In our situation we are likely to only have one server communicating with one predictor at any one time therefore this implementation is slightly overkill.

An alternative and the standard for local inter process communication is to use sockets. Sockets allow for processes to transfer data between them across ports, these can be open to a network as well as local to a machine. This is a slightly more simple approach however it has all the functionality that we need for our use case.

#### Socket Type

There are two main ways to send data through sockets in a network, these are either a connection orientated approach, or a connection-less approach. Each method has its own advantages and disadvantages in our scenario.

Connection orientated approaches involve clients setting up a connection with a server, sending data back and forth between the server and client until the client closes the connection. This is very useful when we have a client and server constantly communicating with each other, or when large amounts of data need to be sent. The disadvantage with this method is the overhead needed in setting up and closing a connection, which is not worthwhile for small connections. Furthermore if the connection were to be broken then problems can occur, for this reason we do not want to keep connections open indefinitely.

Connection-less approaches however involve a client sending a packet to a server which contains all the connection information needed for the communication as well as the data. The advantage of this method is that no handshaking is needed between the client and server this means that for small packages the overhead is much smaller. As the package is self contained, if data gets lost the package can just be resent instead of all the data needing to be resent; the individual speed of packages on a network is also generally faster. The disadvantage with this is that the overhead on each package sent is much greater as lots of added information needs to be sent per package. If users need to just send a few smallish packages at any one time then Connection-less is probably optimal.

For our scenario we are likely to send one value at a time from a client to a server. The fact it is likely to be the same client and server communicating is a plus for the connection orientated approach. As we are only going to be sending one package at a time at potentially long intervals means that we would likely have to set up handshakes fairly often. Furthermore the connections will be potentially open indefinitely which is not likely to be possible. For these reasons we have decided to go for a Connection-less implementation using Datagrams.

**Transport Layer Protocol**

There are a number of different protocols that we could use for a connection-less implementation. The most popular ones at the moment are User Datagram Protocol (UDP) and Datagram Congestion Control Protocol (DCCP). UDP however is the most common, well documented and robust method available. There are also implementations of UDP sockets in most common languages, C++ and JVM included.

**Containerisation**

Containerisation[11] is about packaging software and its dependencies such that it can be ran on any platform or technology. When a programmer builds software on a container they can be confident that the container will be able to run on any machine or on the cloud free of issues. Containerisation stems from Virtualisation which is similar however it uses virtual machines rather than containers, Containerisation offers many more advantages and is a reflection of progress away from Virtualisation.

Containers are lightweight, secure, portable, scalable and often contain a host of management tools. The most common technology for Containerisation is Docker [17]. Docker provides all the functionality to use containers, the containers are an abstraction at an application level and can be isolated from other processes. Docker is by far the industry standard for Containerisation and there are not many good alternatives to compete with it, we will therefore use Docker to containerise our web server.

### 5.4.3 Implementation

#### Command Line Arguments

The C++ Socket is a CMake command line program that takes in an optional port for the server to listen on, this port is defaulted to 27015. The program then starts an infinite loop that listens for data on that port until the server is killed.

#### API Contract

Although the C++ predictor server is not meant to be client facing we will still need a way for our main server to communicate with it. For that reason we have a small and simplistic implementation of an API contract.

Simply the way the server works is that a client sends a datagram containing the string representation of an integer, this compensates for the fact that different languages represent integers differently. Once the server receives the datagram, it makes a prediction on that integer, whilst training on it, and returns the string representation of the prediction inside a datagram back to the client.

As well as making basic predictions, the client is able to reset the algorithm and make predictions without training. To reset the algorithm, the client sends a datagram with the string `"r <initial_value>"`. This resets the algorithm with the initial value (defaulted to 0) and sends the client a datagram with a confirmation. To perform predictions without training, the client should send the string `"n <value>"`.

Finally the server allows users to ping it to check they are able to communicate, sending the string `"p"` to the server will cause the server to send a confirmation back to the user. This is a very basic contract and should only be used in conjunction with a server, and not as a client facing API.

#### Docker

Our C++ Socket implementation uses Docker to containerise the Web API. This containerisation involves building a Docker image and then deploying that image.

Each Docker container in a network is given a unique IP Address to communicate with it.

Our Docker build script loads in just the Alpex library and the Socket API into the Docker container. The container installs a valid C++ compiler, building the ‘SocketAPI’ executable. Port 8080 of the Docker container is exposed to UDP clients and connected to port 80 inside the Docker container. Finally the command for initialising the server is set, where we pass in port 80 for the API to listen on. The container is built using the command:

```
docker build --pull --rm -f "Dockerfile" -t socketapi:latest ".".
```

Once the container is built a user needs to deploy and run the container using the command:

```
docker run --rm -d socketapi:latest.
```

With a deployed container we can obtain the IP address by first getting the container name using `docker ps`, and then running the command:

```
docker inspect -f ‘range .NetworkSettings.Networks.IPAddressend’ container_name.
```

To communicate with the server and make predictions we must send UDP Datagrams to the containers IP Address on port 8080.

## 5.5 Client Facing Web Application

### 5.5.1 Language

There are a number of languages that we could use for our Client Facing Web App, each with their own advantages and disadvantages.

#### C#

C# is one of the most common languages used for Web Apps, as a result web application tools are extensively well documented and highly respected. This would have been a good choice of language to use, however it is becoming less popular than more modern methods and the tools and methodologies are becoming more outdated.

#### Python

Python is a very flexible language, there is not much you can’t do with python, therefore it is entirely possible to host python web servers. Python would have been a good choice for our server as we are using python for our analysis of the algorithm, which is going to be our servers main client. This is not necessarily a good thing however as if we know our client is going to be in Python then we may program specifically for that which will make the API less usable in the future. Furthermore, although there are tools available for this, they are not as well respected as ones for more standard languages.

#### JVM - Kotlin

Java Virtual Machine based languages are becoming increasingly more popular, as a result there is lots of new tools being created and used for the JVM. One of these tools is Spring Boot which handles most of the functionality we need from a Web Server whilst being easy to use and exceptionally well documented due to the wide industry adoption. Kotlin is a natural progression away from Java and is quickly overtaking Java as the main JVM language. Due to the functionality of Spring we shall use Kotlin for our Web API.

### 5.5.2 C++ Server Communication

The JVM has tools built in for Socket communication therefore we are using those to send and receive UDP Datagrams from the C++ Server. Our implementation of the algorithm has one main server that it sends data to and from. If the client does not specify information about the server then messages will be sent to and from the same, default, C++ Server. Clients therefore have the opportunity to specify their own IP addresses for Docker containers they want to communicate with.

### 5.5.3 Spring Boot

Spring Boot [39] is a large suite of tools designed for creating and managing Web Applications in JVM. Spring Boot contains it's own thin client Model View Controller implementation which handles most of the RESTful properties for us as well as Dependency Injection for Unit Testing and component swaps.

#### Dependency Injection

One of the key requirements of the server is that it is well tested and secure. We want a way to unit test sending messages through the socket to the C++ server without actually requiring a connection. The way we can do this is through the Dependency Injection architecture which is implemented by Spring Boot.

Dependency Injection means that objects define their dependencies only through constructor arguments. This means that we can pass in different implementations of the object at runtime. This allows for us to use mock versions of these dependencies during testing and mock responses. This allows us to create dummy responses for our socket so we can test that the server is robust without having to actually communicate with the C++ Socket.

One of the ways we have used DI in our program is to create an interface which looks the same as the API contract for the library. We then have an implementation of this interface which handles the socket communication. If in the future we wanted to use a different implementation instead of sockets, for example Kafka Event Queues, then all we need to do is create a new implementation of our predictor interface and change which implementation of the predictor we inject into our other classes. As we are using this common interface the client will not notice any change.

#### Spring Web MVC

The Spring Web MVC handles most of the RESTful properties of the server. The MVC provides an easy way to implement functions as HTTP methods. The MVC then exposes these methods inside a URI structure. Data that is passed into and returned from these functions is then converted to/from XML and JSON for the client.

#### Spring Security

The Spring MVC has a library for implementing a secure network connection which allows us to authenticate users of the server, making sure that all communication is secure.

### 5.5.4 Swagger

Swagger is a framework that provides a suite of tools for building, designing, documenting and consuming Web API's. Swagger has support for Spring Boot and provides features that we will take great advantage of for this project.

#### Swagger UI

The Swagger UI is a Spring Boot add on which creates a webpage and user interface for making API calls directly to our server. The library allows us to document our functions associated with our HTTP requests from inside our Kotlin code. This documentation is then read by the Swagger UI which displays it with our HTTP method calls.

The UI has two key advantages. The first is that it allows us to see exactly how to make API calls, it allows us to test the API with sample data so we can see how the server reacts to different inputs as well as the format of our output. This is extremely useful from a clients point of view as they are able to test their interactions with the server, it is also useful for programmers as it allows us to quickly test our application. The second advantage of the UI is that it allows us to export the view as our API Contract/Documentation, as the UI contains all the information about our endpoints (which we documented in our code) it acts as a perfect contract to give to clients.

#### Swagger Clients

Swagger Codegen is a piece of software that generates code for both clients and servers based on API contracts. The client code we generate allows us to call functions which abstract all of the HTTP request for the endpoint into native looking functions. An example of how we can use our Python client to interact with our server is seen in Appendix A.5.

### 5.5.5 Maven

There are two main build systems that are used with the JVM, these are Maven and Gradle. There is not much difference between the two, Gradle is the newer build system and is a natural progression away from Maven, however Maven has more backwards compatibility and a larger array of supported software. For this reason we have decided to use Maven as the build system for our project.

### 5.5.6 Docker

In the C++ section we saw the importance of Containerisation and how we can use Docker to implement this. The arguments we previously used for Docker apply just as much on the client facing server, therefore we also package this server inside a Docker container.

# Chapter 6

# Analysis Toolkit

One of the main focuses of this project is to analyse the forecasting capabilities of the algorithm, therefore it is important that we create a suite of tools to facilitate this; allowing Alpex to be accessible for, and empower, data analysts.

## 6.1 Technical Specification

There are a number of requirements that a tool suite will need for this project to be a success.

Each experiment will roughly follow a similar User Journey, this will consist of a number of key steps that are outlined by Figure 6.1.

From the diagram we see that most experiments will contain a way of generating a dataset, a way of predicting on that dataset, and then a way to analyse the predictions. This is what the tools must facilitate.

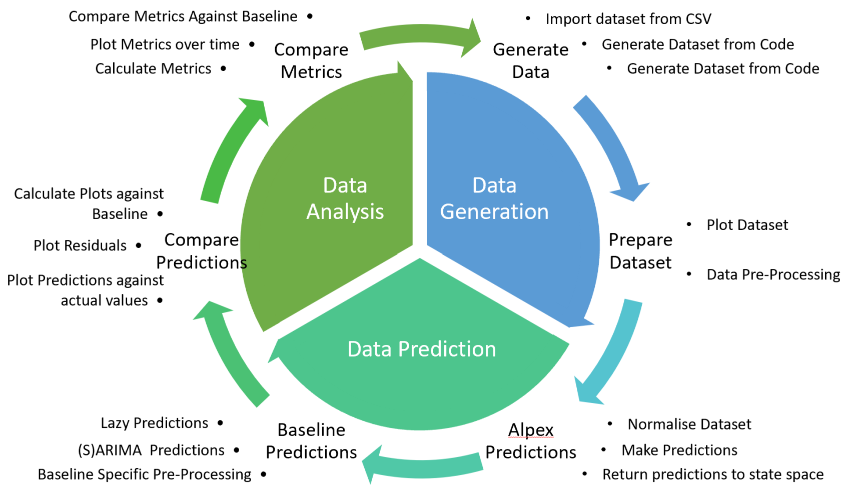


**Figure 6.1:** The User Journey for a typical experiment.

## 6.2 Design

### 6.2.1 Language Choice

The main choices when looking at data science analysis are Python, R and MatLab, each of these languages have their own advantages and disadvantages that could have made them a good fit for this project.

Generally when talking about time series, if we are looking at regression models (such as ARIMA) R is the preferred language, however when it comes to machine learning based methods Python is much more widely utilised.

#### R

The main advantage of R over Python is the focus on statistical and data analysis tools available natively in the language. R is designed for data analysis where as Python supports it through libraries, this is both a positive and negative. This means that we are able to fit our analysis tools into systems that are already using Python rather than having to keep the analysis tools separate.

#### Python

Python is an extremely flexible tool for data analysis due to the expansive suite of tools and libraries readily available. We will go into more detail about the tools we use for Python and the advantage each one brings later.

Python is very high level and generally human readable, this means that even non-programmers or programmers who have not used Python are able to understand the code. Compared to R and Matlab which are lower level, this means that the tools will be more usable.

In terms of performance, with the extremely widely used and well regarded numpy library, Python data analysis performance is comparable to R's.

For these reasons we have decided to use Python for the Analysis Toolkit.

### 6.2.2 Tools

#### Jupyter

Jupyter [34] is a web application which allows for users to create and share documents that contain a mix of markdown text and code. This allows for users to create shareable reports directly from their code instead of having to do this separately, drastically speeding up report writing time.

#### Matplotlib

Matplotlib [62] is one of the most well respected suite of graphing tools, allowing for practically any types of charts to be made; as the analysis will require a huge amount of graphs to be made, this can be extremely useful.

#### scikit-learn

scikit-learn[38] is a collection of statistics tools mimicking the functionality that comes natively to R. Sklearn combined with numpy allows for us to convert Python to a data analysis language. Some of the Sklearn features that we shall be using are:

- ARIMA and SARIMA models
- Seasonal and Trend Decomposition
- Time Series Statistics
- Plotting ACF and PACF graphs
- Splitting data between testing and training
- Metric Calculations
- Linear and Polynomial Regression Models
- And many more...

#### Numpy

Numpy [29] is a Python library written in C that aims to provide mathematical tools in Python with C level speed. This means that maths in Python is not restricted by the Python interpreter but instead can take advantage of the speed of compiled code. Numpy provides tools for array manipulation, mathematical functions (sin, cos, integration, exponentials...) and many more.

#### Pandas

Pandas [31] provides tools that are needed for data analysis in Python. This involves tools for reading and writing from in memory data to csv or other formats. DataFrames are indexed data structures that represent tables in an optimised way, time series optimised DataFrames exist. Tools for manipulating time series such as range generation, moving window statistics, data shifting, lagging, differencing and many more... Critical code is written in C to maintain speed.

#### Swagger Clients

As mentioned in the Time Series as a Service section, we can create Swagger Clients for different languages to allow for http requests to be wrapped in function calls. Swagger has support for Python, where we can automatically generate Python libraries from Swagger enabled servers (like our server). The generated repository includes extensive, automatically generated, documentation about how to interact with the library.

#### pmdarima

pmdarima [33] is an analysis toolkit that provides functions for ARIMA analysis. The main aim of this library is to simulate the 'arima.auto_arima' that R offers. This library performs a hyper-parameter optimisation script to try and estimate a best fit parameter set for ARIMA, effectively this is a non-parametric implementation of ARIMA.

#### Other Minor Tools

As well as these key tools that are listed above, there are lots of minor tools that we have used during the development of the project. These include

1. PyPDF2 [36] is able to perfrom PDF manipulation tools on different PDF's from inside Python, the main functions we shall be taking advantage is the ability to merge different pdf's together.
2. argparse [3] handles paramatisation of command line arguments for Python scripts, this allows us to easily pass arguments as well as generate documentation for command line programs using the '–help' command.
3. quandl [20] is a huge collection of financial datasets and time series, quandl has Python bindings that make collecting and manipulating these datasets extremely easy.
4. Pythons OS [30] library provides operating system independent functionality such as calling command line functions or moving and deleting files.

## 6.3 Implementation

### 6.3.1 Project Architecture

Figure 6.2 gives an overview of the project architecture for the Analysis Toolkit.

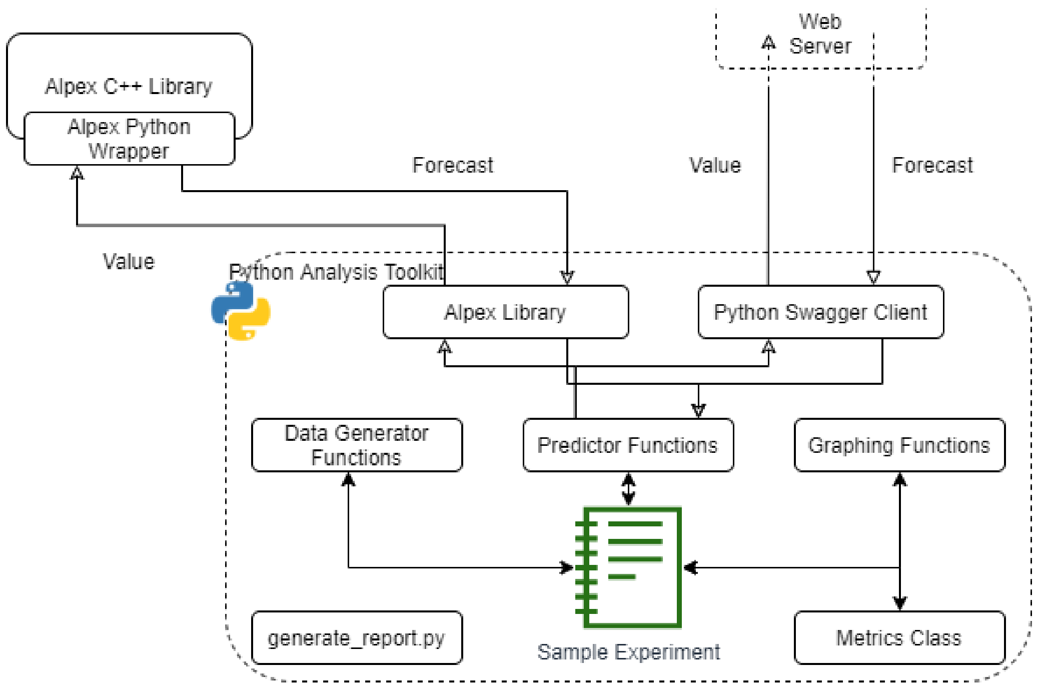


**Figure 6.2:** The project architecture for the Analysis Toolkit.

### 6.3.2 Metrics

The 'metrics.py' file contains a 'Metrics' class which is used to generate a large amount of different metrics comparing a list of predictions and the original dataset. The metrics object takes in a list of predictions and a list of original values as constructors. The class offers print functions to display the metrics as well as functions for displaying a list of metrics as a table, exporting a list of metrics to latex and plotting a list of metrics. Some of the metrics provided by the class are as follows:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)
- Median Absolute Percentage Error (Median APE)
- $R^2$ against mean ($R^2$)
- Explained Variance
- Max Error
- and more...

### 6.3.3 Generating Synthetic Time Series

There are a number of mathematical time series that we are likely to want to implement. For some of these standard series I have created generators that take in parameters and return the series. A list of the different series that are available is:

- Sawtooth
- Sine Curve
- Logistic Map
- Lorenz System
- Henon Map
- Mackey Glass
- Random Series

Each file/generator has the same function structure,

```
def generate_data_set({String → Any}) → PandaSeries,
```

meaning that all a user needs to do is import the correct function for the generator they want. An example of a generator is seen for the Henon Map in Appendix A.4.

### 6.3.4 API Implementations

At the moment we have two ways to make predictions, these involve using the Time Series as a Service or using the Python Wrappers that we created for the library. These API's are used to communicate with these two methods.

#### C++ API

The 'cpp_api.py' file is fairly simple, it provides constructors for the Alpex C++ Object. Once the functions are called to generate the object users can then interact directly with the object, therefore other functions are not needed.

#### Time Series as a Service

The 'network_api.py' file provides functions for communicating with an Alpex server. As discussed in a previous section, we are able to create a Python Swagger client for the server. This client wraps the complexity of http requests into native Python functions such that clients would not necessarily know that they were making server requests. This API file is an extension of this that includes error handling and simplifies usage by not requiring the user to make API instances. An example of one of these functions has been previously seen in Appendix A.5.

### 6.3.5 Predictor Code

The predictor code interacts with the different API's available to make different types of predictions. The library code is designed to be as basic but as expandable as possible, with this in mind, things like T+N predictions are not natively supported. This file implements different prediction methods as well as different normalisation strategies. A list of some of the different types of functionality this file provides is as follows:

- Predicting from a list
- Predicting individual values
- Splitting a dataset into train and test, and returning test predictions
- Predicting for a list of datasets with and without resetting the algorithm between
- Performing basic normalisation
- Performing tanh and sigmoid normalisation
- Performing all prediction methods with or without normalisation
- Performing t+n predictions
- Making predictions without training the algorithm
- Conducting these predictions on the normalised and different types of datasets.
- Removing trend from a time series by removing the rolling mean
- Removing trend from a time series by plotting a Linear or Polynomial Regression model
- And even more functionality...

The predictor has a flag in the file called predictor which a user can set to either 'LOCAL' or 'SERVER' which corresponds to making the predictions using the C++ API or using the server.

### 6.3.6 Graphing Code

As time series are represented as graphs, it is useful for us to have different ways of graphing them. The 'graphing.py' file acts as a wrapper to the many different matplotlib graphing tools available simplifying the parameters for our use case. There are two different types of tools available, these are for displaying the time series or for plotting metrics.

The different tools available for displaying the time series involve:

- Plotting the Predictions against the Time Series
- Plotting the Predictions against the Time Series and different baseline predictions
- Plotting a grid of predictions against Time Series
- Plotting Metrics over factors (for example sample size)
- Plotting Metrics in a grid
- Plotting Predictions against Actual and zooming into sections

### 6.3.7 Generating Reports

Our experiments are centered around creating Jupyter Notebooks that will form the report. One of the key advantages of using Jupyter notebooks is that we can mix code and markdown such that the whole report can be written in the notebook. Once we have this report we need a way to export the notebook into a more readable file type, usually a pdf. Jupyter makes this easy to do for one notebook as we are able to export notebooks to pdfs directly from the web app. However we often want to combine many different notebooks into one, and if we have to go into the web app for every notebook it can turn into an extremely long process. This is the problem the 'generate_report.py' function aims to solve.

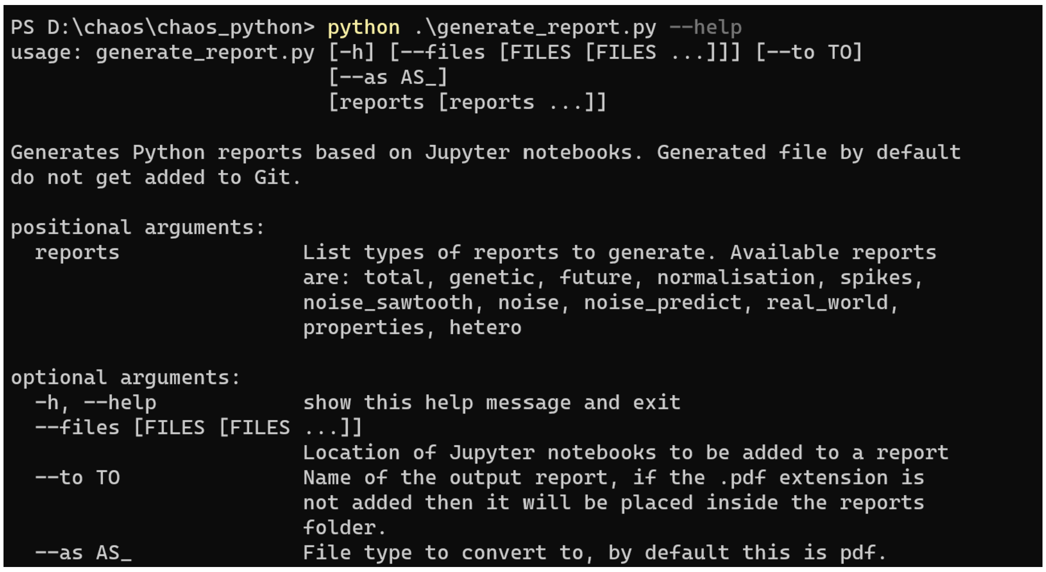
The 'generate_report.py' file is a command line program that converts Jupyter Notebooks into pdf's or tex files. The program can take in individual, or a list of, notebooks and convert them into the desired file type outputting them inside the reports folder.

As well as supporting list of files being inputted directly, which can make for very long scripts, users can update the Python file to add a predefined list of notebooks that form an experiment. Users can then type in the experiment name as a command line argument and the program will generate the respective report. An example of how to add a list of files for an experiment is shown in Appendix A.6 as well as the command for generating the experiment.

The process of converting Jupyter notebooks into other data types uses the 'jupyter nbconvert ...' function. This function takes a notebook as input, converts it to the correct type and then outputs it in the same directory. The problem with this is that it is slow to run, and for large experiments this script can take a long time.

Due to this, our command line program runs the conversion script inside a multi-threading environment. We spin up a process for each file in an experiment and then, once all of the pdfs (or other types) have been made, we run a script which combines the respective files. We are utilising a map-reduce architecture [54].

The reduction function is different for various file types. For merging pdfs we use the pypdf2 function to combine all the different files, we then use Pythons OS module to delete the files.



```
PS D:\chaos\chaos_python> python .\generate_report.py --help
usage: generate_report.py [-h] [--files [FILES [FILES ...]]] [--to TO]
                          [--as AS_]
                          [reports [reports ...]]

Generates Python reports based on Jupyter notebooks. Generated file by default
do not get added to Git.

positional arguments:
  reports               List types of reports to generate. Available reports
                        are: total, genetic, future, normalisation, spikes,
                        noise_sawtooth, noise, noise_predict, real_world,
                        properties, hetero

optional arguments:
  -h, --help            show this help message and exit
  --files [FILES [FILES ...]]
                        Location of Jupyter notebooks to be added to a report
  --to TO               Name of the output report, if the .pdf extension is
                        not added then it will be placed inside the reports
                        folder.
  --as AS_              File type to convert to, by default this is pdf.
```

**Figure 6.3:** An example of using the help function on the Python report generator.

An example of the commands available can be seen in Figure 6.3. We use the argparse library to generate the output to the help function as well as handling argument parsing for our program. An example usage of the generator is seen in Listing 6.1.

**Listing 6.1:** An example of generating the sample_experiment.pdf inside the reports folder with experiments sample and sample_2.

```
python .\generate_report.py
    --files .\time_series\sample\sample .\time_series\sample_2\sample
    --to sample_experiment
    --as pdf
```

## 6.3.8 ARIMA Baseline Predictions

For our project we are assessing how well the algorithm is able to forecast, however it is not possible to say how well we can predict unless we have a baseline to compare it against. We are already using the mean as a benchmark with our $R^2$ metric, however we also want to compare this with more complex methods.

The two forecasting methods we could use to benchmark against would either be ARIMA or a Neural Network implementation, for our project we are going to be focusing on an ARIMA baseline.

One of the key advantages that Alpex has over ARIMA is that it is non-parametric, this means that no specialist training is needed, furthermore for ARIMA models parameter tuning is a long and arduous process that doesn't add much benefit to the project as a whole. For this reason we are going to be using an automated ARIMA model, pmdarima, which automatically performs a hyper-parameter search to determine our parameters and then perform predictions.

The 'arima' folder inside the project provides all the necessary code to be able to build an ARIMA model and then fit it to our dataset such that we can compare our models against the ARIMA models. The ARIMA model library inside scikit does however have many limitations, for example we cannot diff a dataset more than twice, we cannot have an AR parameter of more than 3 and increasing the MA parameter drastically increases the memory requirement. For these reasons we are comparing the algorithm against a simple, non-parametric, version of ARIMA and not ARIMA in its entirety. The library also supports SARIMA for seasonal models.

### 6.3.9 Sample Experiments

The goal of the sample experiment is to allow for non-programmers or people unfamiliar with Python to be able to conduct experiments in the way outlined in the user journey, see 6.1. As well as allowing anyone to be able to make predictions the sample experiment allows for us to quickly analyse if a dataset is predictable. We offer a number of sample experiments, most notably a very simple implementation and a complete implementation.

All of the sample experiments follow a similar structure. This structure involves a notebook and an accompanying Python file. The accompanying Python file always includes a function for generating the dataset,

```
def generate_data_set({String → Any}) → PandaSeries.
```

The accompanying notebook imports this function, and then performs prediction operations on this. The file structure of our experiments is as follows:

```
./sample
├── data
│   └── data.csv
├── out
│   ├── alpex.obj
│   └── arima.obj
├── sample.ipynb
└── sample.py
```

#### Simple Implementation

The simple implementation is about allowing users to make predictions in the easiest and quickest way.

The simple implementation involves the 'generate_data_set()' function reading a csv file located inside the data folder, obtaining values for a specified column and returning them as a Pandas Series. The column name is defaulted to 'values' and the file name is defaulted to 'data.csv'. The user should either edit these values accordingly, in the Python file, or edit the file name and column name for their data.

The simple implementation also has functions 'generate_arima()' and 'generate_algorithm()'. The 'generate_arima()' function uses our ARIMA library to try and automatically generate predictions from the dataset. The 'generate_algorithm()' function performs a basic de-trend of the dataset it then passes the data through a basic scaled normalisation method, making the predictions on that dataset and then un-normalising the data.

When the client imports the Python file, both the generation functions are called and the outputs are written to pickle files 'alpex.obj' and 'arima.obj' inside the 'out' folder. The script then imports these pickle files, plots the dataset and predictions, for the baseline (ARIMA) and the algorithm. Metrics are then calculated for the respective predictions and displayed in a table.

The only process a user needs to do is copy their data into a file 'data/data.csv' and rename the column they want to predict for 'values', then run the notebook. No programming is required in this method.

**Normal Implementation**

The normal implementation is similar to the simple method, however it allows for the user to configure their predictions in different ways.

Like the simple prediction, the accompanying Python file contains 'generate_data_set()' and 'generate_arima()' functions. These operate in the same way, however the 'generate_data_set()' function takes in 'filename' and 'column' parameters.

The notebook then differs as the user imports the 'generate_data_set(filename, column)' function from the Python file and calls it with their respective values. From here, the notebook performs the de-trend. The notebook then makes the prediction on the dataset and does the same analysis on the predictions.

The difference with this method is the fact that the code involving alpex occurs inside the notebook instead of the Python file. The process is still accessible to non-programmers, however the user can perform their own data pre-processing on the dataset before making predictions.

An example of how a normal experiment looks is seen in Appendix E.1.

# Chapter 7

# Visualisation

The visualisation of the algorithm aims to make Alpex accessible to researchers looking to learn more about how Alpex makes predictions. Although there is not much we are able to discuss about our Visual representation of the algorithm we are able to go through our design decisions and show some examples.

The idea for visualising the algorithm came after a discussion between Ben and myself during one of our weekly meetings, where Ben had shown a naive tree representation of the algorithm that he had once created. This representation lacked scalability therefore, as the trees were exceedingly large, conclusions were not able to be drawn. This conversation however got us very excited about some different ways to represent the algorithm. Great curiosity about such a visualisation led me to wonder what could be done in this direction - therefore the extension was built just as much from a genuine excitement as well as weighing up the benefits that could be gained from analysing the representation.

## 7.1 Design

### 7.1.1 Motivation

Before this project we had next to no understanding of the structure of the algorithm and how this influenced predictions. We knew the algorithm had a constant memory complexity. In an attempt to gain an understanding of how memory was being used we decided that, as an extension, we should generate a way for us to visualise the algorithm.

We know that the algorithm involves large trees therefore we want a scalable way to display large and complex trees.

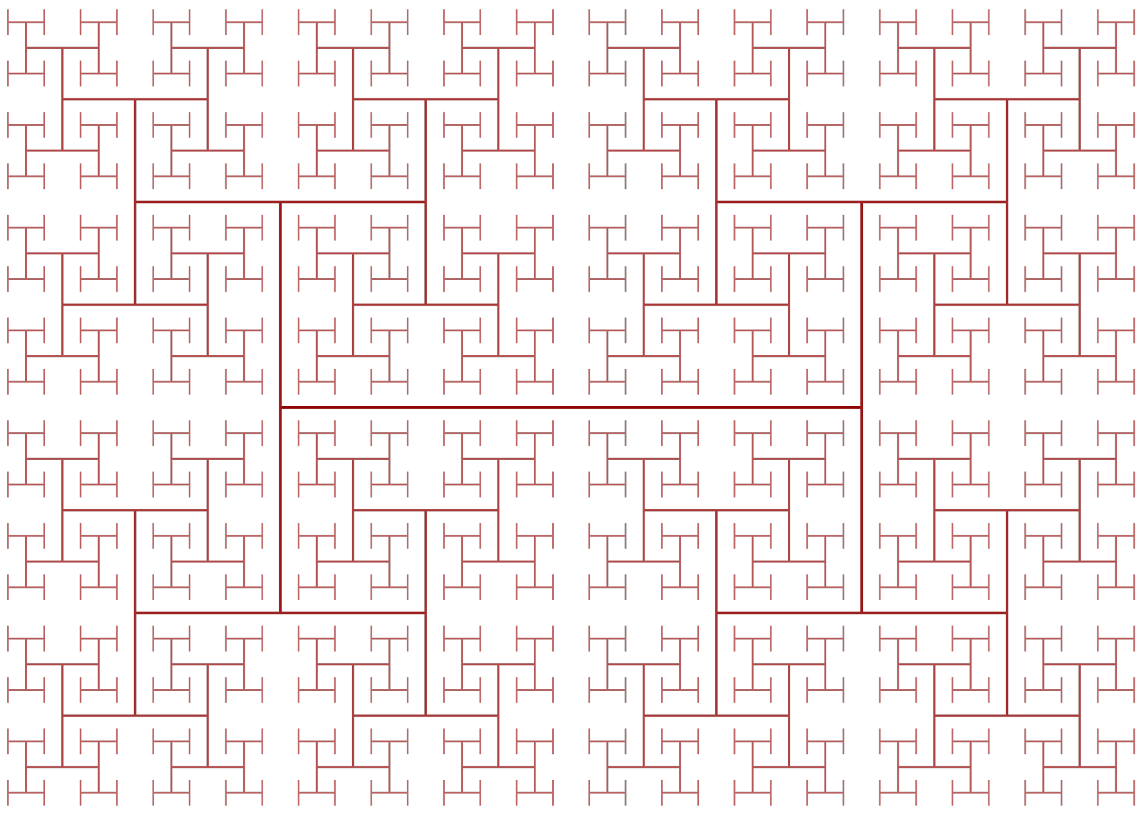


**Figure 7.1:** An example of a H-Fractal tree with a depth of 10 [53].

## 7.1.2 Displaying Trees

Following on from our initial meeting discussing a visual representation, we had a further meeting where we looked at some of the different possible ways of displaying large trees.

### H-Fractal Trees

Fractal trees are a robust way for generating large trees. This method of generating trees has been around for a long time and is definitely a compelling candidate. Figure 7.1 shows an example of a H-Fractal tree. The concern with these types of trees is that although they are good for large trees they probably won't scale to the size that we require.

### Circular Phylogenetic Trees

Phylogenetic trees have been used in biology to represent how species have evolved for a long period of time. These evolutionary paths are often very large and therefore these trees support the depth that we require for our trees. Figure 7.2 shows an example of a Phylogenetic tree being used in Biology. As this type of tree both looks very interesting and supports a large enough scale we decided to use this for our visualisation.

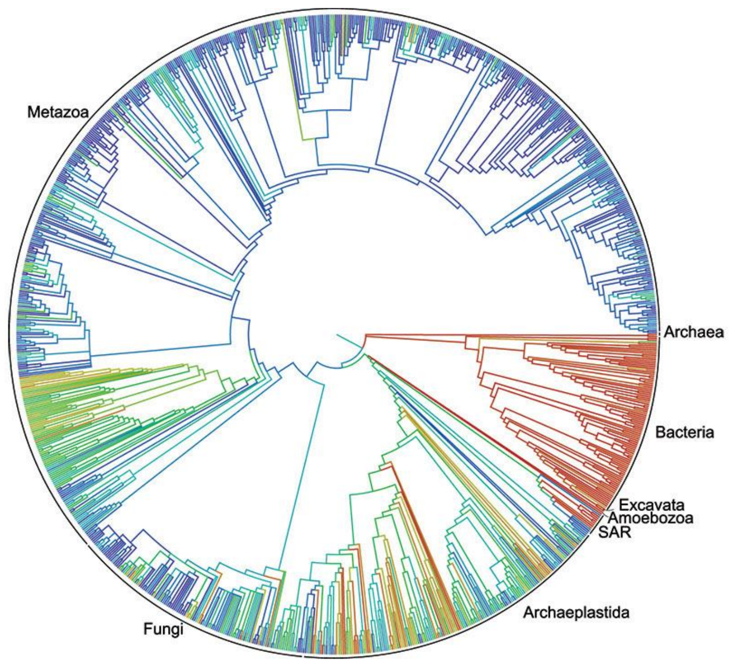


**Figure 7.2:** An example of a Circular Phylogentic Tree of Life [75].

**Other Representations**

There are lots of different ways that we could represent these large trees, if you are interested in looking at different methods then this paper [55] offers some really novel and interesting ideas.

## 7.2 Circular Phylogenetic Trees

### 7.2.1 C++ Library

Our current implementation of the C++ library would not, and should not, allow us to display a visualisation of the algorithm. Supporting this feature within our library would be a breach of security and as this is not a necessary feature that we need to support, it will not be added to the original C++ API.

Instead of adding the required code to support the visual representation we decided to fork the library and create a less secure version with an extra API call that would allow us to export the representation. This version of the library should not be distributed and should be used for experimental purposes only.

### 7.2.2 Tree Generation

There are a number of libraries available that we can use to generate these types of trees.

#### ete Toolkit

The ete toolkit[19] is a suite of phylogenomic tools for visualising trees in a biological setting. The library also has a python wrapper which is useful as it means that we are limiting the number of languages our tools are in. The library also contains all of the functionality that we need to represent the algorithm. The library also allows us to customise everything about the trees this will allow us to 'colour' the trees.

#### Phytools

Phytools is another tool that we could use for developing our trees. Compared to ete the documentation of Phytools is lacking, and the wrappers are written in R therefore when we have a tool like ete that works, I feel that Phytools is not the right approach.

### 7.2.3 Graph Colouring

Part of the way the algorithm learns is through graph colouring, the ete toolkit allows for us to create custom rules for generating the tree such that we can display this tree colouring.

As we are building very large trees, the colour scheme that we choose is extremly important. Ben and I iterated on this and eventually concluded that using the colours red(255, 28, 28) and blue(15, 15, 165) was the most effective.

### 7.2.4 Implementation

The implementation of the visualisation consists of a number of command line python programs that can be used to generate different types of trees. Scripts were also made to combine different images of trees together and, as the generation of these trees can take between 10-30 minutes, batch scripts were made to automate this process.

## 7.3 Alpex Visualisation

We show just three examples of the many trees generated for different mathematical series and this has allowed us to draw some interesting conclusions.

Figure 7.4 shows the Phylogenetic trees generated for the sawtooth example at different depths. We can see some beautiful patterns generated by the algorithm. We can conclude on this dataset that the trees actually have quite a lot of redundancy.

Figure 7.3 shows what a tree looks like for a chaotic time series, the logistic map. The interesting conclusion that can be made from this is how detailed the tree is. The fact the tree

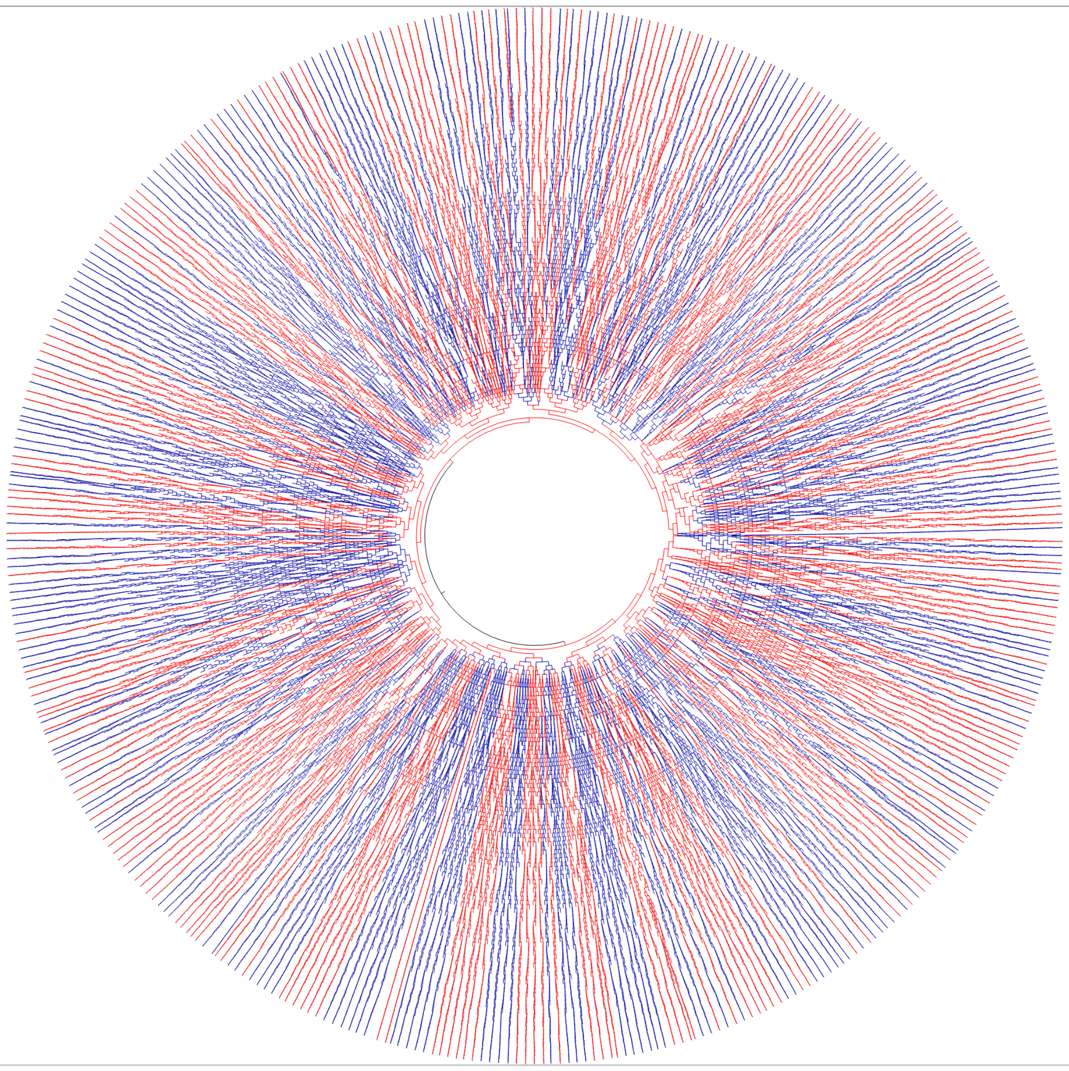
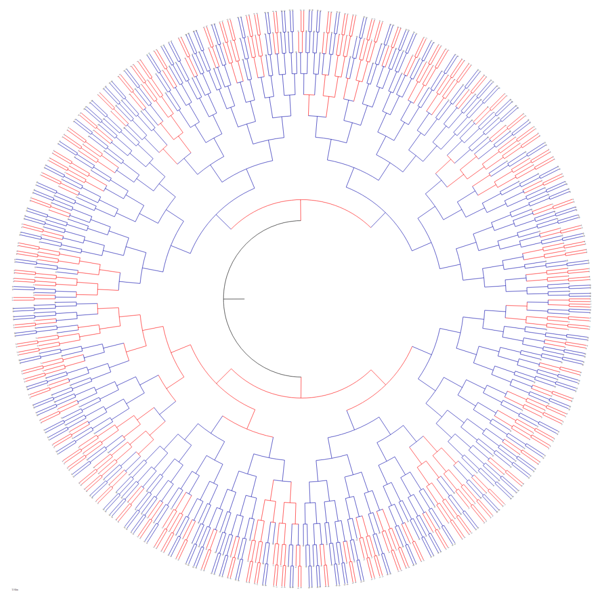


**Figure 7.3:** An example of a tree for the Logistic Map dataset trained on 410 values with a max-depth of 100.

is so detailed actually tells us that the algorithm needs the complete tree to be able to make good predictions and therefore there may not be much redundancy.

Another interesting note relates to the file size of the images, the sawtooth image with a 100 depth is 766KB where as the logistic map is 2.26MB and we were unable to even calculate it for a random series, at this depth, as the image would just be too big. The images were done at such a resolution that, on the original versions, we are able to fully zoom into the tree without any loss of accuracy.



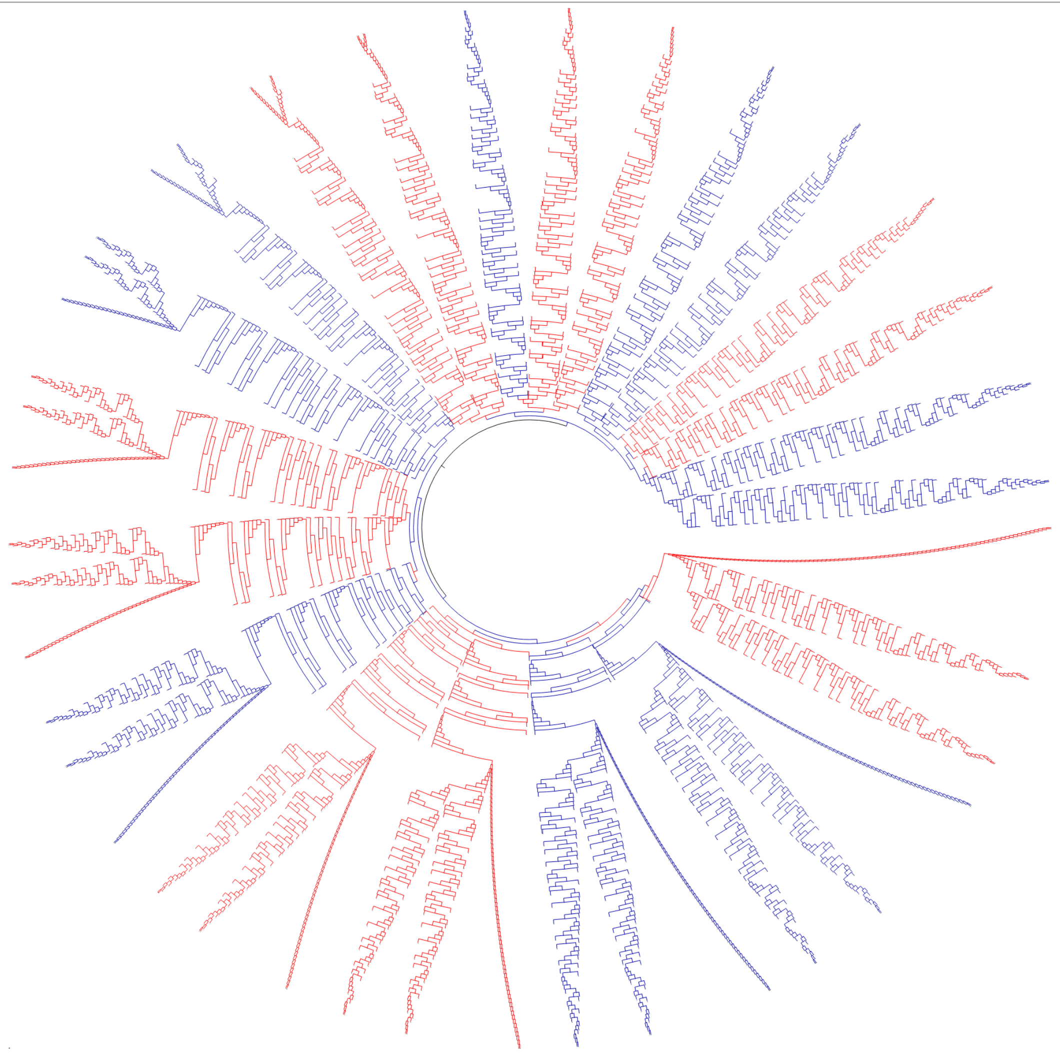


**Figure 7.4:** An example of a tree for the Sawtooth dataset trained on 4096 values with a max-depth of 10 (Top) and trained on 410 values with a max-depth of 100 (Bottom).

# Chapter 8

# Evaluation

The aim of the first part of the project was to answer the question "How can we make Alpex accessible and deployable?". We have achieved this by: creating a C++ library with an object based API to make the algorithm accessible for developers, deploying a Time Series as a Service implementation to provide forecasts for clients, developing an Analysis Toolkit to empower data analysts and visualising the algorithm to allow researchers to study it's properties. We can see how these components interconnect in our complete system architecture, Figure 8.1.

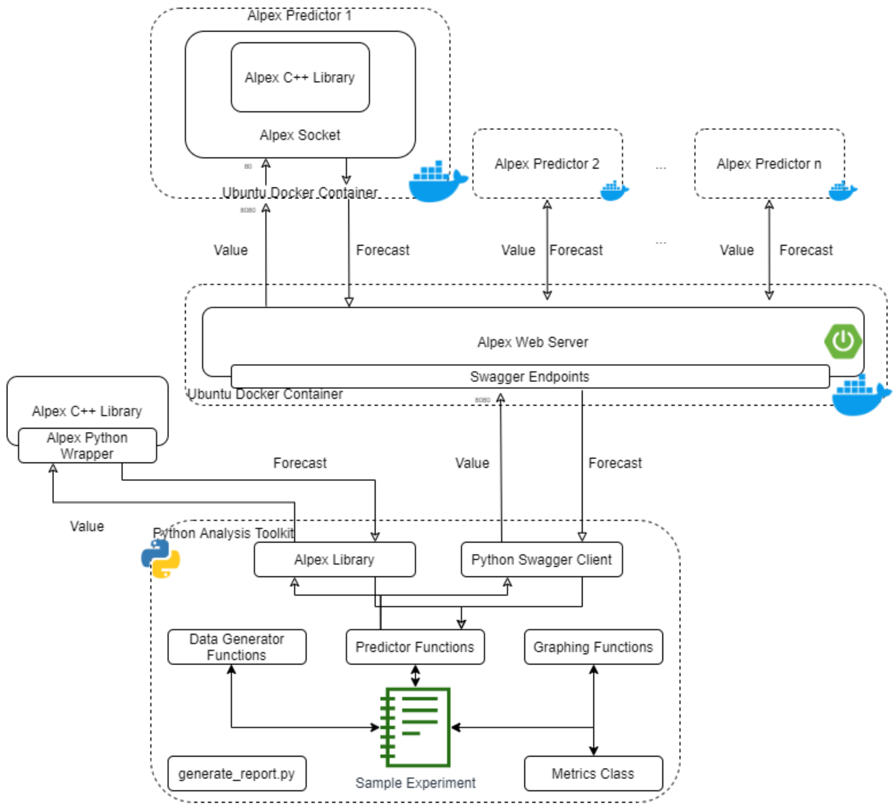


**Figure 8.1:** Complete Architecture for the project.

## 8.1 Alpex C++ Library

The success of this library revolved around creating the most simplistic implementation of the algorithm which was: secure, extendable, fast, documented, well-tested and Object Orientated. Through our library we have been able to achieve this.

The library is Object Orientated and the object is well-tested, documented and extendable as well as being usable and secure. The object was also designed in such a way that there is a minimal amount of endpoints whilst still being fully usable, in the simple use case as well as providing different types of predictions such as T+N. Using C-Test we are able to ensure this object is well tested.

As opposed to the original implementation of the project, the C++ API takes advantage of CMake to make it cross-platform. This means that the library can be ran and built on any machine with an appropriate C compiler, and not just using Visual Studio. CMake also offers possibilities for the library to be used inside build pipelines and the unit tests present, ensure the library still works.

The library itself was developed in two vertical slices, one limiting factor of this was that the first slice contained a memory leak. As I had not ran performance checks on the original implementation at this point it was not until near the end of the project that I found this leak, once fixed, runtimes went from minutes to seconds on very large datasets. If I had discovered this at the end of the first vertical slice, then some computation time could have been saved.

As well as the base requirement of the API, the API was extended for Python allowing for it to be used within data analysis tools. The Python API is extremely easy to both set up and use.

## 8.2 Time Series as a Service

In this section we have outlined the need for and the implementation of Alpex as a Time Series as a Service. This implementation was largely successful in its goal of offloading system resources from my laptop such that I was still able to use it whilst experiments were being completed. We were also able to deploy an implementation of how the Algorithm could fit into an ecosystem as a micro-service. This is likely how we would implement the algorithm into a real world scenario, therefore a useful contribution to the project.

The containerised nature of both servers allows for our implementation to be scalable, portable and secure. The fact a client is able to select the instance of the Docker container they want to predict for allows for our Client facing server to support multiple clients. As the Client facing server is stateless, if we run out of resources due to supporting too many clients, we are able to spin up more machines or increase the capacity of our current machine to scale our server with demand. One of the key properties of the algorithm is the constant space complexity. This means that the C++ Socket containers, are inherently scalable.

Although great care was taken to contain the risk of the system, by separating out the C++ Socket, if we wanted to actually implement this in a real world situation we would need to mitigate these risks by having backup solutions for the algorithms state. It would be an interesting balancing act to see how this could be implemented whilst still maintaining the security of the algorithm.

Docker has a JVM wrapper which allows for us to call Docker functions from inside Java code. A limiting factor of this web server is the requirements for users to spin up the predictors separately. A useful extension to this Web Server would be to allow for this to be done using the client facing web app using the Docker wrappers, potentially setting up a new predictor for each new client.

Furthermore if we were implementing this as a system where predictions would automatically cause an action to occur, we would want to implement checks on the server to ensure that our predictions are sensible, and calculate error metrics or probabilistic models.

## 8.3 Accuracy Analysis

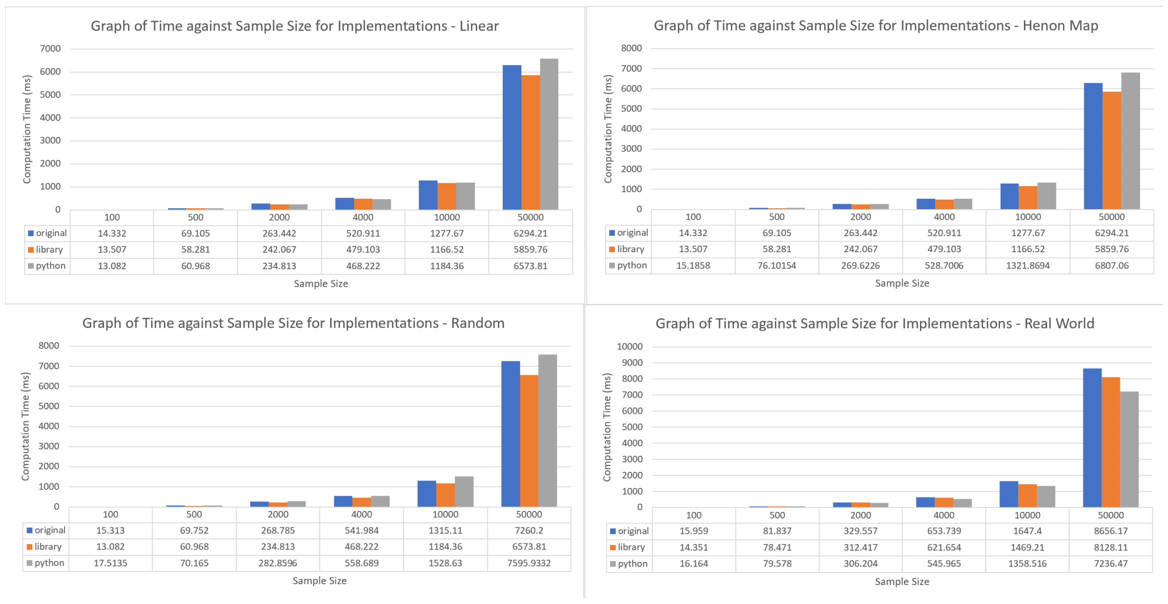
Our test cases allowed for us to check the accuracy of our predictions against the original implementation.

Although we already had test cases to do this, to be absolutely sure that values were identical a two factor authentication process was created between Ben and I to ensure there was no difference. Ben sent multiple excel spreadsheets to me which contained the input and output of the algorithm for multiple different series. I was then able to take the input from each spreadsheet, generate the corresponding output and run his output and my output through a diffing tool to ensure they were identical. Once this had been completed I created an identical excel spreadsheet with my input and my output which I was able to send back to Ben and he could verify with his implementation.

The manual two factor authentication method as well as the test cases meant that we could be certain that the accuracy of the library was consistent.

## 8.4 Complexity Analysis

One of the success criteria of this library was that it was just as fast as the original implementation, therefore any change in computation speed or complexity needs to be explainable. Figure 8.2 shows the change in speed between the original implementation and the library



Linear

| | 100 | 500 | 2000 | 4000 | 10000 | 50000 |
|---|---|---|---|---|---|---|
| ■ original | 14.332 | 69.105 | 263.442 | 520.911 | 1277.67 | 6294.21 |
| ■ library | 13.507 | 58.281 | 242.067 | 479.103 | 1166.52 | 5859.76 |
| ■ python | 13.082 | 60.968 | 234.813 | 468.222 | 1184.36 | 6573.81 |

Henon Map

| | 100 | 500 | 2000 | 4000 | 10000 | 50000 |
|---|---|---|---|---|---|---|
| ■ original | 14.332 | 69.105 | 263.442 | 520.911 | 1277.67 | 6294.21 |
| ■ library | 13.507 | 58.281 | 242.067 | 479.103 | 1166.52 | 5859.76 |
| ■ python | 15.1858 | 76.10154 | 269.6226 | 528.7006 | 1321.8694 | 6807.06 |

Random

| | 100 | 500 | 2000 | 4000 | 10000 | 50000 |
|---|---|---|---|---|---|---|
| ■ original | 15.313 | 69.752 | 268.785 | 541.984 | 1315.11 | 7260.2 |
| ■ library | 13.082 | 60.968 | 234.813 | 468.222 | 1184.36 | 6573.81 |
| ■ python | 17.5135 | 70.165 | 282.8596 | 558.689 | 1528.63 | 7595.9332 |

Real World

| | 100 | 500 | 2000 | 4000 | 10000 | 50000 |
|---|---|---|---|---|---|---|
| ■ original | 15.959 | 81.837 | 329.557 | 653.739 | 1647.4 | 8656.17 |
| ■ library | 14.351 | 78.471 | 312.417 | 621.654 | 1469.21 | 8128.11 |
| ■ python | 16.164 | 79.578 | 306.204 | 545.965 | 1358.516 | 7236.47 |

**Figure 8.2:** Graphs of Computation Time (ms) against sample size for multiple datasets using the different implementations.

implementation.

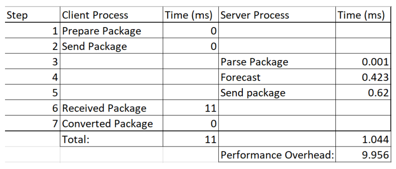
To ensure consistency between the tests, start and end time flags were placed before reading the first value from a file and after the last prediction was made. A quick modification to the original code allowed for it to be ran on a Ubuntu machine therefore all tests were completed on the same computer. The specification of this machine is: i5-9600K x86_64 CPU @ 3.70GHz with 16GB 3000MHz DDR4 memory running Ubuntu Elementary OS. The computation times are for a Linear Series, a Chaotic Series, a Real world time series and a random series at the same data size intervals (100, 500, 2000, 4000, 10000, 50000).

### 8.4.1 C++ Library

From these diagrams we can see that the difference in computation time between the original implementation, the new executable and Python wrappers are negligible. In terms of CPU usage, it was impossible to tell without causing major overhead for the smaller samples, however on the large datasets 100% of one core was being used as well as 1.23GB of RAM.

### 8.4.2 Time Series as a Service

What we saw from our experiments is that the performance overhead of the server is significant. To better understand how this would affect forecasts, I conducted an investigation to



| Step | Client Process | Time (ms) | Server Process | Time (ms) |
|---|---|---|---|---|
| 1 | Prepare Package | 0 | | |
| 2 | Send Package | 0 | | |
| 3 | | | Parse Package | 0.001 |
| 4 | | | Forecast | 0.423 |
| 5 | | | Send package | 0.62 |
| 6 | Received Package | 11 | | |
| 7 | Converted Package | 0 | | |
| | Total: | 11 | | 1.044 |
| | | | Performance Overhead: | 9.956 |

**Figure 8.3:** A table analysing performance overhead of our Time Series as a Service implementation.

find out where the bottleneck of the system was. To do this I calculated the average computation time for each part of the forecasting pipeline, see Figure 8.3.

What we can see from this table is that the computation time on the client facing server is spent sending a value and receiving a response from the predictor. We can see that most of the predictors time is spent forecasting and that this it is extremely quick. If we take the total time the client facing server has to wait and subtract that by the time it takes for the predictor to make a forecast we can estimate the performance overhead, this estimate comes to about 9.955 milliseconds.

Although there is a substantial amount of overhead per value for the TSaaS implementation compared to a locally ran version of the algorithm 9.955 ms compared to 0.423ms, if we compare this to other forecasting methods it is much quicker. See Section 11.2.4.

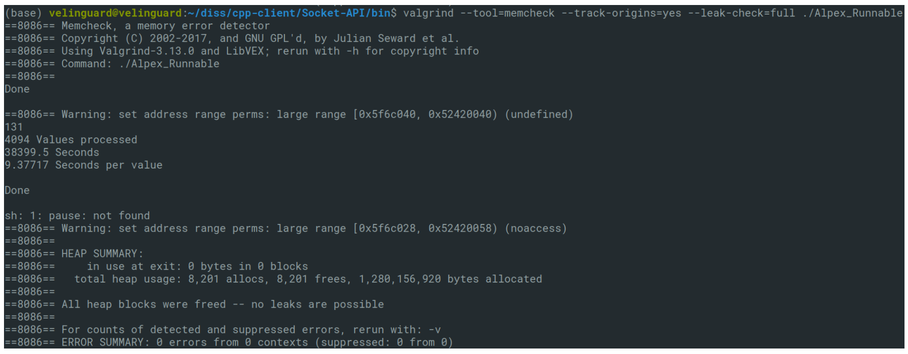
In terms of CPU usage, the Docker Container is using 100% of one of the cores, for large datasets, as well as 1.23GB of memory. One of the interesting things we are seeing is that the server is using up 100% of it's bandwidth, therefore it would appear that the network speed could be the bottleneck. The system resources being used by our client PC to make predictions is negligible, this is a huge improvement over the other implementation.

From our experiments we calculated the upper runtime limit as being 14ms, if we assume that in some situations the runtime may go above this and set it as 20ms, this means that **we are able to predict and train on values arriving at 20ms intervals, from any device!**

Due to the runtime overhead of the Server, the TSaaS implementation is more applicable in situations where we have constantly arriving data; whereas in contexts where we already have all of the data and want to make a forecast on that, directly using the C++ library would be the optimal strategy (resources allowing).

## 8.5 Valgrind and Memory Checks

As we are working with C we must ensure that the code is leak free and that all values in memory are eventually freed. Figure 8.4 shows the output from running the memory checking tool Valgrind, where we are able to see that the algorithm is leak free.



```
(base) velinguard@velinguard:~/diss/cpp-client/Socket-API/bin$ valgrind --tool=memcheck --track-origins=yes --leak-check=full ./Alpex_Runnable
==8086== Memcheck, a memory error detector
==8086== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8086== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8086== Command: ./Alpex_Runnable
==8086==
Done

==8086== Warning: set address range perms: large range [0x5f6c040, 0x52420040) (undefined)
131
4094 Values processed
38399.5 Seconds
9.37717 Seconds per value

Done

sh: 1: pause: not found
==8086== Warning: set address range perms: large range [0x5f6c028, 0x52420058) (noaccess)
==8086==
==8086== HEAP SUMMARY:
==8086==     in use at exit: 0 bytes in 0 blocks
==8086==   total heap usage: 8,201 allocs, 8,201 frees, 1,280,156,920 bytes allocated
==8086==
==8086== All heap blocks were freed -- no leaks are possible
==8086==
==8086== For counts of detected and suppressed errors, rerun with: -v
==8086== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

**Figure 8.4:** Output for running Valgrind on the alpex_runnable executable.

## 8.6 Analysis Toolkit

In this section we have outlined the Analysis Toolkit that we will be using to conduct our experiments on the Alpex Algorithm. This toolkit offers:

- A suite of tools for making different types of time series forecasts, such as T+N or predicting without training. This suite supports local implementations of the algorithm as well as communicating with a Swagger Enabled Server.
- Tools for graphing datasets and forecasts such that we can visually compare and analyse our model.
- An object for generating a variety of metrics for our model such that we can mathematically analyse our predictions.
- An array of sample experiments that allow for non-programmers to conduct experiments using our toolkit as well as users being able to conduct experiments quickly.
- A command line program for combining many different experiments, or just one, into reports in the form of pdf or latex that can be shared.
- An implementation of a non-parametric ARIMA model that users can use to benchmark their experiments against.

All of these tools allow any user to be able to complete our User Journey for an experiment, whether that is using a sample experiment or by using the tools that we have provided. Therfore I believe that the toolkit was successful in providing the tools needed to complete this project. This toolkit has been extremely successful in quickly analysing a wide variety of datasets to effectively analyse the limitations and properties of the Alpex library.

Throughout the project we held weekly meetings where we would go through the interesting experiments that I had conducted during the week, in the format of the reports generated by this toolkit. The quality and expansiveness of the reports are one of the key success factors for this project as we were able to analyse and discuss these reports quickly. Furthermore all of the graphs seen in the analysis sections were directly lifted from these reports or generated using the toolkit. Due to this I believe the toolkit was a success for the Algorithm and the project.

If we were to expand the tool kit, we would want to add more benchmarks. Currently we only look at an ARIMA implementation to benchmark the algorithm, however it would be useful to explore other methods, such as Amazons Time series as a Service, or machine learning models. As the project was mainly focused on properties of this algorithm and not other algorithms this was not a top priority of the project.

One limiting factor of the client focused agile approach taken for the development of the toolkit is that there is not much consistency in the experiments throughout the project. This means that some of our earlier experiments are of a fairly significantly lower quality.

One of the limitations of the report generation script is that the title of the report is the name of the notebook and I was unable to find a way to change this. Our sample experiments will often keep the name ‘normal’ or ‘simple’ for their notebooks, that means that a lot of the Experiment titles are ‘normal’ or ‘simple’.

# Part II

# For which classes of time series is Alpex most effective?

# Chapter 9

# Synthetic Analysis

In our Analaysis Toolkit section we outlined some of the different mathematical functions that we created generators for. These datasets are a mix of simple mathematical functions, such as sawtooth or sine curves, and complex chaotic series, such as the Henon-Map and Lorenz System. We shall therefore be looking at classes of time series with a known ground truth to assess Alpex's effectiveness in this context.

These mathematical series will form the basis of our synthetic investigation. Figure 9.1 shows the different series that we shall be analysing. To replicate these experiments the following base parameters are listed:

1. Sawtooth
2. Cos
3. Logistic Map with $r = 3.8$ and an initial value of $0.5$
4. Mackey-Glass
5. Lorenz System with $\rho = 28.0$, $\sigma = 10.0$, $\beta = \frac{3}{8}$ and an initial value of $[2.0, 5.0, 20.0]$.
6. Henon Map with $\alpha = 1.4$, $\beta = 0.3$ and an initial value of $[0.2, 0.2]$.

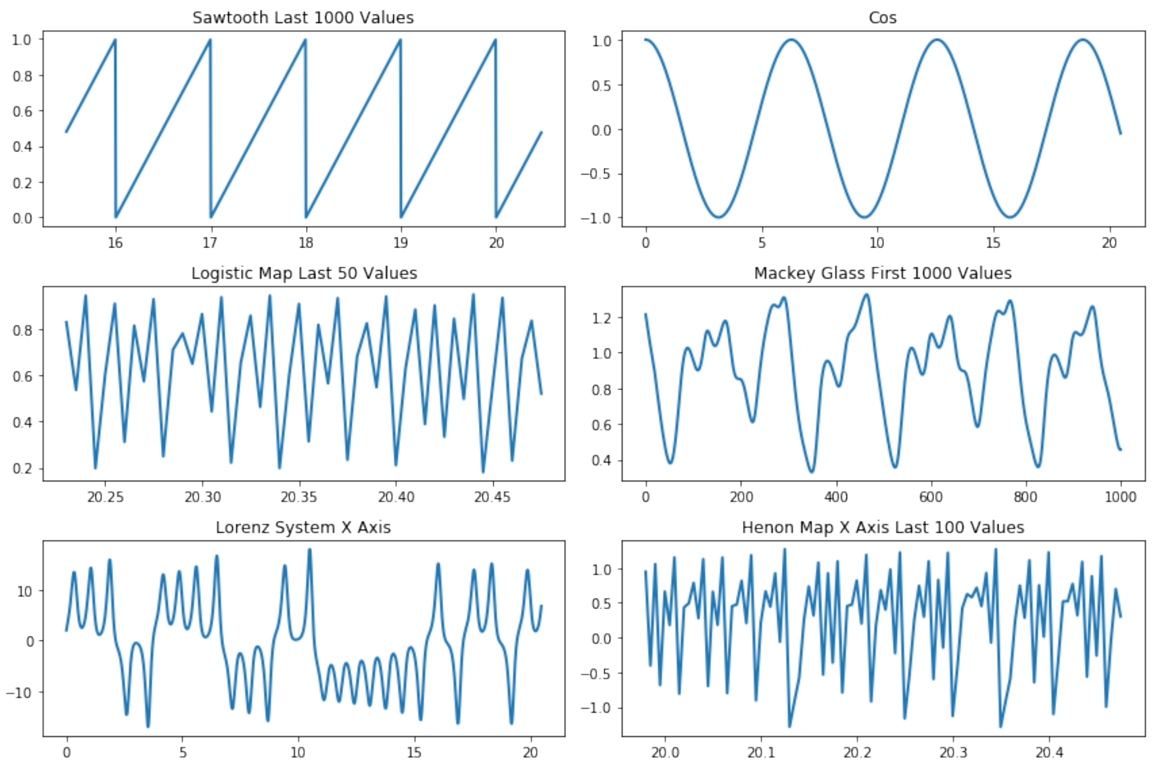


**Figure 9.1:** Examples of the datasets used for our Synthetic Investigation.

## 9.1 Chaotic Time Series Analysis

As previously mentioned, chaotic time series are characterised by non-linear dynamical systems typically described by relatively simple equations, which are deterministic in nature, yet are capable of producing unpredictable and divergent behaviour. Non-linearity in the context of a feedback process results in a system which displays sensitive dependence on initial conditions implying that small changes in any value may produce wildly different results. This makes Chaotic time series incredible difficult to predict.

From our original excel spreadsheets, we are able to see how the algorithm seems to learn these chaotic time series at an unprecedented speed. The excel spreadsheets only allow us to hypothesise about this therefore we need to conduct a more thorough investigation into these different time series, comparing them to baseline methods. From this we will be able to confirm, or deny, the ability of this Algorithm in learning Chaotic Time Series.

Although we perform an analysis into all of our mathematical datasets, we are going to focus on the Henon Map and Mackey-Glass series in this section as they are fairly different in terms of their structure and are good examples of how the algorithm deals with chaos.

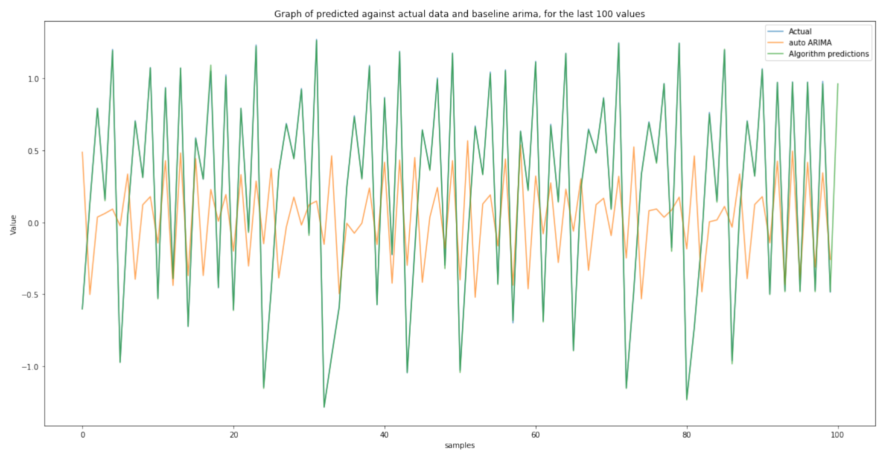


**Figure 9.2:** Graph comparing Alpex's predictions and ARIMA's model against the original dataset for the Henon Map.

### 9.1.1 Henon Map

The Henon Map is an example of a multi-dimensional Chaotic Time Series. The Henon Map is defined as:

$$p_{n+1} = \begin{cases} x_{n+1} = 1 - a * x_n^2 + y_n \\ y_{n+1} = b * x_n \end{cases} \tag{9.1}$$

This means that we can plot either the X axis or the Y axis against time (n). Both of these axis exhibit chaos however we shall conduct our investigation into the X axis with initial parameters: $\alpha = 1.4$, $\beta = 0.3$ and $[x_0, y_0] = [0.2, 0.2]$. The sample size of our dataset is 4096 values. We can see an example of the Henon Map in Figure 9.1.

As per our sample experiments listed in our analysis toolkit, we will be comparing the algorithms performance against an automated ARIMA model.

#### Graphical Differences

As per our experiment, we will be looking at how the algorithm performs against our baseline ARIMA model. Looking at Figure 9.2, we can see how good Alpex really is at learning the henon map. We see that for the last 100 values the predictions perfectly predict the Henon Map. This is backed up by the residuals graph, see Figure 9.3, which shows that after a few 100 values the predictions are near perfect, and for the last 3000 values the difference is near to 0. This is in stark contrast to ARIMA which we can see from the Residuals really struggles to make good predictions for the Henon Map.

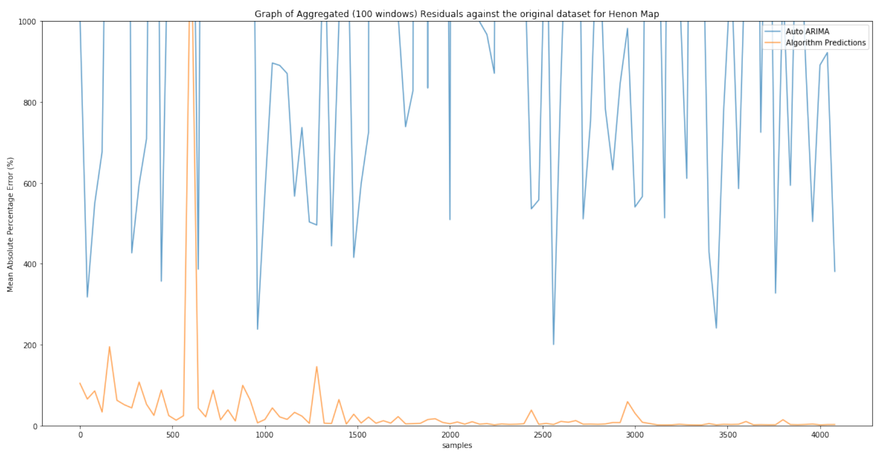


**Figure 9.3:** Graph showing the residuals as a percentage of the values, taken as an average with 100 windows for the Henon Map.

### Metrics

As well as being able to see on the graphs that Alpex is able to outperform our baseline metric, this is also reflected in our metrics. The following table shows how the metrics support our argument and we can see that Alpex can near perfectly learn the Series.

| Model | Mean APE | $R^2$ | Median APE | $R^2_{lag}$ |
|---|---|---|---|---|
| Alpex | 2.364417% | 0.999821 | 0.910843% | 0.999936 |
| Auto ARIMA | 121.506048% | 0.094828 | 83.366962% | 0.677289 |

The $R^2_{lag}$ metric is a metric that was added in later. The metric is outlined in Section 10.1.2 and tells us if our predictions are being proactive. This is not really an issue in this section as for mathematical series we can judge that ourselves. Mean/Median APE stand for Mean/Median Absolute Percentage Error.

## 9.1.2 Mackey-Glass

Another chaotic time series that we shall be investigating is the Mackey-Glass system. This system involves a complex differential equation which is shown by:

$$\frac{dx}{dt} = \beta * \frac{x_\tau}{1 + x_\tau^n} - \gamma * x \tag{9.2}$$

We are able to estimate this with the following equation:

$$x_{i+1} = A * x_i + B * (\frac{x_{i-n}}{1 + x_{i-n}^\gamma} + \frac{x_{i-n+1}}{1 + x_{i-n+1}^\gamma})\text{where, } A = \frac{2 * n - \beta * \tau}{2 * n + \beta * \tau}, B = \frac{\alpha * \tau}{2 * n + \beta * \tau} \tag{9.3}$$

Using this equation we are able to generate the Mackey-Glass dataset seen in Figure 9.1.

#### Graphical Difference

As with the Henon Map, we are able to display our predictions and the baseline predictions for the Mackey-Glass system. This can be seen in Appendix B.1, which shows how both Alpex and ARIMA seem near perfect in predicting the Mackey-Glass series. Looking at the residuals it actually appears that ARIMA outperforms Alpex, although there is not much in it when we look at the scale on the y axis. I believe the reason that Alpex performs worse in this situation is to do with Integer rounding, if we normalise the dataset in the same way, we see that the Alpex forecasts are in deed pinpoint accurate.

#### Metrics

When using our metrics suite to plot a table of metrics, see the below table, we can see how our metrics support our conclusions.

| Model | Mean APE | $R^2$ | Median APE | $R^2_{lag}$ |
|---|---|---|---|---|
| Alpex | 0.465524 | 0.999562 | 0.353670 | 0.878088 |
| Auto ARIMA | 0.021018 | 0.999999 | 0.016771 | 0.999791 |

## 9.2 Spikes

Whilst conducting experiments into the Lorenz System we noticed that we were getting spikes in our results, see Figure 9.4. Consulting with the author of the algorithm on this, he explained that he had seen a similar problem in the past and that this was due to the algorithm not being initialised correctly. However after ensuring that the history was correct and that his implementation was getting the same spikes, we decided to conduct an investigation into the cause of the spikes.

### 9.2.1 Lorenz System

As a cross between both the Mackey-Glass and the Henon Map, the Lorenz System is both a multi-dimensional and differential chaotic time series. Our initial investigation into the Lorenz System used the parameters, $\rho = 28, \sigma = 10, \beta = \frac{8}{3}$ and an initial state of $[1, 1, 1]$. Whilst looking at the Z axis of this series we saw the afformentioned spikes and hypothesised that this was due to a property of the time series.

In our representation of the Lorenz System, Figure 9.4, we see that the dataset starts from a small value, it then rises to a peak and then has a shallow fall before starting the initial sequence.

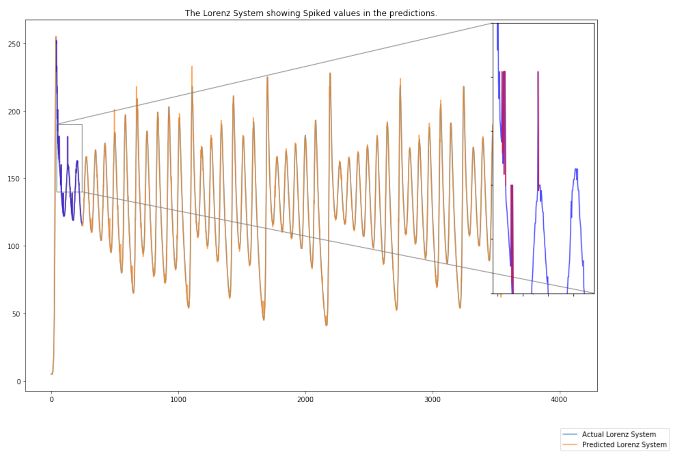


**Figure 9.4:** Graph showing spikes in the prediction of the Lorenz System.

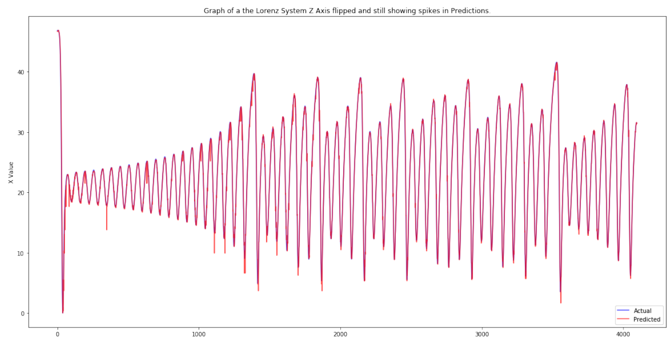


**Figure 9.5:** Graph showing spikes in the prediction of a flipped Lorenz System Z Axis series.

### History Initialisation

One hypothesis into this behaviour was related to the issue of the history not being initiated correctly. If initialising the history with a value of 0, as opposed to the correct value, had caused these spikes previously then the fact we are starting off with values close to 0 could be causing these spikes.

To test this hypothesis we tried flipping the graph in the y axis such that instead of starting with a low value we instead started with a high value. Figure 9.5 shows the results of this where we can clearly see that not only do we still see the spikes but also that the spikes themselves are flipped. This is also confirmed by the Sawtooth example where, even though we start off with a value of 0, we do not see spikes.

### Time Series Property

Another explanation for the spikes was to do with the initial rise and then the shallow dip. What we are seeing here is an example of Heteroscedasticity, which we explore in section

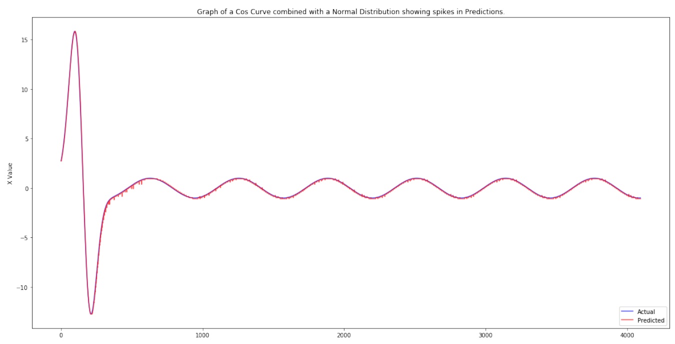


**Figure 9.6:** Graph showing spikes in the prediction of a Cos series combined with a normal distribution.

10.2.2. At the start we go to the extremes of the dataset, by starting off at a minimum peak rising to a maximum peak then having a shallow fall before starting the pattern. From the algorithms point of view, once it has gone through these initial peaks it will expect to see those same peaks again. This means that when we are getting a spike it is because the algorithm thinks that it is going back up to the extreme when in fact it does not. If that were the case then we would be able to implement a **Basic Representation** of a series that experiences spikes.

### 9.2.2 Basic Representation

As hypothesised, if the cause of the spikes was due to the peaks in the dataset and the algorithm thinking it was going back to a peak then we could replicate this by adding in peaks to a simple sequence. By combining a **Normal Distribution** with a Cos curve we were able to confirm this. Figure 9.6 shows how combining a normal distribution with a Cos curve causes spikes to be observed after the initial peak. To combine a normal distribution to our Cos curve we modify our function as:

$$y = \cos(x) * (1 + \frac{a}{\sigma * \sqrt{2 * \pi}} * e^{-\frac{1}{2} * (\frac{x-\mu}{\sigma})^2})$$

Our example uses a mean $\mu = 0$, a scaler $a = 60$ and a $\sigma = 0.6$. We know that as $x$ increases our normal distribution will tend to 0, therefore by incrementing the normal distribution by 1 our function will tend towards the cos curve as $x$ increases.

With both our cos example and the Lorenz System example, we can see that removing the initial peak and trough cause the spikes to disappear. Interestingly, both the peak and trough need to be removed for the spikes to completely disappear. This can be seen in Figure 9.7 where we have systematically removed the trough and then the peak for the Lorenz System dataset and seen that the spikes disappear.

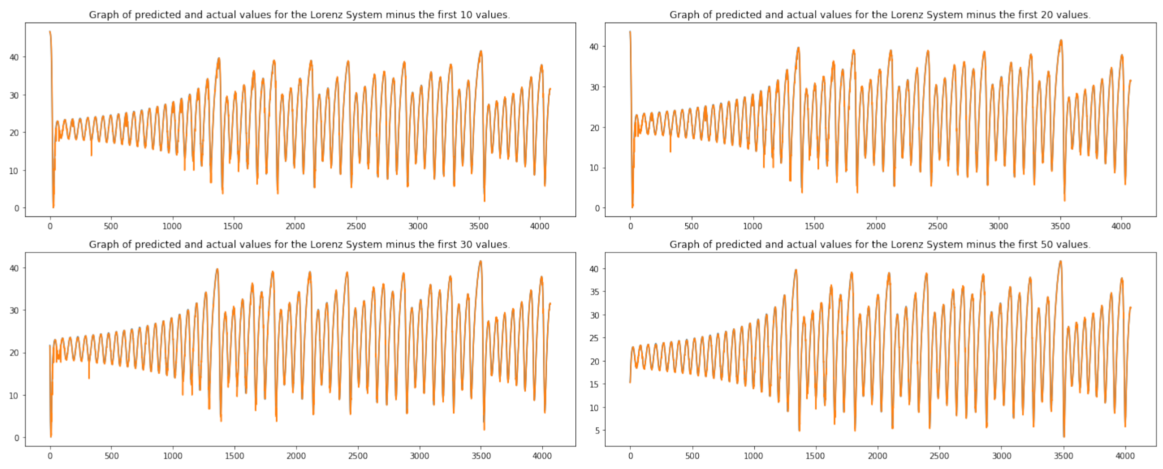


**Figure 9.7:** Graph showing how removing the initial peak and trough causes the spikes to disappear from the predictions.

The fact we are still able to see spikes in the most basic example, and that removing these extreme values removes the spikes, means that our hypothesis that a dataset starting with initial peaks and troughs is what causes the spikes to be observed.

### Location of Peaks

Although we have shown that starting with these extreme values causes spikes, we need to investigate whether or not the spikes only appear with initial extreme values. Luckily our new implementation of the cos curve allows us to change the mean of our distribution such that we can move the extreme values. We can hypothesise that if the peaks are later in the dataset then the algorithm will have already had a chance to learn the pattern such that it should be less effected by the peaks however I believe we are still likely to see spikes.

Figure 9.8 shows a variety of different configurations of our normal distribution patterns and the predictions the algorithm is able to make for each. We can clearly see that when the peaks and troughs happen later on the algorithm is much less affected than if they happened at the start. For example, in the last example, we see that after the peaks the predicted values fluctuate between a small range causing smaller, mini spikes.

Our investigation into how the location of the peaks and troughs effect the spikes conclude that any extreme change in variance can cause spikes to be observed however if the algorithm has already learnt the pattern then these spikes are less pronounced and the algorithm can quickly switch back to the original pattern.

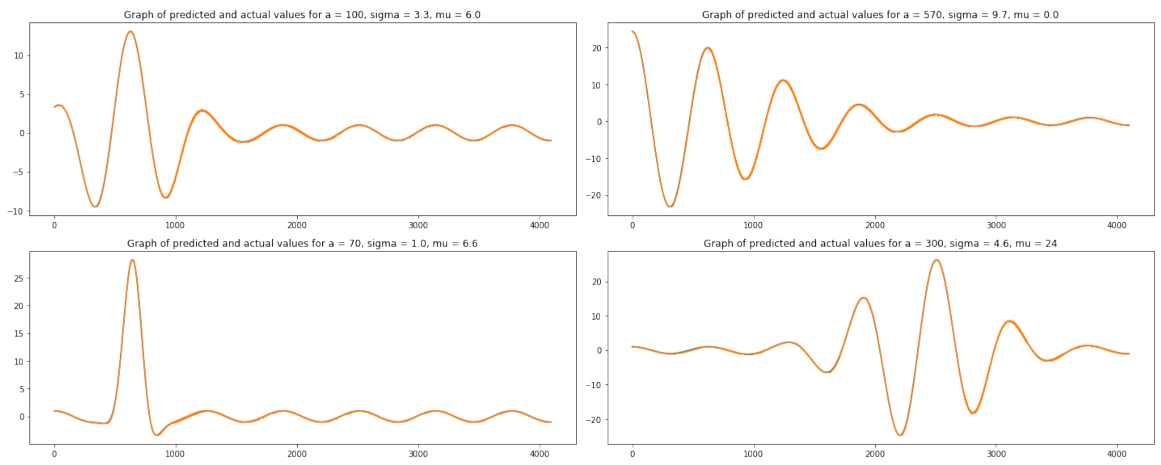


**Figure 9.8:** Graphs showing how the location of peaks effects spikes in the predictions.

### 9.2.3 Mitigating Spikes

We have seen through this investigation why and when spikes can occur in our predictions, however we have not discussed methods for mitigating these.

One method that can be used is seen in, Figure 9.7, where we saw that removing the peaks and troughs from the dataset removes the spikes. This is a solution however we may want to be able to predict future peaks and troughs and this removes the algorithms ability to do that.

Before we are able to predict values, due to the nature of the algorithm, we first need to normalise our data. This is usually done using a basic normalisation technique defined by:

$$x = \frac{x - min(X)}{max(X) - min(X)} * predictable\_range$$

Where $min(X)$ and $max(X)$ represent the range of values in the dataset and $predictable\_range$ is equivalent to the $max\_value$ variable used to initialise the algorithm, this is defaulted to 254.

One proposed solution is to normalise our dataset using an activation function, such as sigmoid or tanh. These methods cause outliers to be less pronounced, therefore we can still keep the peaks and troughs inside the dataset without spikes being present.

### 9.2.4 Ignoring Spikes

Sometimes we may not actually care if we have spikes in our dataset. If we do have spikes in the data it can throw the Mean Absolute Percentage Error completely off, therefore if we want to ignore the spikes we should calculate the Median Absolute Percentage Error.

## 9.3 Noise

In the real world, time series are not going to follow exact mathematical formulas. We are likely to get anomalies in the data due to: observation errors, physical anomalies, or the nature of the series itself. Therefore, a time series prediction algorithm needs to be tolerant to non-exact, or noisy datasets.

### Types of Noise

Most signals measured by a scientist or an engineer are polluted by noise such that,

$$Observation = Signal + Noise$$

Noise is naturally occurring in real world datasets, however when modelling noise we need a way to mathematically generate this. This is often done by applying filters to a 'clean' dataset, using random numbers.

**Gaussian Noise** A Gaussian Filter uses a normal distribution to apply noise to a clean dataset. This is done by creating a random vector sampled from a normal distribution with a given, variance and mean.

$$g \sim \{\mathcal{N}(\mu,\, \sigma^2)\,\}.$$

This is what creates the noise, which we can sum with the signal to simulate an observation.

We know from a normal distribution, that the higher the variance, the flatter the curve. This translates to values in our random vector being further away from zero, and therefore increases the amount of noise we are simulating.

An example of this can be seen in Figure 9.9 where we have a straight line at $y = 1$ and a random vector sampled from a normal distribution, these are then summed together to obtain a noisy dataset. As we can see, by increasing the value of sigma (the variance) we get a flatter curve that the random vector is sampled from, and therefore more noise in the 'observation' dataset.

We can see how this translates to a more complex equation by applying noise to a cos curve, this is seen in Figure 9.10.

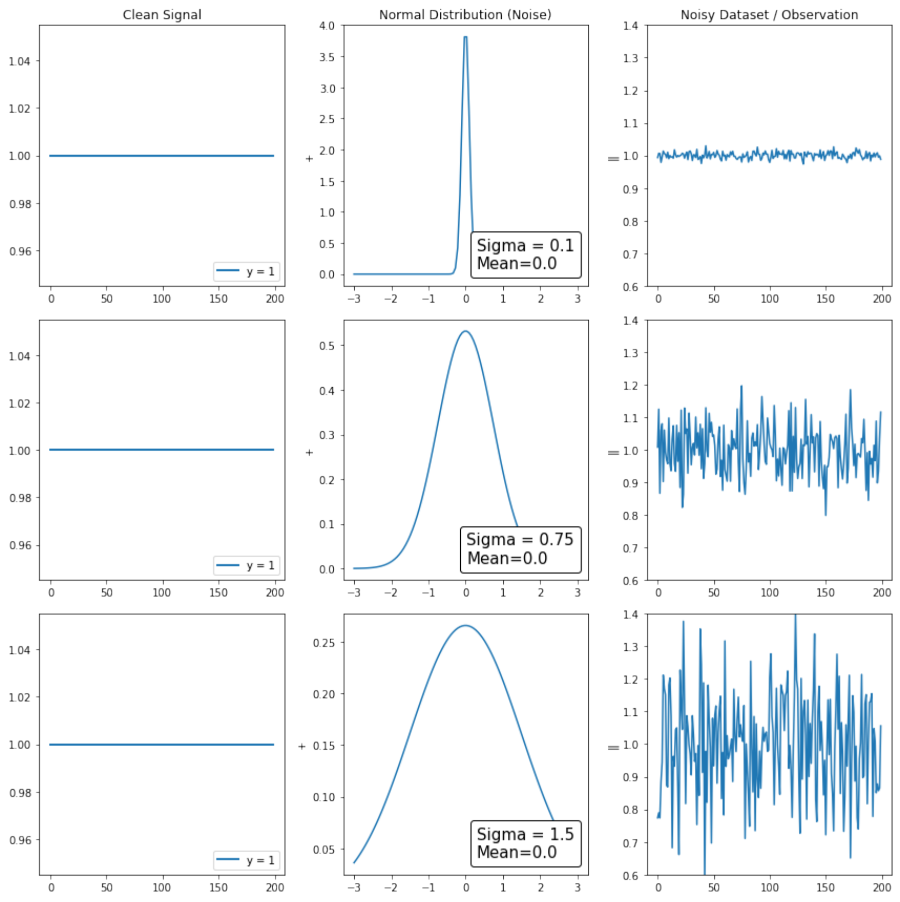


**Figure 9.9:** An example of how we can use Gaussian Distribution to apply noise to a straight line.

### 9.3.1 Problems with Noise

For any forecasting algorithm, we must look at the past to predict the future. We can reason that if we know exactly what the past is then we can accurately predict the future, this is the property we have been able to show the algorithm is extremely effective at. However, when we throw noise into the data, the past we are looking at and learning from is itself not correct, therefore we are not just learning on incorrect data, but we are trying to predict incorrect data.

We can reason from this that the problem of a perfect forecasting algorithm is itself intractable, therefore the best fit algorithm needs to strike a balance between over-fitting and under-fitting. We can use artificial noise to asses how the algorithm performs with this.

### 9.3.2 Baseline Metrics

Before we start any experiment, we need to set a baseline for the metrics, this can be done by running the algorithm with a sigma of 0, which is the same as applying no noise. This will act

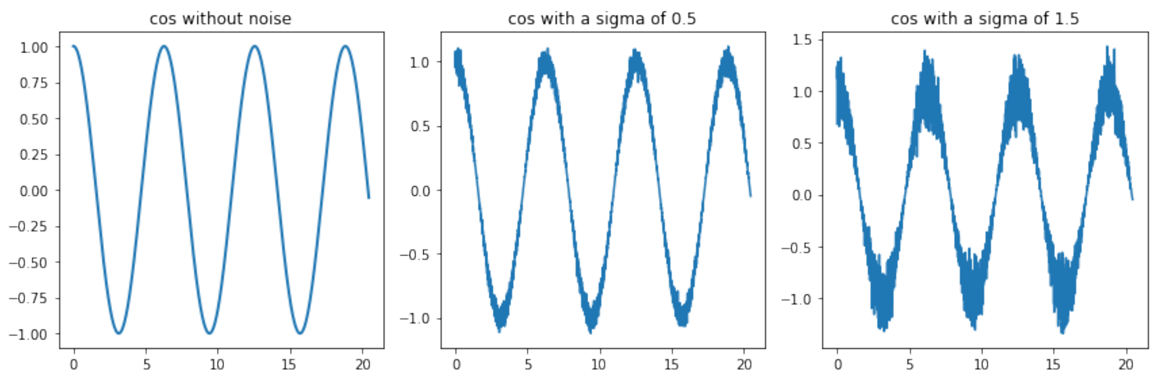


**Figure 9.10:** An example of how increasing the variance effects the amount of noise on a Cos curve.

| Dataset | MAE | MAPE | Change of Direction Error (CODE) | $r^2$ |
|---|---|---|---|---|
| Random | 85.6704 | 262.9275% | 49.9389 | -1.0 |

**Table 9.1:** Metric scores for predicting a random dataset.

as an upper bound to the performance of the algorithm, as we do not expect the algorithm to perform better on noisy data, in comparison to clean data.

As well as setting an upper bound, we must also set a lower bound for the performance, this can be achieved by predicting values for a random dataset and generating metrics for this. Random number generators in computers, are only ever psuedo-random [85] and we know that the generator used for python [37] is too, it is even advised that this generator is not secure enough for cryptography. As of this nature, there is a possibility that the algorithm could learn the generator itself, however this is unlikely. Furthermore as the same generator is used for applying the noise, this will serve as a sufficient lower bound. We can see the resulting metric values in Table 9.1.

With this in mind, we will be able to measure the success of the algorithm in a number of ways. We will be able to see the point at which data becomes too noisy to predict, and then compare this against the original dataset at different scales. We can also use this to find the point where the algorithm starts to lose accuracy. These can be measured by looking at the metrics in relation to the bounds.

Our investigation into noise is about finding out how well the algorithm is able to deal with noise and not how this compares to other algorithms, therefore for this section we will not be comparing the algorithm against other forecasting methods such as ARIMA.

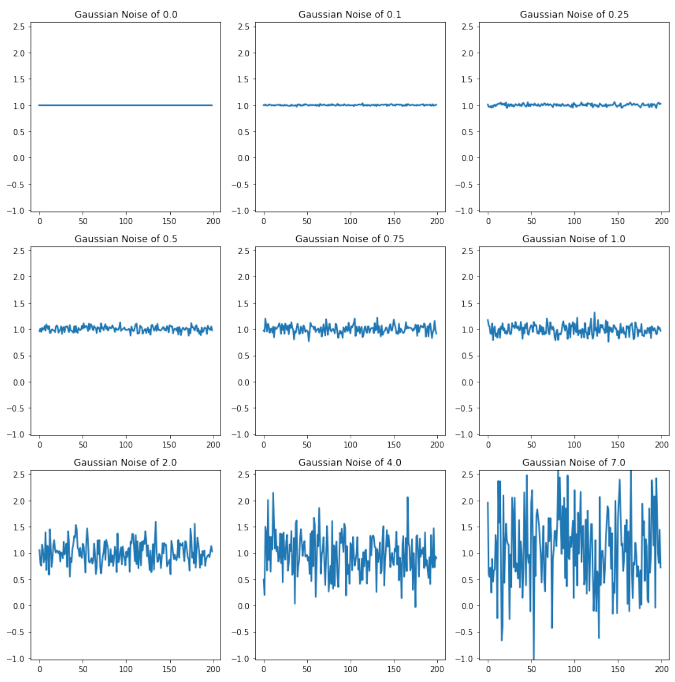


**Figure 9.11:** The effect of the testing sigma values on a straight line.

### 9.3.3 Experimental Datasets and Parameters

We will use the same scale of sigma values (variance) for each experiment, these are:

$$sigmas = [0.0, 0.1, 0.25, 0.5, 0.75, 1.0, 2.0, 4.0, 7.0]$$

We can see the effect these sigma values have on a straight line in Figure 9.11. As we can see from this, 0.75 is about where the line starts to break down, with anything above 1.0 being where a straight line is indistinguishable.

### 9.3.4 Noise Experiments

#### Sawtooth

The Sawtooth series is a very simplistic time series, which makes it a perfect starting point. Figure B.6 shows a microscopic view of the last 30 values of the sawtooth time-series.

We can see how the predicted values compare with the actual values in Figure 9.12. From a visual inspection of these graphs we can see that anything after a sigma of 1.0 the algorithm begins to struggle, however even at 0.75 the algorithm is good at making predictions.

This visual inspection is backed up by the metrics, in Table 9.2 and in Figure 9.12(right), where it is able to accurately predict with some noise up until 0.75, when the Mean Absolute

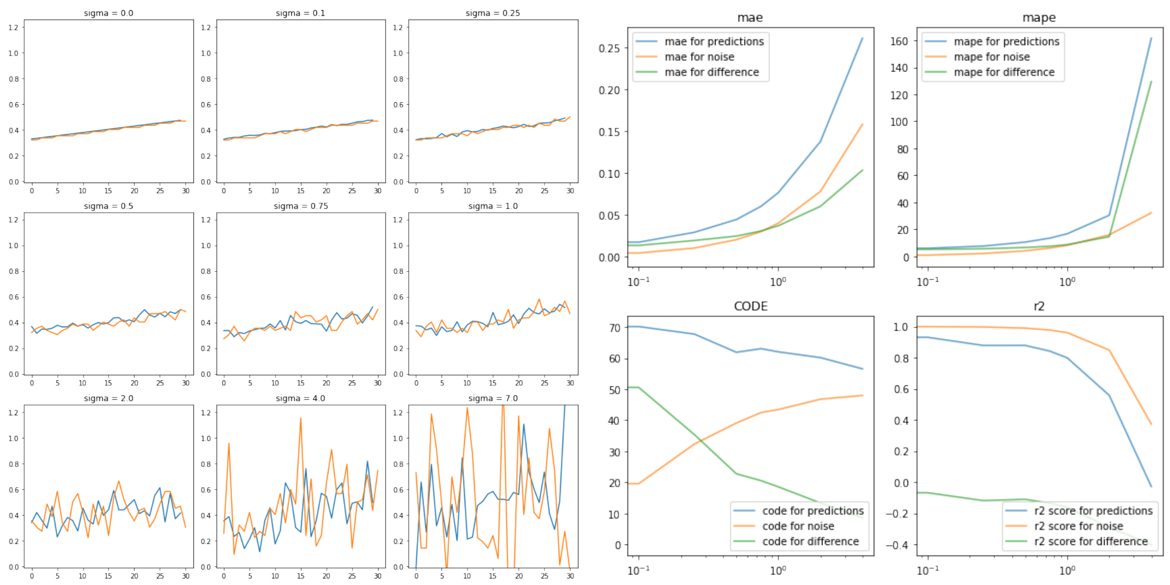


**Figure 9.12:** A zoomed in view of the last 30 values of the predicted (orange) vs actual (blue) sawtooth dataset at different sigma values(left). A plot of metric scores against Sigma for the Sawtooth dataset(right).

Percentage Error goes above 15%. We also take a look at how the metrics compare with the amount of noise, the orange line in Figure 9.12(right) shows the difference between the noisy set and the clean set. We can then look at the green line, which is the difference between the prediction scores and the differences scores, this can give us a scaled idea of the performance. From this we can see that when scaled, we get very good accuracy up until a sigma value of 1.0, and relatively good predictions up until 2.0.

| Sigmas | MAE | MAPE (%) | Change of Direction Error (CODE) | $r^2$ |
|---|---|---|---|---|
| 0.00 | 0.008573 | 5.212907 | 69.084249 | 0.996014 |
| 0.10 | 0.016788 | 5.898174 | 70.134310 | 0.931077 |
| 0.25 | 0.028553 | 7.515865 | 67.716728 | 0.878641 |
| 0.50 | 0.043855 | 10.458887 | 61.855922 | 0.878902 |
| 0.75 | 0.059587 | 13.393406 | 63.028083 | 0.842262 |
| 1.00 | 0.076076 | 16.640696 | 62.002442 | 0.798805 |
| 2.00 | 0.137081 | 30.348109 | 60.146520 | 0.557737 |
| 4.00 | 0.260531 | 161.506849 | 56.532357 | -0.028621 |
| 7.00 | 0.425786 | 673.442345 | 54.139194 | -0.450938 |

**Table 9.2:** Metric scores for predicting the Sawtooth Dataset.

#### Logistic Map

Increasing the complexity of the dataset, we can look at how the algorithm performs on a Logistic Map, seen in Figure B.8. The results for this experiment are found in Appendix B.2

From the visual inspection of the differences, we can see very similar results to both the sawtooth and the cos curve, where up until a sigma value of 2.0 we are able to make good predictions.

The metrics somewhat support this. The interesting insight from this dataset compared to the previous, is that up until a sigma value of 0.75, the algorithm has a low Mean Absolute Percentage Error, however above 0.75 that begins to rapidly increase, faster than the other two. Another interesting observation is that the $r^2$ falls very quickly for the Logistic Map, with it having a value of 0.78 at a sigma value of 0.75 as opposed to the 0.84 and 0.98 seen by the sawtooth and cos curves respectively. Finally, it is interesting that the Change of Direction Error for this dataset, is fairly low, only going above 10% after a sigma value of 1.0.

### 9.3.5 Predicting with Noise

Noisy dataset's are difficult for this new algorithm to predict due to the deterministic nature of the algortihm. Noise in the training set will propagate through to the predictions, and as the noise is random and different every time, this makes the predictions less accurate. The aim of this experiment is to look at different prediction methods that could be used to mitigate this.

#### Different Ways to Predict with Noise

Typically we would use this algorithm to train and predict at the same time, such that when we receive a value, we train on that and then predict another. Figure 9.13 (left) shows how this is normally done.

In some real world situations we are able to say that a time series is similar to a mathematical time series that we know. If we know what the signal should be, and the observation, then we can try training the algorithm on the signal and then carry out the predictions on the observation like normal. Figure 9.13 (right) shows this.

As we are using the noisy dataset to predict in the normal way, in the last method, then we are still training on noisy data, therefore another method would be to train on the clean data, then **just** predict (without training) on the noisy set. Figure 9.14(left) shows this.

The last two methods focused on mitigating the problem of noise in the training set, however another method we could consider would be to gradually build noise into the training set. In

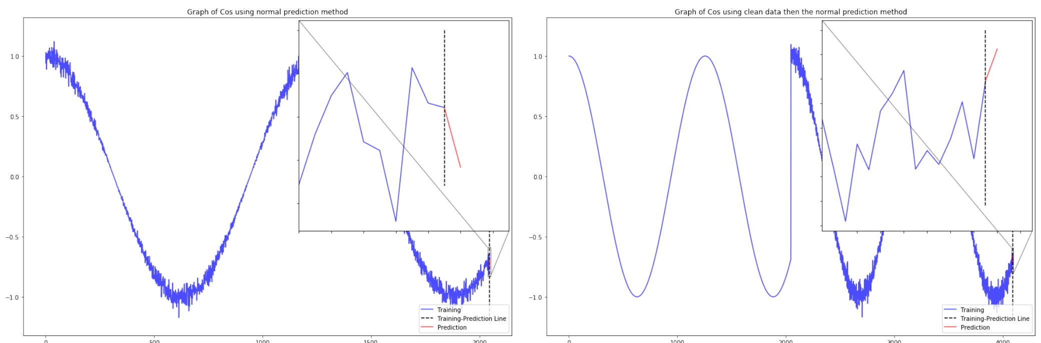


**Figure 9.13:** A graph showing how normal prediction works(left) and training on a clean dataset, then predicting in the normal way(right) with the algorithm on a noisy Cos curve with a sigma value of 0.5.

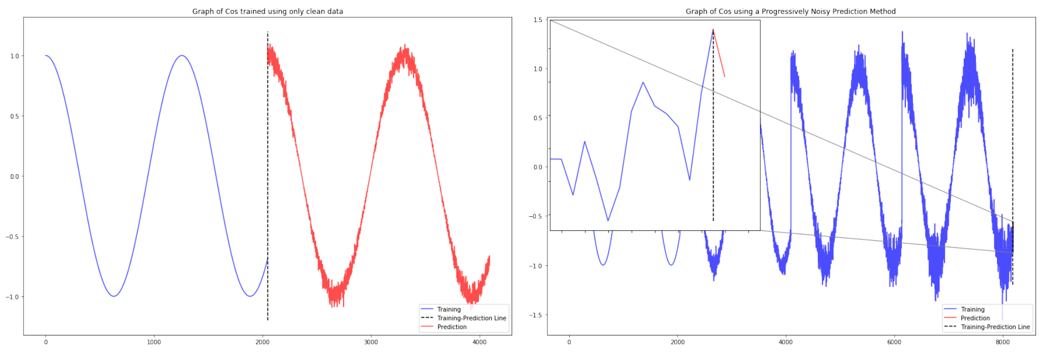


**Figure 9.14:** A graph showing how we can train on a clean dataset, then predict without training(left) and how we can predict on gradually noisier datasets(right), then predict on a noisy Cos curve with a sigma value of 0.5.

practice this could reduce over fitting as the algorithm will have been made gradually more tolerant to noise. Figure 9.14(right) shows this.

### Experiments to Determine the Feasibility of Different Prediction Methods with Noise

We are able to replicate the tests setup in the previous section, see 9.3.2, and the some of the datasets that were used, see 9.3.3, to evaluate the success of this method.

We have produced metrics for the Sawtooth, Cos (Appendix B.3) and the Logisitic Map (Figure 9.15) to determine if any of these methods produce better predictions on noisy data.

### Evaluating the Feasibility of Different Prediction Methods with Noise

From the metrics we produced of the Sawtooth, Cos and Logisitc Map we are able to see very similar results.

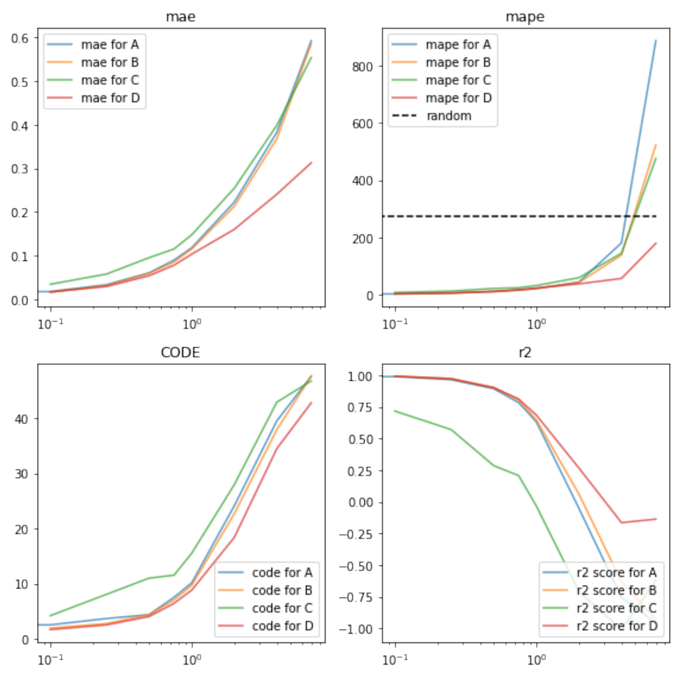


**Figure 9.15:** Metrics for the Logistic Map using different prediction methods, where:
A = Normally trained predictions B = Trained on clean data first
C = Trained on progressively noisy data D = Trained only on clean data

We can see that using the method of training on only clean data (the red lines) consistently outperformed the other methods on extremely noisy time series. Previously, after a sigma of 2.0 the algorithm would always break down on noisy datasets, with whatever metric we are looking at getting exponentially worse. Due to the fact that the algorithm is not being trained on the noisy set at all, we do not see this exponential explosion.

Between the other methods, there is no significant increase of accuracy over the initial method of training and predicting. It seems as though training the algorithm on progressively noisy datasets will increase the accuracy, however this is not conclusive as in some experiments and metrics this is observed but in others it is not.

---

**The significant increase in accuracy brought by training on only clean data has an implication for real world time series, where techniques to smooth or remove outliers during training may provide compelling improvements.**

---

# Chapter 10

# Real World Analysis

We have conducted experiments to assess the capabilities of Alpex on Synthetic datasets with a known ground truth. This section focuses on exploring real world datasets with an unknown, underlying, structure to determine if Alpex is effective in this context. This investigation will involve experiments into real-world time series properties to determine which classes of real-world datasets Alpex can proactively forecast for.

## 10.1 Lagged Predictions

One of the emergent properties of the algorithm is that when it has not seen a value before it will predict the previous value it has seen. This can be seen in the simple example of the algorithm learning the sequence [1, 2, 3]:

| | | | | | |
|---|---|---|---|---|---|
| Input | 1 | 2 | 3 | 1 | 2 |
| Prediction | 1 | 2 | 3 | 2 | 3 |

This can also be seen for the Sawtooth series, where on the first upwards slope the predictions lag behind the dataset. Figure 10.1 clearly shows how on the first slope the algorithm predicts the previous value, we call this a **Reactive** prediction. The graph then shows how once the algorithm has seen all of the values in the sequence it is able to accurately predict the next value rather than predicting the past value, we call this a **Proactive** prediction.

### 10.1.1 Problems with Lag

One of the difficulties with lag is how hard it can be to detect and measure. With the Sawtooth example we were able to obviously see the predictions lagging behind the series, however for very complex series this is not always possible. Aside from a visual inspection, the metrics we have already defined do not help us:

| Series | MAPE | $R^2$ |
|---|---|---|
| Reactive Sawtooth | 3.89% | 0.9409 |
| Proactive Sawtooth | 2.497% | 0.9704 |

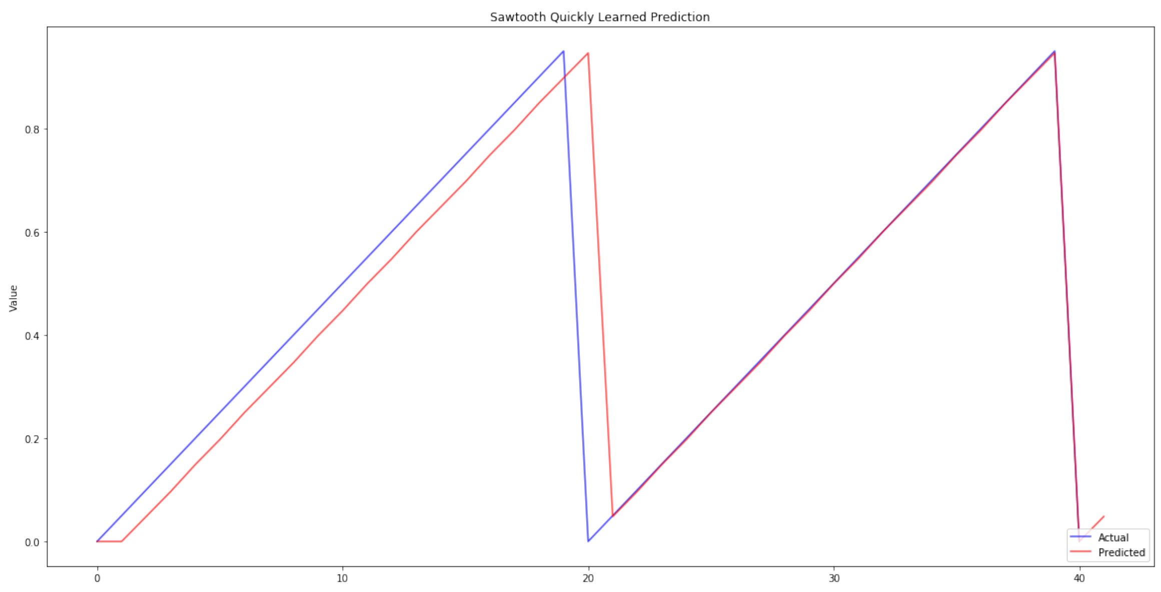


**Figure 10.1:** A Graph showing the reactive prediction of the Sawtooth series and then the proactive prediction.

If we were to just look at the metrics, we see that the Reactive Sawtooth actually seems extremely good, however we know from a visual inspection that all it is doing is predicting the last value.

### 10.1.2 Detecting Lag

#### Visual Inspection

As seen with the Sawtooth example, in some cases a visual inspection of the predictions against the actual values is enough to detect lag in the predictions. For more complex cases this isn't always so clear and it can be clearer to have the predictions leading the data. In this way if the predictions and the series overlap then we know that we are making reactive predictions.

#### Metric Inspection

The Sawtooth example showed us that a reactive predictor was still able to get very good $R^2$ and MAPE scores, that means that these metrics alone would not be enough to determine if a forecast is Proactive or Reactive.

If we revist our definition of the coefficient of determination, $R^2$, as:

$$R^2 = 1 - \frac{MSE(Model)}{MSE(Base)} = 1 - \frac{\sum(\hat{y} - y)^2}{\sum(y - \bar{y})^2}$$

The model we have been using for $MSE(Base)$ is the mean. If we were to replace the base model to be the previous value (a naive forecast), then we would be comparing whether the algorithm is more proactive than reactive. A positive value, close to 1, means the algorithm is proactive. We can define this metric as:

$$R^2_{lag} = 1 - \frac{\sum_t(\hat{y}_t - y_t)^2}{\sum_t(y_t - y_{t-1})^2}$$

If we revisit our Sawtooth example with this metric, we get:

| Series | MAPE | $R^2$ | $R^2_{lag}$ |
|---|---|---|---|
| Reactive Sawtooth | 3.89% | 0.9409 | -0.00489 |
| Proactive Sawtooth | 2.497% | 0.9704 | 0.9990 |

Another method we can use for detecting lag is to plot how the MAPE and $R^2$ change over time, if we then calculate these metrics, assuming we are lagging behind such that $R^2$ would be defined as:

$$R^2_{prev} = 1 - \frac{\sum_t(\hat{y}_t - y_{t-1})^2}{\sum_t(y_t - \bar{y})^2}$$

if we are being proactive, there will be a point where the initially calculated metrics outperform the lagged metrics.

## 10.1.3 Examples of Lag

In the Sawtooth example it is very obvious to see that the algorithm is reactive for a stage, and then proactive, however this is not always so clear. In this section we are going to look at a couple of examples where it is not obvious and we are going to apply the tests that we previously laid out to determine if our predictions are proactive or reactive.

#### Venice Hourly Water Level

The Venice Hourly Water Level [45] dataset is a large stationary real world dataset making it perfect for analysing the Algorithm. We can see the Venice water supply dataset in Appendix C.1.1.

We have calculated the MAPE and $R^2$ for the last 1000 predictions as:

| Series | MAPE | $R^2$ |
|---|---|---|
| Venice | 26.168% | 0.981 |

On the surface this looks very good, however we do not know if the algorithm is being proactive or reactive, therefore we need to conduct our lag tests.

We can first perform a visual test on the predictions, this can be seen in Figure 10.2(left). From these graphs we can see that the predictions lead the actual data, therefore we can hypothesise that the algorithm is proactive. However this is only a snapshot of the data therefore a deeper analysis into the whole dataset is needed.

We can then fall back to our metrics, the first thing we shall look at is our $R^2_{lag}$ metric:

| Series | MAPE | $R^2$ | $R^2_{lag}$ |
|---|---|---|---|
| Venice | 26.168% | 0.981 | 0.889 |

The fact the $R^2_{lag}$ is close to 1 tells us that our predictions are better than just predicting the last value, this is a good indicator that our algorithm is being proactive.

To be 100% sure, we shall look at how the metrics change over time. Figure 10.2(right) shows that after just a few values, the normal prediction metrics become better than the ones for the lagged prediction. The fact the normal predictions are always better than the lagged predictions after this point tells us that the algorithm is now proactive.

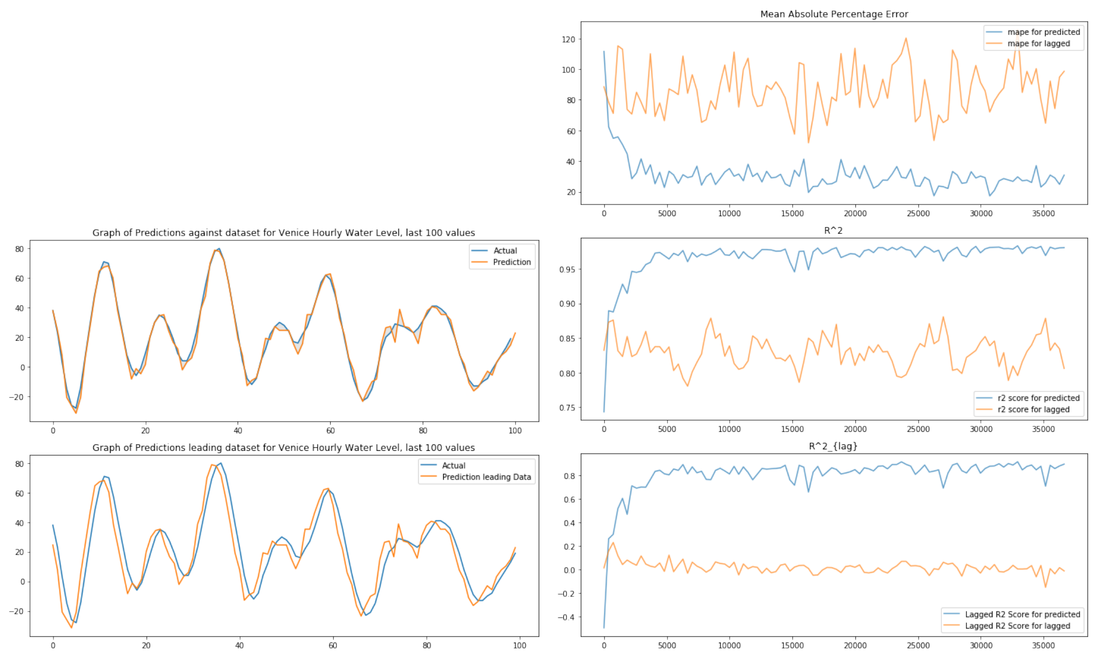


**Figure 10.2:** Visual inspection of lag(left) and Metrics plotted against time(right) for the Venice Hourly Water Level Dataset.

## Belgium Offshore Power Generation

Similar to the Venice dataset, the Belgium Offshore Power Generation Dataset [15], seen in Appendix C.1.2, is large and stationary however it is less clear if our predictions for this dataset are lagged therefore we must conduct our experiments.

Performing a visual analysis, Figure 10.3(left) appears to show that the algorithm is just making reactive predictions and not actually able to make accurate forecasts. This is seen by the graph with predictions leading the dataset where the plots closely overlap. The metrics also support this:

| Series | MAPE | $R^2$ | $R^2_{lag}$ |
|---|---|---|---|
| Belgium | 18.604% | 0.864 | -0.408 |

We see that the $R^2_{lag}$ metric is negative meaning that our forecasts are worse than if we just predicted the last value.

Our final check is to plot the metrics against time. Figure 10.3(right) shows that the lagged predictions perform consistently better than the actual predictions and that this doesn't change over time. Our predictions therefore consistently fail all of our checks therefore we can conclude that the algorithm is being reactive and not proactive on this dataset.

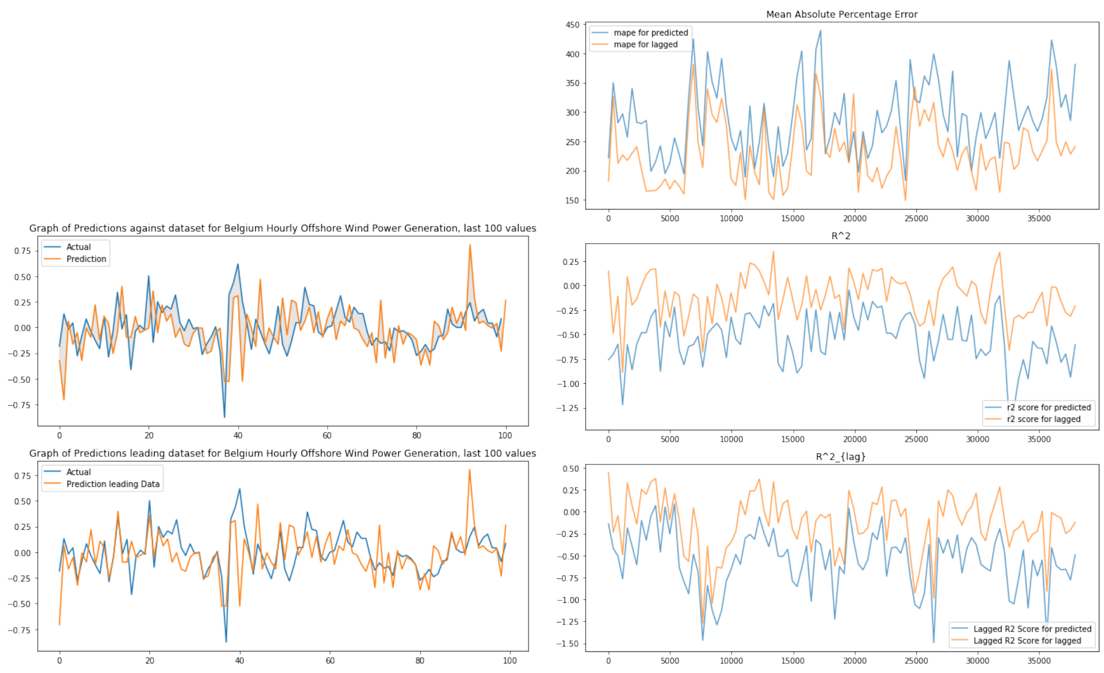


**Figure 10.3:** Visual inspection of lag(left) and Metrics plotted against time(right) for the Hourly Belgium Offshore Power Generation Dataset.

### 10.1.4 Conclusions

Throughout this section we have set out the tests that we can perform on a prediction to ensure that it is proactive and not reactive. The tests that need to be passed are:

1. The prediction must pass a visual inspection by plotting the prediction as leading the dataset and ensuring the prediction and actual data do not overlap.
2. The prediction must get a positive number near to one for the $R^2_{lag}$ metric that we set out.
3. When metrics are plotted against time for the lagged prediction method and the regular prediction, we must see a point where the lagged predictions become worse and stay worse.

If a prediction is able to satisfy these tests then we can be confident that it is forecasting into the future and not repeating the last value it has seen.

The $R^2_{lag}$ metric has also been added to the Metrics object such that this metric is automatically calculated and our sample experiments include the graphical tests for lag.

## 10.2 Exploring Time Series

In this section we are going to be conducting a systematic investigation into time series properties exhibited by real world datasets. In analysing these properties we have conducted many different experiments however we are going to focus on only a few examples of the results of these experiments rather than showing them all.

During our analysis of mathematical series we actually encountered a lot of these properties without explicitly stating them. For example the Spikes section, see section 9.2, looks at Heteroskedasticity in datasets. Our Sawtooth analysis shows trend on the first upwards slope, and cos curves experience seasonality. We will therefore be able to hypothesise about the different time series properties and then perform investigations to prove or disprove these.

### 10.2.1 Trend

Trend is generally thought of as being a change in mean over time, for example if we are looking at stock prices a generally increasing price means we have a positive trend. Trend can be: linear, exponential or logarithmic(damped). Trend generally comes in two forms, an additive model for linear trends

$$observation = signal + trend + noise$$

and a multiplicative model for non linear trends

$$observation = signal * trend * noise$$

Most forecasting algorithms are not able to deal with trend in datasets and therefore trend is often removed. There are two main ways to remove trend, these are differencing and plotting a function to estimate the trend.

#### Alpex

Before we test how the algorithm performs when trend is present, we can hypothesise that the algorithm will struggle. The reason for this refers to our Lagged Predictions section where we saw that for new values the algorithm will just predict the last value it has seen. If there is a trend in the dataset then we are constantly going to see new values and therefore the algorithm will resort to performing a lagged prediction.

Figure 10.4 shows how the NASDAQ highest daily price over time [25], we can see that there is a clear trend in the dataset and we can see that the algorithm is not able to make predictions for that. This is backed up by other experiments and we can therefore conclude that trend must be removed from a dataset before predictions are made.

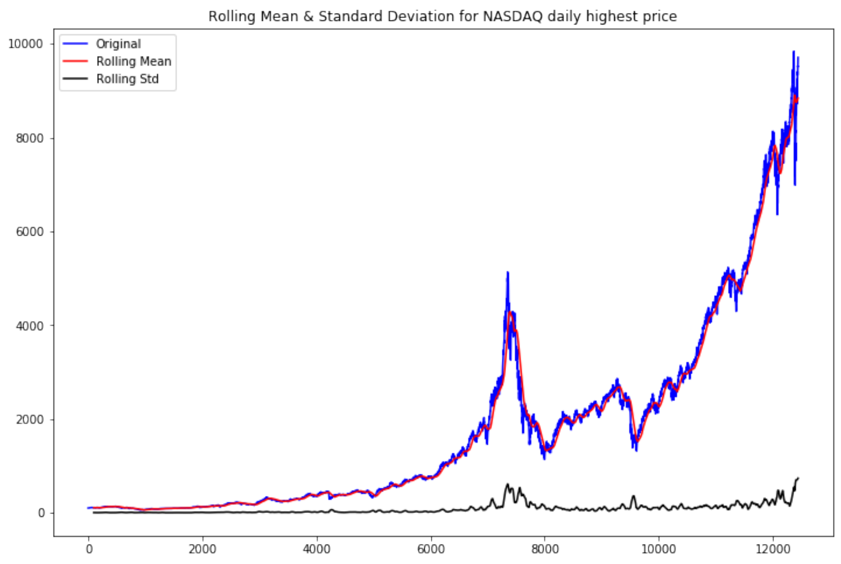


**Figure 10.4:** The daily highest price of NASDAQ showing trend.

### Differencing

Differencing with the Alpex seems to not improve the accuracy, unlike ARIMA where we want to difference the algorithm to remove trend and seasonality, often differencing will also remove periodic trends which the algorithm needs to be able to predict for.

### Trend Estimation

Trend estimation is the preferred way to remove trend for Alpex, there are two ways to do this. We can either take a rolling moving average of the series and subtract that, or we can plot a function to estimate the trend. When making t+1 predictions, the rolling mean method is often preferred. however with anything further we would want to use a functional plot. When using the rolling mean methodology it is important we do not lose any of our seasonality, therefore the window size should be equal to the seasonal size.

## 10.2.2 Heteroskedasticity

Heteroskedasticity is observed when a series variance is a factor of time, this means that the variance changes throughout the series. There are two types of Heteroskedasticity that we are going to look at, these are pure Heteroskedasticity and a varied Heteroskedasticity, see Figure 10.5. In section 9.2 we explored spikes in our predictions of the Lorenz System, the reason for this was that we had a changing variance, Heteroskedasticity. We can therefore base our hypotheses on the results that we found on the mathematical series.

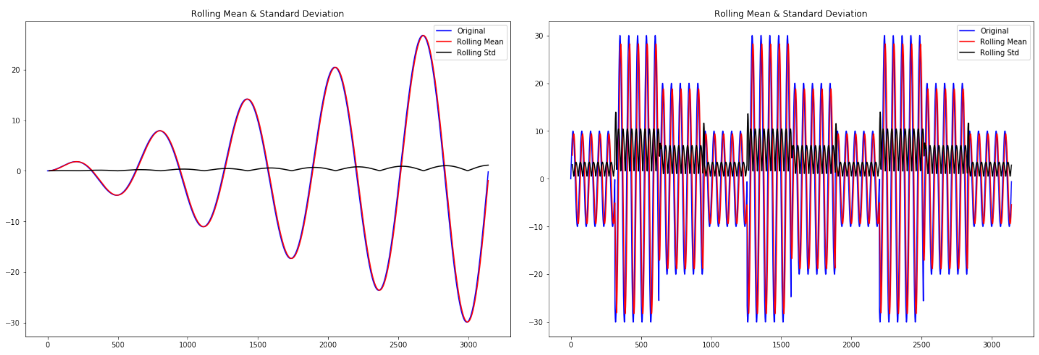


**Figure 10.5:** An example of Pure (left) and Varied (right) Heteroskedasticity.

### Detecting Heteroskedasticity

To detect Heteroskedasticity, as seen in Figure 10.7, we want to plot a rolling standard deviation such that we can see how the variation changes over time.

### Pure Heteroskedasticity

Pure Heteroskedasticity is very similar to trend as we are constantly seeing new values as the variance increases. We can therefore hypothesise that a constantly increasing variance means we are seeing new values, the algorithm will just make reactive predictions. We can also hypothesise that a constantly decreasing trend will cause spikes in the predictions.

If we look at the Actual wind generation in Denmark 1 (bidding zone) in MW from [15], seen in Figure 10.6, we can see how the variance constantly increases and the algorithm is not able to make proactive predictions and instead just lags behind the dataset.

We can conclude that pure Heteroskedasticity is something that the algorithm is not able to forecast for. One way that a user may want to mitigate this is by taking a log of the dataset.

Pure Heteroskedasticity is very similar to, and in many situations is the same as, a multiplicative trend.

### Varied Heteroskedasticity

Our varied Heteroskedasticity however fits the nature of the algorithm quite nicely. When we switch between different time series the algorithm is able to quickly pick up on the pattern after switching. This is replicated with the varied Heteroskedasticity as the algorithm is able to pick up on the different sections of variance.

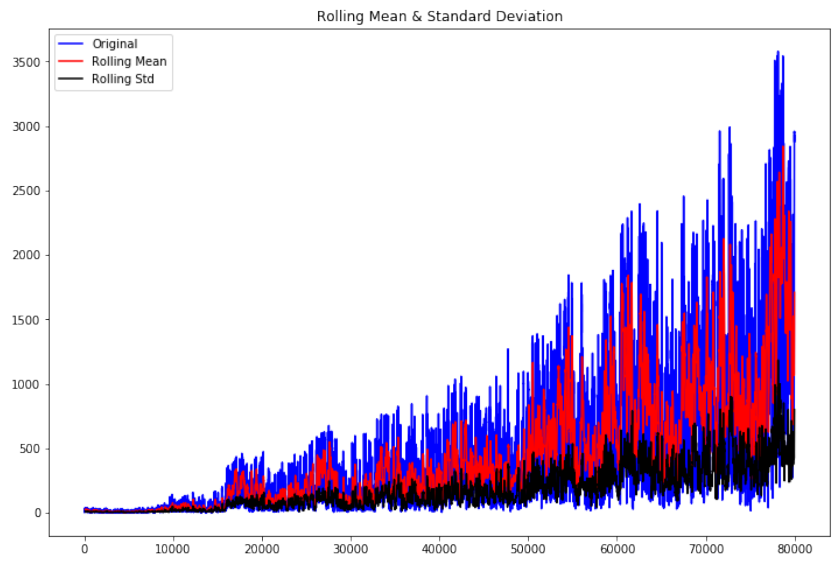


**Figure 10.6:** An example of a dataset we can not predict for as it exhibits pure Heteroskedasticity.

This is shown in Figure 10.7 which shows the real world Germany Hourly Solar Panel Generation Dataset [15]. We can see that the rolling standard deviation at the bottom of the graph changes over time, however what we see is that this change appears to follow a pattern. It is similar to switching series, with the different variance levels being different series. This is exciting as algorithms like ARIMA will use methods to try and remove this before predicting, Alpex however is able to make predictions straight away.

### 10.2.3 Seasonality

Seasonality is a rigid periodic trend which occurs in set time intervals, for example, if a hotel is looking at number of bookings for a year, they may find they have more bookings in summer months as opposed to winter, we can say the dataset has yearly seasonality. We can expand our defintion of observation, for an additive model, to include this such that:

$$observation = signal + trend + seasonality + noise$$

What we saw from the mathematical analysis is that Alpex is extremely good at picking up on patterns in a dataset. Moving this into the real world, we can hypothesise that a dataset will need to have some periodic pattern or periodic trend, one type of periodic trend is Seasonality.

One way to detect Seasonality is to plot the Auto-Correlation Function against lags for a dataset. What this tells us is how previous values in the dataset influence the current value. If seasonality is present then we can see distinct patterns in the ACF, however if there is no seasonality then we cannot. The correlation between predictability and seasonality is distinct

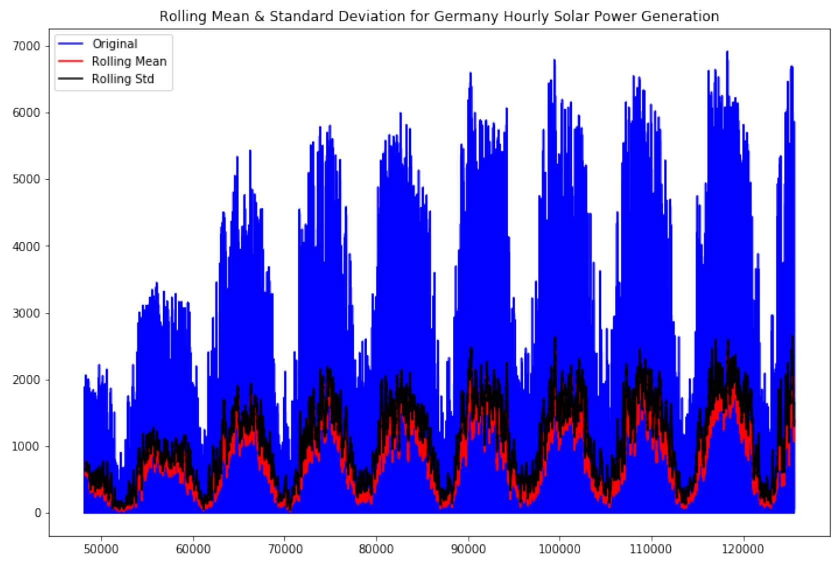


**Figure 10.7:** An example of a dataset we can predict for as it exhibits varied Heteroskedasticity.

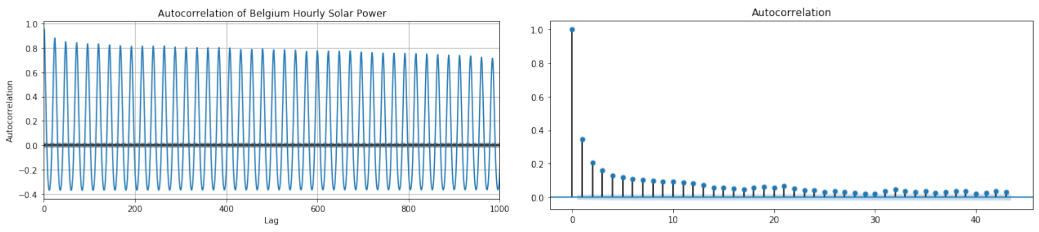


**Figure 10.8:** The autocorrelation function of: the Belgium hourly solar power generation with strong seasonality (left) and the Swiss hourly wind power generation that is stationary (right).

from our experiments. Every single dataset that we were able to predict experienced some sort of periodic trend.

Figure 10.8 shows the ACF for the Belgium Hourly Solar Power generation dataset, that we have observed to be predictable, and the Swiss Hourly Wind Power generation dataset which is not predictable, both datasets are from the same project [15]. We can see that on the Swiss dataset there is no clear pattern, but on the Belgium ACF we have a clear seasonal pattern. If we were looking at this ACF from an ARIMA point of view we would hypothesise that the second dataset is better for us to predict from. ARIMA by itself cannot deal with seasonality and needs to removed by using a Seasonal ARIMA (SARIMA) model. This is a key and distinctive advantage this algorithm has over ARIMA.

As well as being able to deal with one period of seasonality, Alpex is also able to make predictions when we have overlapping seasons. This is seen in the ACF of the Venice Hourly Water Level dataset, which we have previously seen, see Figure 10.9.

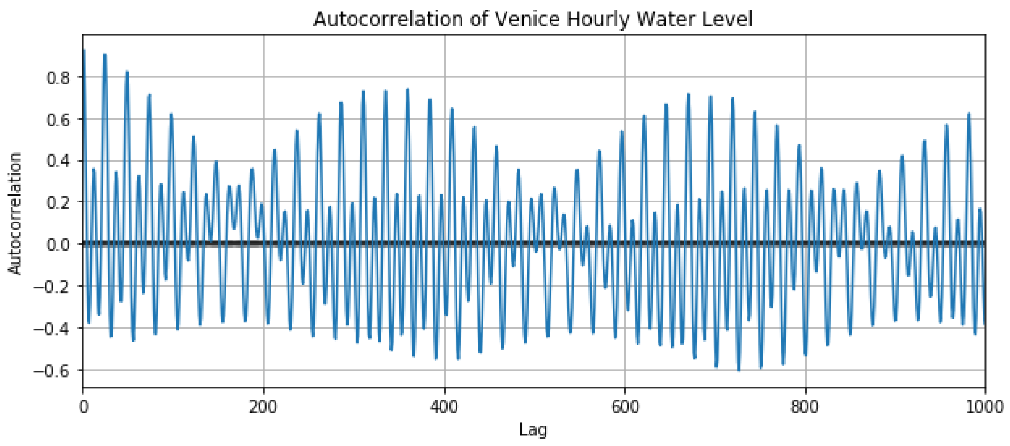


**Figure 10.9:** The Auto Correlation Function of the Venice Hourly Water Level dataset showing overlapping seasonality.

### 10.2.4 Cyclical Trend

A Periodic or Cyclical Trend is very similar to Seasonality and is often mistaken for it [41]. Seasonality is a very formal and rigid concept it says that every set period of time this period will occur, for example: every week on a Friday the bar will be busiest. A Cyclic trend is much more loosely defined, it does not say when these periods occur it just says that we have times in the series when they are followed. The stock market is said to often have a periodic trend where we have times of high and low prices that can follow similar patterns.

We know that the algorithm is able to swap between two different series and quickly pick up the pattern. This same situation occurs with a periodic trend. The algorithm has no knowledge of the current time, it just predicts values based on their current context, this means that it is able to pick up on these periods quickly even if they are not in set time intervals.

This is a really important distinction between our algorithm and SARIMA. SARIMA requires us to have strict periodicity (seasonality) where we know how long each season will last for, for this reason SARIMA will not work on datasets when the periodicity is loosely defined.

### 10.2.5 Coefficient of Variation

The coefficient of variation is a scaled version of the standard deviation. It is defined as being

$$C_v = \frac{\sigma}{\mu}$$

and shows the variation of a population in respect to the mean of the data. During our experiments we found that datasets with a very large or very low Coefficient of Variation we were unable to proactively predict for. The range of values that we were able to predict for were

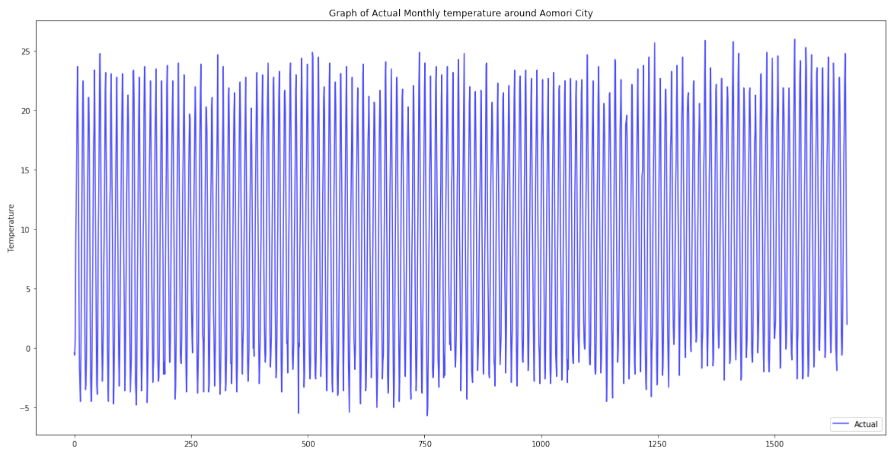


**Figure 10.10:** An example of a dataset with a small sample size but many periods that we are able to forecast for.

in the range of $0.1 < C_v < 500$.

This is not a definitive fact, that we can only predict for datasets in this range, however what it does tell us is that we are unable to predict very volatile datasets.

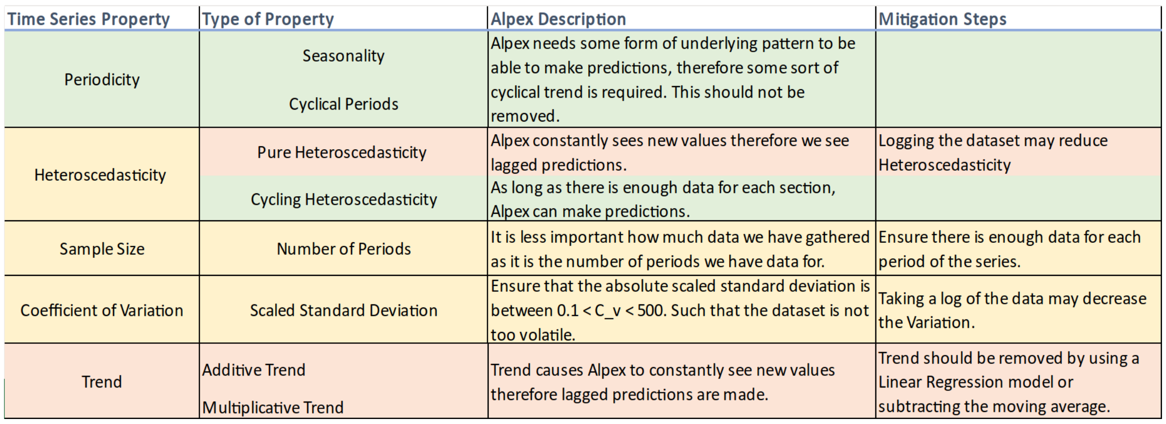
The table of results can be seen in Appendix C.3.

### 10.2.6 Sample Size

One problem this algorithm has when looking at real world datasets is the number of samples that are needed before predictions can be made. ARIMA is able to start making forecasts, albeit not necessarily good ones, with a small amount of samples. Due to the algorithm needing to have previously seen the pattern, to make good predictions we often need several periods before we can make forecasts. Required sample size for this dataset should not be about the number of data points but the number of seasons we observe.

An example of this is in the temperature around Aomori city dataset [28], see Figure 10.10. Here we can see that we only have 1500 data points to predict for, which is not that much, however we have got lots of cycles. Therefore we are able to make proactive forecasts. When analysing the change in metrics over time we can see that we are confidently making proactive forecasts after only 800 time steps.

## 10.3 Table of Properties



| Time Series Property | Type of Property | Alpex Description | Mitigation Steps |
|---|---|---|---|
| Periodicity | Seasonality<br>Cyclical Periods | Alpex needs some form of underlying pattern to be able to make predictions, therefore some sort of cyclical trend is required. This should not be removed. | |
| Heteroscedasticity | Pure Heteroscedasticity | Alpex constantly sees new values therefore we see lagged predictions. | Logging the dataset may reduce Heteroscedasticity |
| | Cycling Heteroscedasticity | As long as there is enough data for each section, Alpex can make predictions. | |
| Sample Size | Number of Periods | It is less important how much data we have gathered as it is the number of periods we have data for. | Ensure there is enough data for each period of the series. |
| Coefficient of Variation | Scaled Standard Deviation | Ensure that the absolute scaled standard deviation is between 0.1 < C_v < 500. Such that the dataset is not too volatile. | Taking a log of the data may decrease the Variation. |
| Trend | Additive Trend<br>Multiplicative Trend | Trend causes Alpex to constantly see new values therefore lagged predictions are made. | Trend should be removed by using a Linear Regression model or subtracting the moving average. |

**Figure 10.11:** A table of properties that Time Series should or should not exhibit for them to be predictable.
Green means the property should or can be seen.
Yellow means it is a property that you need to be wary of.
Red means the property should be removed.

## 10.4 Case Study A: Southern Italy Power Load

One of the datasets we are going to study is the Total load in IT-Centre-South (bidding zone) in MW as published on ENTSO-E Transparency Platform from the Open Power System Data collection [15]. This experiment is going to follow the 'normal' sample experiment format, see Section 6.3.9. For the full experiment report, see Appendix E.1.

The first step in our experiment is to gather the data that we will be forecasting from. This is done by calling the 'generate_data_set()' function from inside our 'normal.py' file. We input the location of the dataset and the column name to this function. We then plot a section of this graph, see Figure 10.12.

Once we have the data generated we need to perform our pre-processing. To do this we call our 'test_stationarity()' function from our 'predictor.py' file. This function plots a graph of the whole dataset with a rolling mean and rolling standard deviation overlaying it, see Figure 10.13. From this diagram we can see that both trend and Heteroskedasticity are present therefore we set our 'remove_trend' and 'log' flags to True.

With the data pre-processed we want to check for periodicity to see if the algorithm can predict from the dataset. To do this we call our 'plot_acf_()' function from our graphing tools. We can see from our ACF that there does appear to be periodicity inside the dataset, see Figure 10.14.

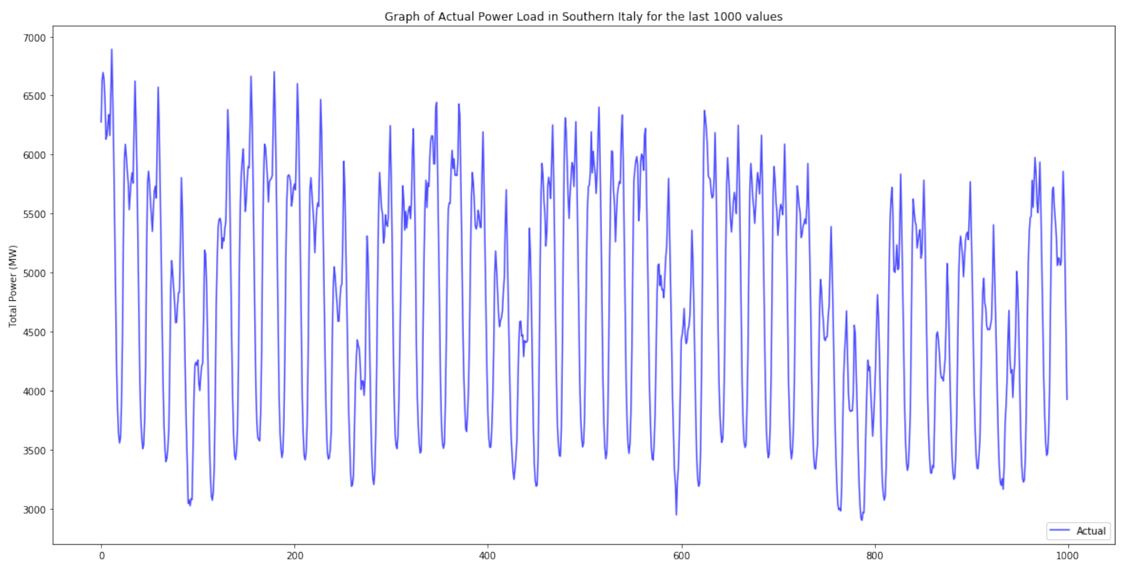


**Figure 10.12:** Case Study A: Visualisation of the Southern Italy Power Load dataset.

Once we have completed pre-processing we need to generate and import our baseline metrics. To do this we call our 'generate_arima()' function with our normalised dataset. This function finds the best (S)ARIMA model for the dataset and outputs the forecasts to a pickled file 'out/arima.obj'. The optimal configuration was found to be SARIMA(3, 0, 2).

Once the data is fully pre-processed we can perform our Alpex predictions using the 'make_predictions_wi function from our predictor tools. This normalises the dataset to the range $[0, max\_value)$, resets the algorithms state and performs predictions.

With the predictor work complete we first want to perform a visual inspection to test our forecasting accuracy. To do this we plot the ARIMA model, the Alpex forecasts and the original dataset overlapping, See Figure 10.15. As well as this we plot the residuals of the models. From this we can see that the algorithm seems to be making proactive forecasts.

To complement our visual representation we also calculate metrics using our Metrics object. Table 11.4, shows the results of this where we can see that Alpex is able to make very good predictions and appears to be doing this proactively with the $R^2_{lag}$ metric being high.

| Model | Mean APE | $R^2$ | Median APE | $R^2_{lag}$ |
|---|---|---|---|---|
| Alpex | 0.5944% | 0.9545 | 0.4228% | 0.6242 |
| Auto ARIMA | 0.6936% | 0.9494 | 0.6032% | 0.5820 |

The last part of our experiment is to ensure that the algorithm is being proactive by plotting the change in metrics over time against the lagged prediction. Figure 10.16 shows the result

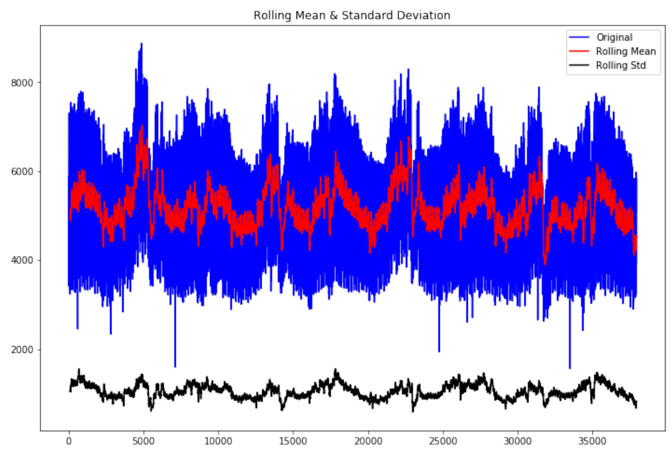


**Figure 10.13:** Case Study A: Stationarity tests for Southern Italy Power Load dataset.

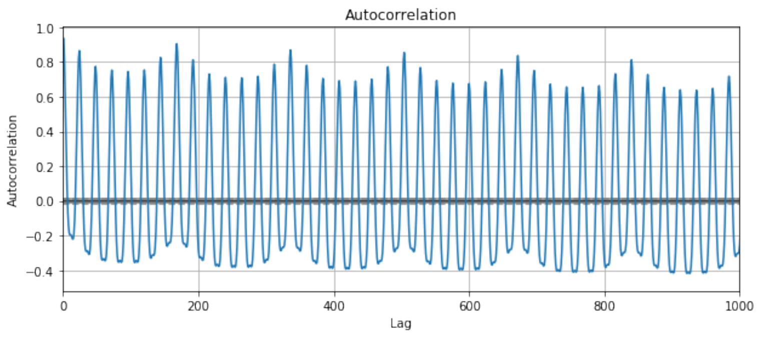


**Figure 10.14:** Case Study A: ACF plot for Southern Italy Power Load dataset.

of this where we can clearly see that our predictions are proactive.

Once our experiment is complete we need to draw conclusions and evaluate the results. What we have seen from this experiment is that the algorithm is not just able to compete with the ARIMA implementation, but it is actually slightly better for this dataset. Considering the numerous advantages Alpex has over ARIMA, this is very promising. One explanation for this could be to do with the periodicity of the dataset. The ACF and zoomed in view of the dataset show a very clear seasonality, however when looking at the dataset as a whole there also seems to be periodicity every 10,000 values. This could be a factor into why Alpex is able to outperform ARIMA in this situation.

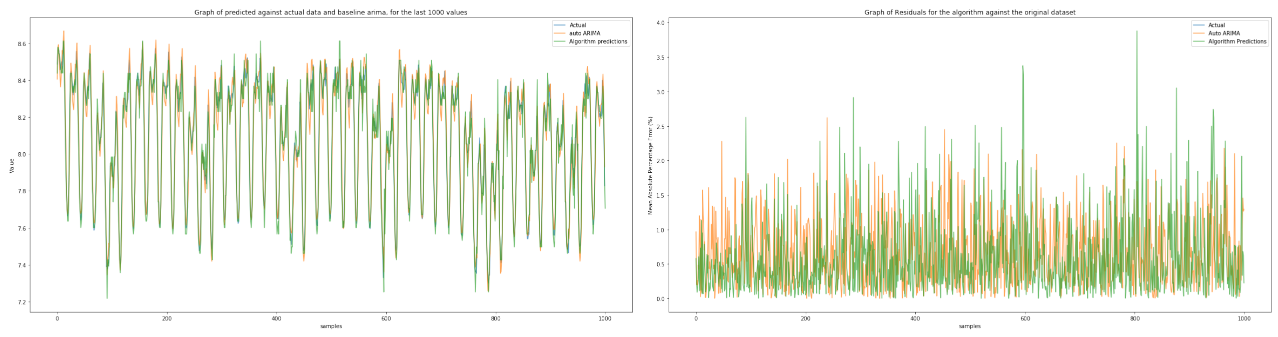


**Figure 10.15:** Case Study A: Graph of predictions for Alpex and ARIMA overlapping dataset values (left), graph of residuals (right).

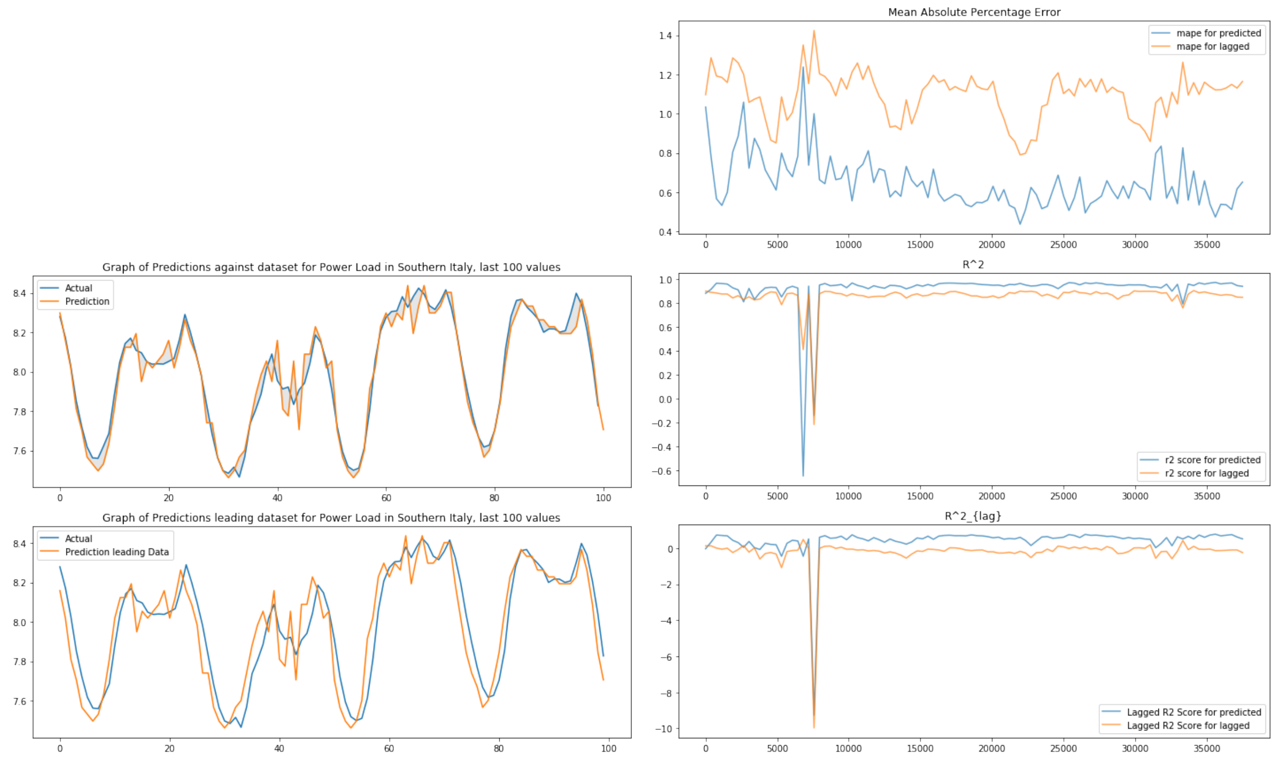


**Figure 10.16:** Case Study A: Graph of predictions leading the dataset (Top) and the plot of change of metrics over time (Bottom) to form a part of our lagged tests.

# Chapter 11

# Evaluation

The aim of the later part of this project was to answer the question "For which classes of time series is Alpex most effective?". We achieved this in two parts; by first studying how Alpex forecasts on Synthetic Time series that we had produced, comparing our predictions against a ground truth, and secondly, by investigating the different Time Series properties of real-world datasets where the underlying structure is hidden. This evaluation aims to compare how Alpexs' predictions, in both contexts, compare against an automatic (S)ARIMA implementation.

## 11.1 Synthetic Analysis

### 11.1.1 Chaotic Time Series

Table 11.1, shows the results of our experiments into the complete range of Mathematical series. From our results, compared to the Auto Regressive model, we can see why the algorithm is so exciting. In many of the series Alpex outperforms against one of the best forecasting methods available today whilst maintaining properties such as computational complexity, processing speed and ease of use. We can see from our study into the Henon Map that as well as being more accurate than ARIMA, the algorithm grasps the chaotic series much faster than the Auto Regressive model. The Mackey-Glass system appears to disprove this, however we can see that Alpex does still perform extremely well.

The results we have seen here could provide valuable insights into the nature of chaotic time series. There is something fundamental about the way this algorithm works that makes it so well suited for Chaotic series, a whole project could be spent analysing what this is.

| Series | Model | Mean APE | $R^2$ | Median APE | $R^2_{lag}$ |
|---|---|---|---|---|---|
| Mackey Glass | Alpex | 0.465524% | 0.999562 | 0.353670% | 0.878088 |
| | Auto ARIMA | 0.021018% | 0.999999 | 0.016771% | 0.999791 |
| Henon Map | Alpex | 2.364417% | 0.999821 | 0.910843% | 0.999936 |
| | Auto ARIMA | 121.506048% | 0.094828 | 83.366962% | 0.677289 |
| Logistic Map | Alpex | 0.37532665% | 0.999917 | 0.2659329% | 0.999974 |
| | Auto ARIMA | 34.699896% | 0.442250 | 25.20355615% | 0.827470 |
| Cos | Alpex | 31.6197468% | 0.999467 | 0.59813% | 0.787909 |
| | Auto ARIMA | 0.46794592% | 1.000000 | 0.0177237% | 0.999987 |
| Sawtooth | Alpex | 1.102% | 0.999940 | 0.389% | 0.998993 |
| | Auto ARIMA | 3.66897% | 0.940523 | 1.7321% | 0.003753 |
| Lorenz System | Alpex | 10.0671% | 0.999633 | 1.68723% | 0.606641 |
| | Auto ARIMA | 0.16988% | 1.000000 | 0.051280% | 0.999835 |

**Table 11.1:** Results of an investigation into Mathematical Time Series.

### 11.1.2 Noise Experiments

The general trend that we saw was that after a sigma value of about 0.75, the algorithm would struggle to perform accurate predictions, and by about 1.0 or 2.0 it would completely break down. This has been recorded in Table 11.2, where it is important to note that the original MAPE and $R^2$ have been taken into consideration when deciding the breakdown points.

Some of the time series would break down because of the $R^2$ value becoming too low, and some would break down because of the Mean Absolute Percentage Error being too high.

This allows us to categorise the series' into two groups, see Figure 11.1. We can see that the smoother graphs were more likely to break down due to the MAPE being too high, however the 'bouncy' graphs were more likely to break down due to the $R^2$ value. If we are looking at real world data, we can gauge an idea of which metric it is likely to fail on due to the property of the series.

We are able to plot these metrics together and see how they compare, see Figures 11.2 and 11.3. Here we can see the groups based on their breakdown point, we can also group the data as follows: Henon Map and Sawtooth, Cos and Lorenz System, Logistic Map and Mackey Glass; based on their metric scores. This is interesting, going back to Figure 11.1, in that there are distinct similarities between the series in each group.

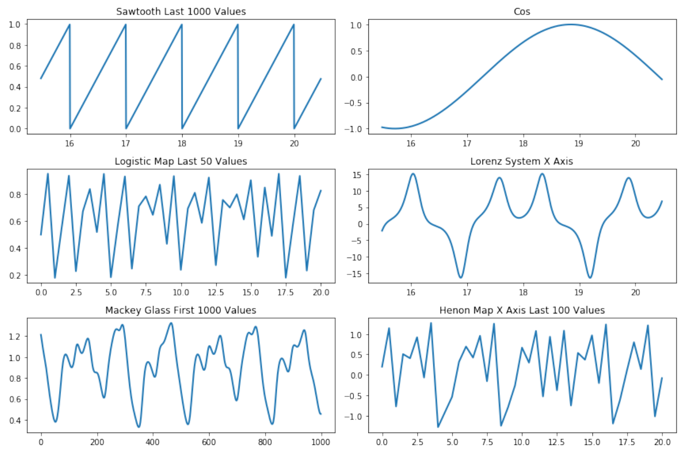


**Figure 11.1:** Time Series Graphs grouped by the breakdown cause, the left hand side being for $R^2$ and the right hand side being MAPE.

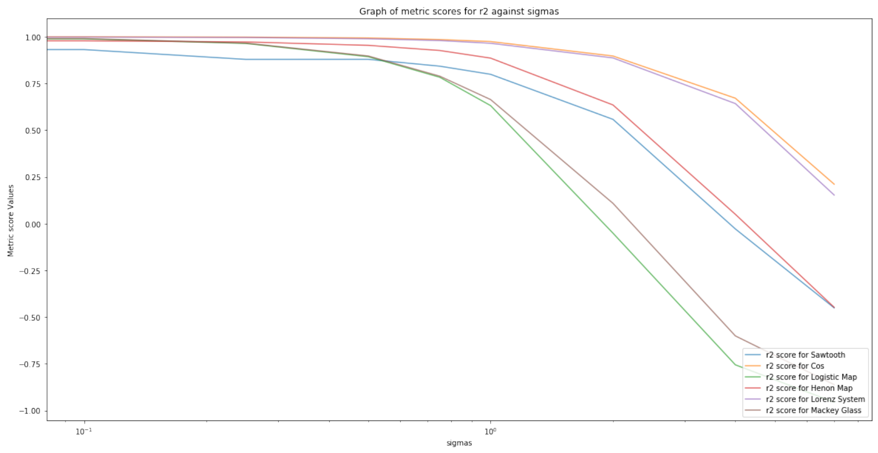


**Figure 11.2:** The graph of $R^2$ against Sigma for all of the time series.

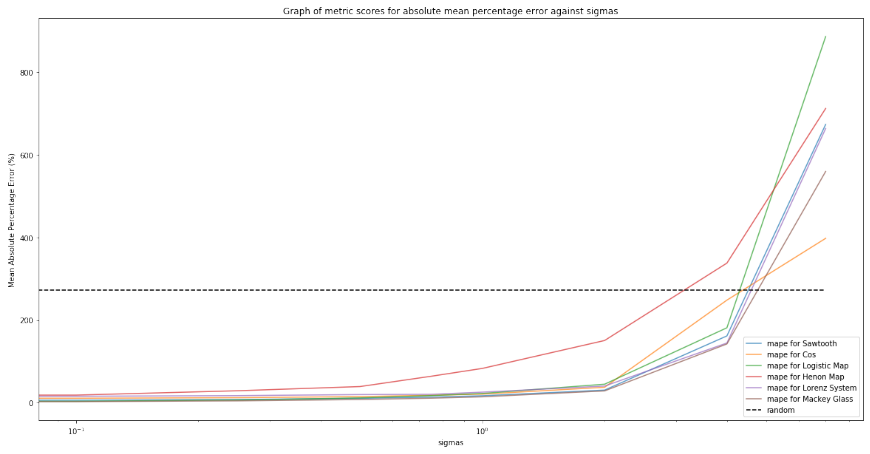


**Figure 11.3:** The graph of Mean Absolute Percentage Error against Sigma for all of the time series.

| Dataset | Predictions are Accurate | | Predictions Breakdown | | Reason |
|---|---|---|---|---|---|
| | Sigma | Dataset Difference | Sigma | Dataset Difference | |
| Sawtooth | 0.75 | 6.02% | 1.0 | 8.05% | $R^2$ |
| Cos | 1.0 | 7.97% | 2.0 | 16.23% | MAPE |
| Logistic Map | 0.5 | 3.96% | 0.75 | 5.85% | $R^2$ |
| Henon Map | 0.5 | 3.98% | 0.75 | 5.98% | MAPE |
| Mackey Glass | 0.75 | 6.04% | 1.0 | 7.97% | $R^2$ |
| Lorenz System | 1.0 | 8.2% | 2.0 | 15.9% | MAPE |
| Avg. | 0.75 | 6.02% | 1.25 | 10.00% | |
| MAPE Avg. | 0.83 | 6.72% | 1.58 | 12.70% | |
| $R^2$ Avg. | 5.34 | 6.02% | 0.92 | 7.29% | |

**Table 11.2:** A table showing the sigma values of where each time series prediction breaks down and the percentage difference the noise adds to the clean dataset.

## 11.2 Real World Analysis

### 11.2.1 Forecasting Accuracy

What we saw from our experiments is that the algorithms performance on datasets that we had been able to proactively forecast, had performance that was similar to SARIMA. We also saw that there were some datasets that SARIMA was able to make forecasts for and Alpex could not, this was usually due to periodicity. Table 11.4 analyses how Alpex's forecasts compare with (S)ARIMA's for a number of different datasets, these include predictable datasets as well as non-predictable.

### 11.2.2 Runtime

In terms of runtime the algorithm has two areas that it might be able to outperform SARIMA. These are configuration time and model fit time.

One of the key features of Alpex is the fact it has no parameters and is therefore non-configurable, it just works. This is very different to SARIMA where specialist training is required to be able to fit an SARIMA model by hand. For this project we mitigated this training by using an Automatic SARIMA model fitter however this only produced best fit models, and not necessarily optimal ones. Furthermore this algorithm performs a slow grid search, with some runtimes being between 20 and 30 minutes.

In terms of the model fit time, SARIMA is thought of as having a linear runtime complexity [40], this is the same as Alpex as there is a constant complexity for each prediction, therefore the complexity for a series of length n is $O(n)$.

Table 11.3, gives an idea of the configuration time for the Auto-Arima implementation, this is a configuration time that we do not have for Alpex. The model fit time is negligible therefore it is not recorded here:

| Series | Sample Size | Configuration Time (s) |
|---|---|---|
| Hourly Total Load in Hungary | 38161 | 161.784 |
| Hourly Total Load in Denmark | 81756 | 497.199 |
| Hourly Total Load in England | 31471 | 186.957 |
| Pump Time Series Data | 7344 | 53.057 |
| Venice Hourly Water Level | 37000 | 168.416 |

**Table 11.3:** Table showing configuration time, including model fit time, for Automatic ARIMA implementation.

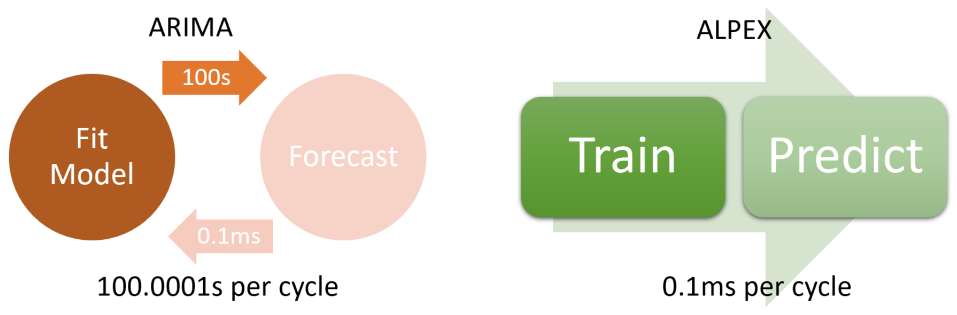


**Figure 11.4:** An example of what could be a Train Prediction cycle for ARIMA (left) and Alpex (right), this is likely on a large dataset.

### 11.2.3 Space Complexity

In terms of space complexity, the ARIMA implementation I was using has at least an exponential complexity $O(n^2)$.

This is shown when I was trying to fit an ARIMA model for the Hourly Total Load in Denmark dataset. Trying to call the function 'fit_arima()' with this dataset resulted in the error message:

```
MemoryError: Unable to allocate 38.3 GiB for an array with shape (71694, 71694)
and data type float64
```

If we are using an array of size $n * n$ then we know our complexity must be at least $O(n^2)$.

This is a huge advantage that Alpex has over this ARIMA implementation. Alpex operates at a constant space complexity $O(1.2GB)$ and will not exceed this no matter the size of the dataset that we are forecasting.

### 11.2.4 Train and Predict Architecture

ARIMA's forecasting speeds have been seen to be comparable to Alpex's, however the model fit time for ARIMA is substantial. With ARIMA, when we get new data we may find that our model is no longer valid and we need to refit it. We have seen that refitting the algorithm takes a substantial amount of time. Alpex trains whilst it predicts new values, this means there is no extra fitting time needed and allows for the algorithm to be extremely adaptable to new information. Figure 11.4 shows the differences in Architecture between the two algorithms.

This limitation of ARIMA limits the agility of models in the face of new data as the implementation might not have enough time to train before new data arrives. Alpex opens up a new level of granularity where we can train on series that generate data at much smaller time intervals. An example of where this could be used is in tick data for stock markets where data arrives too fast for us to be able to make accurate forecasts.

| Series | Model | Mean APE | $R^2$ | Median APE | $R^2_{lag}$ |
|---|---|---|---|---|---|
| Southern Italy Power Load | Alpex | 0.5944% | 0.9545 | 0.4228% | 0.6242 |
| | Auto ARIMA | 0.6936% | 0.9494 | 0.6032% | 0.5820 |
| Hourly Total Load in Austria | Alpex | 140.720078% | 0.961690 | 1.384506% | 0.621234 |
| | Auto ARIMA | 134.242511% | 0.965528 | 0.659179% | 1.320954 |
| Hourly Total Load in Whole of Italy | Alpex | 848.205752 | 0.948082 | 2.021083 | 0.506295 |
| | Auto ARIMA | 746.520618% | 0.967780 | 2.362892% | 0.693607 |
| Hourly Total Load in Hungary | Alpex | 6.282209% | 0.718857 | 4.888864% | -2.147322 |
| | Auto ARIMA | 2.052061% | 0.961229 | 1.509075% | 0.527138 |
| ENTSO-E Hourly Total Load in Bulgaria | Alpex | 302.497282% | 0.656374 | 49.501566% | 0.401014 |
| | Auto ARIMA | 725.249027 | 0.675125 | 50.035069 | 0.433699 |
| ENTSO-E Hourly Total Load in Bosnia | Alpex | 148.380830% | 0.691989 | 24.018268% | 0.410471 |
| | Auto ARIMA | 159.541079 | 0.674314 | 49.865963 | 0.376641 |
| ENTSO-E Hourly Total Load in Denmark | Alpex | 527.86571% | 0.988226 | 10.363896% | 0.884808 |
| | Auto ARIMA | Cannot Allocate enough Memory Error | | | |
| ENTSO-E Hourly Total Load Transparent in England | Alpex | 3.679979% | 0.940073 | 2.378602% | 0.478779 |
| | Auto ARIMA | 3.345916% | 0.958853 | 2.358746% | 0.642121 |
| Venice Hourly Water Level | Alpex | 34.968363% | 0.977826 | 9.973376% | 0.884426 |
| | Auto ARIMA | 16.856310% | 0.991506 | 6.709817 % | 0.955727 |
| ... | | | | | |

**Table 11.4:** A random subset of metrics comparing Alpex prediction accuracy against our Automatic ARIMA implementation.

# Chapter 12

# Conclusion

In this section we are going to review the contributions that I have made to the Alpex algorithm - these include direct code contributions, conclusions made about the properties of the Algorithm as well as informal discussions that allowed for us to gain a deeper understanding of Alpex.

## 12.1 Project Contributions

### 12.1.1 Project Outline

At the beginning of the project, Alpex was a new and exciting rough diamond. We knew that the algorithm was extremely fast in terms of runtime as well as being able to quickly learn chaotic mathematical series. Before the project the algorithm was a collection of procedural C code inside one file. We did not know to what extent the algorithm could learn these chaotic series; how this compared to other methods and how this could be applied to real world series. Therefore the initial specification of the project was to:

- Create a C++ API to handle streaming of data that was robust, extensible and object orientated; making Alpex accessible.
- To conduct an investigation into the algorithm to determine which classes of time series it is most effective at forecasting.

As well as the original specification of the project we also achieved multiple extensions. Some of these were to meet problems that appeared in the project, some were a result of analysing experiments and then others were due to my own curiosity. Each extension added a new and unique contribution to the project.

### 12.1.2 Software Engineering Element

The contributions that were made on the software engineering side were:

- A C++ API was created that allowed for the streaming of data in and out with the use of Objects. This was completed with speed, extend-ability, security and accessibility in mind. I also extended upon the API for:
  - A Python wrapper such that the algorithm could be used from a data analytical environment.
  - A command line executable where users could input files or stream data using standard input or standard output.
  - Example pseudocode for extending the functionality of the algorithm without changing the API.
- A Time Series as a Service Web Application consisting of:
  - A Containerised implementation of the API in C++ that used sockets to allow clients to make API requests and maintained state independently.
  - A Kotlin based Spring Boot RESTful Web App. Due to the RESTful nature and containerisation this is scalable, and can handle simultaneous users communicating with different algorithm state machines.
  - A Swagger generated Python library that allowed for communicating with the Web API using functions wrapping the algorithms complexity.
- A suite of tools for analysing Time Series in Python:
  - A Metrics object for calculating a large array of metrics that a user may want to calculate, as well as tools for graphing these, displaying them in a table and exporting to LaTeX.
  - Graphing tools aimed at Time Series experiments.
  - Interchangeable implementations of the API in a local mode where the user makes the predictions and server mode where the user communicates with our server.
  - An array of example experiments that non programmers could use to make forecasts using the algorithm.
- I created a suite of Python command line programs for generating Phylogenetic representations of the algorithm. This allowed for us to draw conclusions about the redundancy of the algorithm. The graphs also look amazing and are an exciting contribution to the project.

### 12.1.3 Analysis Side

As well as my extensive contribution to the Software Engineering side of the Algorithm, I also made contributions both in terms of the required analysis as well as separate experiments that I developed whilst completing the project.

- Experiments into Chaotic Time Series and comparing these to ARIMA models.
- An extension experiment into an emergent property of the algorithm that caused spikes to appear in the predictions.
- Whilst coming up with solutions for the spikes in the predictions we explored different ways that data can be pre-processed and normalised such that the algorithm is able to make predictions, this includes using tanh and sigmoid activation functions.
- An investigation into the noise tolerance of the algorithm as well as exploring different ways the algorithm could forecast in the presence of noise.
- I developed a new series of tests, including a new metric, that we could use to ensure that the algorithm was making proactive forecasts.
- An analysis into different time series properties that a real world time series might exhibit for it to be predictable by Alpex. This includes periodicity and compares this to ARIMA models, drawing conclusions between the two. This allowed us to create a table of properties that should be referred to before making predictions.
- As well as all the formal contributions, many experiments were conducted into different time series and although many of these did not make it into the final report they did form discussions that have helped Ben, Will and I to understand the algorithm in novel ways.

## 12.2 Challenges

This project presented itself with a unique challenge in terms of time and resource management. As the project had no clear finish line it was impossible to judge if, at any point, I was on track. Coupled with this the many different directions the project went, and could have gone, were endless and made me want to explore each one of these further. Furthermore during the analysis stage we could have gone into near endless detail on each section, it was therefore crucial that we found a point that balanced: moving towards the final goal, allowing for the exploration of tangents and ensuring that each section was explored in enough detail. I dealt with this unique challenge by making sure that I set out clear goals for what I aimed to achieve in a week, and ensuring that both project Supervisors approved these and could therefore hold me accountable in the next meeting.

With the analysis of the project being so open ended, we had to make a lot of different decisions about what directions the project should go in and what would be worth exploring.

Some of these decisions did not bear fruit and were abandoned after some time. Therefore an important challenge I had to overcome was deciding when to stop chasing a lead and concentrate on a different experiment.

Due to the security of the project every attempt was made to ensure the algorithm was not compromised. This meant that files had to be encrypted; the language carefully chosen and that the code did not appear online. This meant that I needed to implement my own infrastructure in my own local network. Implementing custom Git Servers, running the Web APP components on another device in the same network and ensuring that all of this worked between a windows laptop and a Linux PC. These infrastructure challenges were a great learning experience and posed unique challenges that I did not necessarily expect when starting this project.

As well as all of the regular challenges that I needed to overcome, there was also the very unique challenge imposed by 2020. The project was almost entirely completed during lock down inside a pandemic. This meant that physical resources were unavailable, meetings could not take place face to face and the general mental pressure caused by the situation was unprecedented. Unique solutions had to be sought for completion during this special time.

The Web Server had a very interesting architectural challenge of trying to create a stateless server that was able to maintain state. The solution to this, which was discussed, was about isolating the risk of a stateful server by separating the stateful components and placing them inside their own Web API. This balancing act between usability and scalability presented itself in an interesting scenario and allowed me to gain insights into the drawbacks of RESTful servers as well as the benefits of a micro-service architecture in dealing with this.

Before this project I had worked as an intern in Data Development for a Financial Tech company. During this internship I gained an understanding of how the data was gathered for time series and how it could be manipulated at scale. This project therefore allowed me to apply this knowledge and gave me great insight into what happens to the data further down the chain. This posed a challenge as I had to apply some of my knowledge on how this is done at a huge scale but adapt it to an isolated situation, I also had to deal with biases I had towards the analysis of financial series and ensure this did not affect a well balanced experiment.

Finally this project required a deep understanding of how ARIMA works and the different ways it can be applied. This also included the pre-processing and data analysis that is required. Leaning about Auto Regression and the role that auto correlation plays in the nature of series and the fact that we can measure, using the Auto Correlation Function, how values are influenced by previous values in a series is particular interesting. Expanding this to see how we can detect periods in the ACF of a series and how this leads to seasonality and Auto Regression models.

## 12.3 Future Work

### 12.3.1 How can we make Alpex accessible and deployable?

Although my contribution to this element was more than the initial specification dictated, there is still a lot of work that could be done. While creating the API's and different tools this future work was considered, therefore the path to some of it has already been laid.

- The implementation of the original algorithm had the memory and the process completely intertwined. Moving towards an Object Orientated approach I began to unravel this until we could create the Object based API. I began looking into completely separating the memory into its own object, beginning to change the code and creating a 'bare bones' pseudocode representation of the algorithm, however this was not a priority of the project so it was never finished. This would be a nice extension to the project as it would allow for users to innovate on the memory and the algorithm separately.
- One of the reasons I began separating the memory manipulation from the process was to pave the way for a multi-threaded implementation of the algorithm. This is something I would like to have done and I hypothesised the ways we could have done this.
- Once the algorithm was multi-threaded, the containerised C++ sockets would had paved the way for us to easily convert the algorithm into a distributed implementation where we could split the memory requirements over multiple servers.
- Although we created the Web API and a client library for the Time Series as a Service web app, an extension for this would have been to create a complete GUI for this such that clients could easily input data into the server and make predictions. This could be done in a way that is similar to how Causal Lens [8] offers up methods for time series forecasting.
- As well as creating a complete Web app that has a GUI clients can interact with, an extension to the project would be a way to automatically pre-process data and make predictions based on that. More research would need to be done in terms of pre-processing methods for this to be useful.
- An exciting extension to this project would have been to actually implement the algorithm into some sort of prediction feedback loop. For example one type of time series we look at is temperature, if we can accurately forecast temperature changes then we can be proactive with our heating solutions and be more energy efficient, it would be interesting to implement the algorithm in one of these systems and measure how much energy is saved. Following on from this another interesting application would be if we found a stock series the algorithm could predict for, we could implement the algorithm inside an automated trading simulator and see if the algorithm would be able to earn money.
- If we were to separate the memory from the procedure then a study could be conducted into different memory optimisations to try and make the algorithm overfit less. For example we could look at different pruning methods.

- One of the possibilities of this project would be to implement a Random Forest. This could allow us to reduce overfitting for the algorithm.
- Although we created a few Python programs that allowed for us to create Phylogenetic trees of the algorithms state, an exciting extension would be to try to find a way to have these trees automatically generate whilst the algorithm is running.
- As mentioned in the design of the C++ sockets, we decided to use sockets for our implementation as opposed to Kafka event queues as they were outside the scope of the project. One extension would be to implement our C++ predictors in this way.

### 12.3.2 For which classes of time series is Alpex most effective?

The analysis side of the project really just touched the surface in terms of what the algorithm can do. We managed to prove that there is a place where Alpex could be used in the real world, however there is still lots of work that can be done into studying this algorithm and I believe one of the key takeaways of this project is the fact we have been able to highlight some of the areas where future work could be most beneficial.

- We know that the algorithm operates under a constant space complexity and we know that in terms of runtime the algorithm is extremely fast, however we do not know how fast this is compared to other algorithms. An experiment that could be performed is to formally compare the computational requirements of Alpex compared to other metrics.
- Following on from the point about computational complexity, I began analysing the runtime of the algorithm and conducted some experiments into how this changes compared to sample size, see Appendix D.1. I noticed that for some of the real time series that we could not forecast for, there was a point where the runtime rapidly increased. A future experiment could explore why this happens and if this relates to any of our time series properties.
- When comparing the algorithm against a benchmark we focused on an ARIMA implementation, for a more thorough analysis of the algorithms performance against others we would want to compare the algorithm against other techniques. These would include machine learning techniques or packaged solutions such as Amazons Forecasting service.
- For some datasets we can gather lots of similar series, for example in the power generation sets we studied we have data available for lots of different Countries. As the algorithm learns from seeing past patterns it would be interesting to study how the performance of the algorithm could be improved by first training it on similar series.
- When studying different ways to predict in the presence of noise, we discovered that by first training the algorithm on clean data and then noisy data we were able to see a consistent increase in prediction accuracy. We hypothesised that using this we could apply a smoothing technique to data before making predictions, this may increase accuracy of datasets.

- We noticed that the algorithm performs well when there is an underlying pattern to the dataset. An investigation could be completed to try and categorise and analyse these patterns, we could answer the question, “are there patterns that the algorithm performs better or worse for?”.
- Although we looked at well over 50 datasets to generate our time series properties, there are always more datasets that could be predicted for. An example of a dataset I would have liked to analyse is the possibility of the algorithm to predict intraday stock prices. Intraday prices are said to experience periodicity therefore the algorithm may be able to predict well in this setting.
- We looked at some different normalisation techniques to map an infinite range of values into a predictable range, a useful extension would be to properly study these techniques and analyse how the performance of the algorithm changes with these.
- Although we briefly analysed the tree visualisations and we drew some conclusions from these about the redundancy of the algorithm, it may be useful to conduct a full investigation into these to try and learn how and why they work.
- One avenue we began to explore with Alpex was t+n predictions. All of the experiments that we conducted concerned themselves with t+1 predictions. It would be an interesting investigation to analyse how the algorithm performs in this scenario. Our Analysis Toolkit supports this.

# Bibliography

[1] Amazon Timestream. pages 26

[2] Apache Kafka. pages 46

[3] argparse — Parser for command-line options, arguments and sub-commands — Python 3.8.3 documentation. pages 56

[4] Auto-train a time-series forecast model - Azure Machine Learning — Microsoft Docs. pages 26

[5] Azure Machine Learning documentation — Microsoft Docs. pages 26

[6] Boost C++ Libraries. pages 31

[7] Boost.Python - 1.66.0. pages 39

[8] causaLens - A Machine that Predicts the Global Economy in Real-Time. pages 25, 123

[9] Choosing an Amazon Forecast Algorithm - Amazon Forecast. pages 26

[10] CMake. pages 32

[11] Containerization Explained — IBM. pages 47

[12] CPP / C++ Notes - Doxygen - Documentation Generator. pages 34

[13] ctest(1) — CMake 3.17.3 Documentation. pages 36

[14] Daily and monthly sunspot number (last 13 years) — SILSO. pages 13

[15] Data Platform – Open Power System Data. pages 13, 99, 103, 104, 105, 108

[16] DeepAR+ Algorithm - Amazon Forecast. pages 26

[17] Empowering App Development for Developers — Docker. pages 47

[18] Enterprise IT executives expect 60% of workloads will run in the cloud by 2018 - 451 Research - 451 Research - Analyzing the Business of Enterprise IT Innovation. pages 25

[19] ETE Toolkit - Analysis and Visualization of (phylogenetic) trees. pages 67

[20] Financial, Economic and Alternative Data — Quandl. pages 56

[21] Forecasting. pages 25

[22] google/googletest: Googletest - Google Testing and Mocking Framework. pages 31

[23] GotW #100: Compilation Firewalls (Difficulty: 6/10) – Sutter's Mill. pages 33

[24] Heteroskedasticity Definition. pages 7

[25] ˆIXIC 9,837.50 23.42 0.24% : Nasdaq - Yahoo Finance. pages 101

[26] Machines are driving Wall Street's wild ride, not humans. pages 1

[27] Microsoft.VisualStudio.TestTools.CppUnitTestFramework API - Visual Studio — Microsoft Docs. pages 31

[28] Monthly temperature around Aomori City — Kaggle. pages 107

[29] NumPy. pages 55

[30] os — Miscellaneous operating system interfaces — Python 3.8.3 documentation. pages 56

[31] pandas - Python Data Analysis Library. pages 55

[32] PImpl - cppreference.com. pages 33

[33] pmdarima · PyPI. pages 56

[34] Project Jupyter — Home. pages 54

[35] pybind/pybind11: Seamless operability between C++11 and Python. pages 39

[36] PyPDF2 · PyPI. pages 56

[37] random — Generate pseudo-random numbers — Python 3.8.3rc1 documentation. pages 89

[38] scikit-learn: machine learning in Python — scikit-learn 0.23.1 documentation. pages 55

[39] Spring — Home. pages 50

[40] time series - What is the computational complexity of arima() function in R? - Stack Overflow. pages 116

[41] time series - What is the difference between period cycle and seasonality? - Cross Validated. pages 106

[42] Time Series Forecasting — Machine Learning — Amazon Forecast. pages 26

[43] Tutorial: Multistep Forecasting with Seasonal ARIMA in Python - Data Science Central. pages 17

[44] Tutorial: Understanding Linear Regression and Regression Error Metrics. pages 28

[45] Water Levels in Venezia, Italia — Kaggle. pages 97

[46] Wrapping C/C++ for Python — Intermediate and Advanced Software Carpentry 1.0 documentation. pages 39

[47] A. E. Amin. A novel classification model for cotton yarn quality based on trained neural network using genetic algorithm. *Knowledge-Based Systems*, 39:124–132, 2 2013. pages 21

[48] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000. pages 21

[49] Geoff Boeing. Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction. *Systems*, 4(4):37, 11 2016. pages 6, 10, 11

[50] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing*, 3(4):449–458, 2015. pages 1

[51] Raymond J Carroll. Adapting for Heteroscedasticity in Linear Models. *The Annals of Statistics*, 10(4):1224–1233, 1982. pages 7

[52] Ma Dan. The business model of "Software-As-A-Service". In *Proceedings - 2007 IEEE International Conference on Services Computing, SCC 2007*, pages 701–702, 2007. pages 25

[53] David Eppstein. H tree fractal, 5 2007. pages 65

[54] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Technical report. pages 60

[55] Thomas Ertl, Ken Joy, Beatriz Santos, Petra Neumann, Sheelagh Carpendale, and Anand Agarawala. PhylloTrees: Phyllotactic Patterns for Tree Layout. Technical report, 2006. pages 66

[56] Julian Faraway and Chris Chatfield. Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(2):231–250, 6 2008. pages 9

[57] H Glejser. A New Test for Heteroskedasticity. 2012. pages 7

[58] Stephen M. Goldfeld and Richard E. Quandt. Some Tests for Homoscedasticity. *Journal of the American Statistical Association*, 60(310):539–547, 1965. pages 7, 9

[59] Celso Grebogi, Edward Otr, and James A Yorxe. Chaos, Strange Attractors, and Fractal Basin Boundaries in Nonlinear Dynamics Downloaded from. Technical Report 1, 2020. pages 10

[60] Siana Halim and Indriati N. Bisono. Automatic seasonal auto regressive moving average models and unit root test detection. *International Journal of Management Science and Engineering Management*, 3(4):266–274, 2008. pages 18

[61] Seng Hansun. A new approach of moving average method in time series analysis. In *2013 International Conference on New Media Studies, CoNMedia 2013*, 2013. pages 15

[62] J D Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. pages 55

[63] KPMG International. Journey to the cloud The creative CIO Agenda. Technical report, 2017. pages 25

[64] J. W. Taylor, P. E. McSharry, Senior Member, IEEE, and R. Buizza. Wind Power Density Forecasting Using Ensemble Predictions and Time Series Models. Technical report, 2009. pages 1

[65] Weikuan Jia, Dean Zhao, Tian Shen, Yuyang Tang, and Yuyan Zhao. Study on optimized Elman neural network classification algorithm based on PLS and CA. *Computational Intelligence and Neuroscience*, 2014, 2014. pages 22

[66] Timo Koskela, Mikko Lehtokangas, Jukka Saarinen, and Kimmo Kaski. Time Series Prediction with Multilayer Perceptron , FIR and Elman Neural Networks. *In Proceedings of the World Congress on Neural Networks*, pages 491–496, 1996. pages 23

[67] George Lawton. Developing software online with platform-as-a-service technology. *Computer*, 41(6):13–15, 6 2008. pages 25

[68] Henry Leung and Simon Haykin. The Complex Backpropagation Algorithm. *IEEE Transactions on Signal Processing*, 39(9):2101–2104, 1991. pages 21

[69] Xiuming Li, Tianyi Zhao, Jili Zhang, and Tingting Chen. Predication control for indoor temperature time-delay using Elman neural network in variable air volume system. *Energy and Buildings*, 154:545–552, 11 2017. pages 24

[70] Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 3 1963. pages 12

[71] Michael C. Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 7 1977. pages 12

[72] Abbas Majdi and Morteza Beiki. Evolving neural network using a genetic algorithm for predicting the deformation modulus of rock masses. *International Journal of Rock Mechanics and Mining Sciences*, 47(2):246–253, 2 2010. pages 21

[73] Michael D. Mundt, W. Bruce Maguire, and Robert R. P. Chase. Chaos in the sunspot cycle: Analysis and prediction. *Journal of Geophysical Research: Space Physics*, 96(A2):1705–1716, 2 1991. pages 13

[74] Eduardo H.M. Pena, Marcos V.O. De Assis, and Mario Lemes Proença. Anomaly Detection Using Forecasting Methods ARIMA and HWDS. In *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, volume 0, pages 63–66. IEEE Computer Society, 7 2013. pages 17

[75] Elizabeth Pennisi. First comprehensive tree of life shows how related you are to millions of species. *Science*, 9 2015. pages 66

[76] Fred L. Ramsey. Characterization of the Partial Autocorrelation Function. *Annals of Statistics*, 2(6):1296–1301, 1974. pages 8

[77] D. J. SCHIMPF, S. D. FLINT, and I. G. PALMBLAD. Representation of Germination Curves with the Logistic Function. *Annals of Botany*, 41(6):1357–1360, 11 1977. pages 10

[78] Manoj Kumar Shukla and Bharat Bhushan Sharma. Investigation of chaos in fractional order generalized hyperchaotic Henon map. *AEU - International Journal of Electronics and Communications*, 78:265–273, 8 2017. pages 11, 12

[79] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. In *Proceedings - International Conference on Distributed Computing Systems*, pages 328–339. Institute of Electrical and Electronics Engineers Inc., 7 2017. pages 25

[80] Malcolm E. Turner, Brent A. Blumenstein, and Jeanne L. Sebaugh. 265 Note: A Generalization of the Logistic Law of Growth. *Biometrics*, 25(3):577, 9 1969. pages 10

[81] Elif Derya Übeyli and Mustafa Übeyli. *7 Case Studies for Applications of Elman Recurrent Neural Networks*. pages 24

[82] J. R. Wallis and N. C. Matalas. Correlogram Analysis Revisited. *Water Resources Research*, 7(6):1448–1459, 12 1971. pages 8

[83] M A Wani and D T Pham. Efficient control chart pattern recognition through synergistic and distributed artificial neural networks. Technical report, 1999. pages 25

[84] Halbert White. A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity. *Econometrica*, 48(4):817, 5 1980. pages 6

[85] B. A. Wichmann and I. D. Hill. Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator. *Applied Statistics*, 31(2):188, 1982. pages 89

[86] J. Wise. The Autocorrelation Function and the Spectral Density Function. *Biometrika*, 42(1/2):151, 6 1955. pages 8

[87] W. K. Wong, Min Xia, and W. C. Chu. Adaptive neural network model for time-series forecasting. *European Journal of Operational Research*, 207(2):807–816, 12 2010. pages 22

[88] Kun Xie, Hong Yi, Gangyi Hu, Leixin Li, and Zeyang Fan. Short-term power load forecasting based on Elman neural network with particle swarm optimization. *Neurocomputing*, 2019. pages 24

[89] Xiaomin Xu, Sheng Huang, Yaoliang Chen, Kevin Brown, Inge Halilovic, and Wei Lu. TSaaaS: Time series analytics as a service on IoT. In *Proceedings - 2014 IEEE International Conference on Web Services, ICWS 2014*, pages 249–256. Institute of Electrical and Electronics Engineers Inc., 2014. pages 26

# Appendices

# Appendix A

# Alpex C++ Library

## A.1 API Contract

# Alpex Class Reference

A time series forecaster using the **Alpex** Algorithm. More...

```
#include <alpex.h>
```

## Public Member Functions

| | |
|---|---|
| void | **reset** (int initial)<br>Resets the state of the predictor. More... |
| int | **predict** (int value)<br>Trains the algorithm on the inputted value, then predicts the next value. More... |
| bool * | **train** (int value, bool *hst)<br>Trains the algorithm on the inputted value. More... |
| int | **predict_without_training** (int value, bool *hst)<br>Performs a T+1 prediction from the history array, using the current state of the algorithm. More... |
| bool * | **add_value_to_history** (int value, bool *hst)<br>Adds a value to the history array. More... |
| | **Alpex** ()<br>Creates a Predictor with default parameters. More... |
| | **Alpex** (int initial)<br>Creates a Predictor with an initial value and default parameters. More... |
| | **Alpex** (int initial, int max_value, int pool_scaler)<br>Creates a Predictor with an initial value. More... |
| | **Alpex** (const **Alpex** &other) |
| **Alpex** & | **operator=** (**Alpex** rhs) |

## Detailed Description

A time series forecaster using the **Alpex** Algorithm.

Forecasts time series based on an inputed sequence using the novel **Alpex** time series prediction algorithnm. This class is the predictor used for forecasting using this method. The MAX_VALUE is set by the constructor, this value is defaulted to 255 and is exclusive.

Example:

```
Alpex alpex = Alpex(1);
alpex.train(1); alpex.train(2); alpex.train(3);
assert(2, alpex.predict(1));
```

**Author**

Sam Brotherton

## Constructor & Destructor Documentation

### ◆ Alpex() [1/3]

Alpex::Alpex ( )

Creates a Predictor with default parameters.

**Warning**
If this is used you should call reset with an initial value before use.

The default parameters are:

- initial = 0
- max_value = 255
- pool_scaler = 8000000

### ◆ Alpex() [2/3]

Alpex::Alpex ( int initial )

Creates a Predictor with an initial value and default parameters.

**Precondition**
The initial value is in the range [0, 255).

The default parameters are:

- max_value = 255
- pool_scaler = 8000000

**Warning**
Throws OutOfRange Exception if initial is in the incorrect range.

### ◆ Alpex() [3/3]

```
Alpex::Alpex ( int  initial,
               int  max_value,
               int  pool_scaler
             )
```

Creates a Predictor with an initial value.

**Warning**
max_value is exclusive (the range is [0, max_value)).

**Precondition**
The initial value is in the range [0, max_value).

The pool_scaler is a large positive integer (the default is 8000000).

**Warning**
Throws OutOfRange Exception if initial is in the incorrect range.

## Member Function Documentation

### ◆ add_value_to_history()

```
bool* Alpex::add_value_to_history ( int     value,
                                    bool *  hst
                                  )
```

Adds a value to the history array.

**Precondition**
The value must be in the range [0, MAX_VALUE).

The history must be of the correct length or a nullptr.

**Parameters**

| | |
|---|---|
| **value** | The previous value in the sequence. |
| **history** | The binary representation of the history of values, nullptr can be used to use the predictors history. |

**Returns**
The history array with the added value.

**Warning**
Throws OutOfRange Exception if value is in the incorrect range.

### ◆ predict()

```
int Alpex::predict ( int  value )
```

Trains the algorithm on the inputted value, then predicts the next value.

**Precondition**

The value must be in the range [0, MAX_VALUE).

**Parameters**

| | |
|---|---|
| **value** | The previous value in the sequence. |

**Returns**

The T+1 prediction for the new value.

The predictor first trains on the inputted value, it then performs a T+1 prediction on the value. The predictor trains from it's current state.

**Warning**

Throws OutOfRange Exception if value is in the incorrect range.

### ◆ predict_without_training()

```
int Alpex::predict_without_training ( int     value,
                                      bool *  hst
                                    )
```

Performs a T+1 prediction from the history array, using the current state of the algorithm.

**Precondition**

The value must be in the range [0, MAX_VALUE).

The history must be of the correct length.

**Parameters**

| | |
|---|---|
| **value** | The previous value in the sequence. |
| **history** | The binary representation of the history of values. |

**Returns**

The T+1 prediction for the new value.

To ensure the history is of correct length, it should be obtained from a train call, and only added to with a add_value_to_history call.

**Warning**

Throws OutOfRange Exception if value is in the incorrect range.

## ◆ reset()

```
void Alpex::reset ( int  initial )
```

Resets the state of the predictor.

**Precondition**

The initial value must be in the range [0, MAX_VALUE).

**Parameters**

**initial** The initial value to set the history to.

**Warning**

Throws OutOfRange Exception if initial is in the incorrect range.

## ◆ train()

```
bool* Alpex::train ( int     value,
                     bool *  hst
                   )
```

Trains the algorithm on the inputted value.

**Precondition**

The value must be in the range [0, MAX_VALUE).

**Parameters**

**value** The previous value in the sequence.

**Returns**

The binary representation of the history of the algorithm.

The predictor trains on the inputted value. The algorithm returns the current state of the history with the added value. This can be used as a parameter for **predict_without_training()**.

**Warning**

Throws OutOfRange Exception if value is in the incorrect range.

The documentation for this class was generated from the following file:

- include/**alpex.h**

## A.2 Doxygen Documented Header File

Due to the fact the library code cannot be included in the final repository, I have included the header file to show how PIMPL and Doxygen commenting works in the Appendix.

**Listing A.1:** Code for the Doxygen documented source file Alpex.h implementing the pimpl idiom.

```
#ifndef _ALPEX_ALGORITHM_H_
#define _ALPEX_ALGORITHM_H_
/**
 *  @file    alpex.h
 *  @brief   Alpex Algorithm API
 *  @author Ben Rogers and Sam Brotherton
 *  @date    2020-05-16
 ********************************************/


#pragma once
#include <random>
#include <memory>

/**
* @brief A time series forecaster using the Alpex Algorithm.
* @details
*  Forecasts time series based on an inputed sequence using the novel
*  Alpex time series prediction algorithnm. This class is the predictor used
*  for forecasting using this method.
*  The MAX_VALUE is set by the constructor, this value is defaulted to 255 and
*  is exclusive.
*
* Example:
* @code
*   Alpex alpex = Alpex(1);
*   alpex.train(1); alpex.train(2); alpex.train(3);
*   assert(2, alpex.predict(1));
* @endcode
* @author Sam Brotherton
********************************************/
class Alpex {
public:
    // ----------------Core Functionality----------------
    /**
    * @brief Resets the state of the predictor.
    *
    * @pre The initial value must be in the range [0, MAX_VALUE).
    *
    * @param initial The initial value to set the history to.
    * @warning Throws OutOfRange Exception if initial is
    *   in the incorrect range.
    */
    void reset(int initial);

    /**
    * @brief Trains the algorithm on the inputted value,
    *   then predicts the next value.
    *
    * @pre The value must be in the range [0, MAX_VALUE).
    *
    * @param value The previous value in the sequence.
    * @return The T+1 prediction for the new value.
    *
    * @details
    *  The predictor first trains on the inputted value, it then
    *   performs a T+1 prediction on the value. The predictor trains
    *   from it's current state.
    * @warning Throws OutOfRange Exception if value
    *   is in the incorrect range.
    */
    int predict(int value);

    /**
    * @brief Trains the algorithm on the inputted value.
    *
    * @pre The value must be in the range [0, MAX_VALUE).
    *
    * @param value The previous value in the sequence.
    * @return The binary representation of the history of
    *   the algorithm.
    *
    * @details
    *  The predictor trains on the inputted value. The algorithm
    *   returns the current state of the history with the added
    *   value. This can be used as a parameter for
    *   predict_without_training().
    * @warning Throws OutOfRange Exception if value is in
    *   the incorrect range.
    */
    bool *train(int value, bool *hst);

    // ------------------Optional Functionality------------------

    /**
    * @brief Performs a T+1 prediction from the history array,
    *   using the current state of the algorithm.
    *
    * @pre The value must be in the range [0, MAX_VALUE).
    * @pre The history must be of the correct length.
    *
    * @param value The previous value in the sequence.
    * @param history The binary representation of the history of values.
    * @return The T+1 prediction for the new value.
    *
    * @details
    *  To ensure the history is of correct length, it should be
    *   obtained from a train call, and only added to with a
    *   add_value_to_history call.
    * @warning Throws OutOfRange Exception if value is in
    *   the incorrect range.
    */
    int predict_without_training(int value, bool *hst);

    /**
    * @brief Adds a value to the history array.
    *
    * @pre The value must be in the range [0, MAX_VALUE).
    * @pre The history must be of the correct length or a nullptr.
    *
    * @param value The previous value in the sequence.
    * @param history The binary representation of the history
    *   of values, nullptr can be used to use the predictors history.
    * @return The history array with the added value.
    *
    * @warning Throws OutOfRange Exception if value is
    *   in the incorrect range.
    */
    bool *add_value_to_history(int value, bool *hst);

    // ----------------Constructors----------------
    /**
    * @brief Creates a Predictor with default parameters.
    *
    * @warning If this is used you should call reset with
    *   an initial value before use.
    * @details
    *  The default parameters are:
    *    - initial = 0
    *    - max_value = 255
    *    - pool_scaler = 8000000
    */
    Alpex();

    /**
    * @brief Creates a Predictor with an initial value
    *   and default parameters.
    *
    * @pre The initial value is in the range [0, 255).
    * @details
    *  The default parameters are:
    *    - max_value = 255
    *    - pool_scaler = 8000000
```

```
     * @warning Throws OutOfRange Exception if initial
     *   is in the incorrect range.
     */
     Alpex(int initial);

     /**
     * @brief Creates a Predictor with an initial value.
     * @warning max_value is exclusive (the range is [0, max_value)).
     * @pre The initial value is in the range [0, max_value).
     * @pre The pool_scaler is a large positive integer (the default is 8000000).
     * @warning Throws OutOfRange Exception if initial is in the incorrect range.
     */
     Alpex(int initial, int max_value, int pool_scaler);
     // Class destructor
     ~Alpex();

     // Copy constructor
     Alpex(const Alpex& other);
     // Copy-assignment operator
     Alpex& operator=(Alpex rhs);
 private:
     class Alpex_Algorithm;
     std::unique_ptr<Alpex_Algorithm> pimpl;
 };

#endif
```

## A.3 Python Wrapper Examples

**Listing A.2:** pybind11 python wrapper code for our Alpex predictor class.

```cpp
#include <pybind11/pybind11.h>
#include "alpex.h"

namespace py = pybind11;

PYBIND11_MODULE(alpex_py, m) {
    py::class_<Alpex>(m, "Alpex")
            .def(py::init())
            .def(py::init<int>())
            .def(py::init<int, int>())
            .def("reset", &Alpex::reset)
            .def("train", &Alpex::train)
            .def("predict", &Alpex::predict)
            .def("predict_without_training", &Alpex::predict_without_training)
            .def("add_value_to_history", &Alpex::add_value_to_history);
}
```

**Listing A.3:** pybind11 cmake build code for alpex python wrapper.

```cmake
cmake_minimum_required(VERSION 3.15)

project(alpex_py VERSION 1.0.1 DESCRIPTION "Alpex Python Client" LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 14)
SET(GCC_COVERAGE_LINK_FLAGS "-fPIC")
add_subdirectory(pybind11)
pybind11_add_module(alpex_py
        Alpex.cpp
        )

set_target_properties(alpex PROPERTIES POSITION_INDEPENDENT_CODE TRUE)
target_link_libraries(alpex_py PUBLIC alpex)
```

## A.4 Example Generator

**Listing A.4:** An example generator for the Henon Map.

```python
def generate_data_set(parameters):
    dataset_x = list()
    dataset_y = list()

    dataset_x.append(parameters["initial_x"])
    dataset_y.append(parameters["initial_y"])

    for i in range(1, len(parameters["t"])):
        x = 1 - parameters["a"] * dataset_x[i - 1] ** 2 + dataset_y[i - 1]
        y = parameters["b"] * dataset_x[i - 1]
        dataset_x.append(x)
        dataset_y.append(y)

    return dataset_x, dataset_y
```

## A.5 Example Instance of Using the Swagger Client to Communicate with the Server

**Listing A.5:** An example usage of the Swagger Client.

```
from __future__ import print_function
import swagger_client
from swagger_client.rest import ApiException

api_instance = swagger_client.BaseControllerApi()

def predict(value, maximum=254):
    try:
        return api_instance
            .predict_one_bounded_using_get(value=value, max=maximum)
    except ApiException as e:
        print("Exception when calling " +
            "BaseControllerApi->predict_one_bounded_using_get: %s\n" % e)
```

## A.6 Example of Adding an Experiment to the report generator

**Listing A.6:** An example of how to add an experiment to generate_report.py.

```
experiment = [
    "./time_series/..path_to_experiment_file_1..",
    "./time_series/..path_to_experiment_file_2..",
    "./time_series/..path_to_experiment_file_3..",
    ...
]

...

arguments = {
    ...,
    "experiment_name": experiment
}
```

Calling the function,

```
python ./generate_report.py experiment_name
```

will create a pdf './reports/experiment_name.pdf' in the format "experiment_1, experiment_2, experiment_3, ..."

# Appendix B

# Datasets for Experiments

## B.1 Mackey-Glass Experiment Results

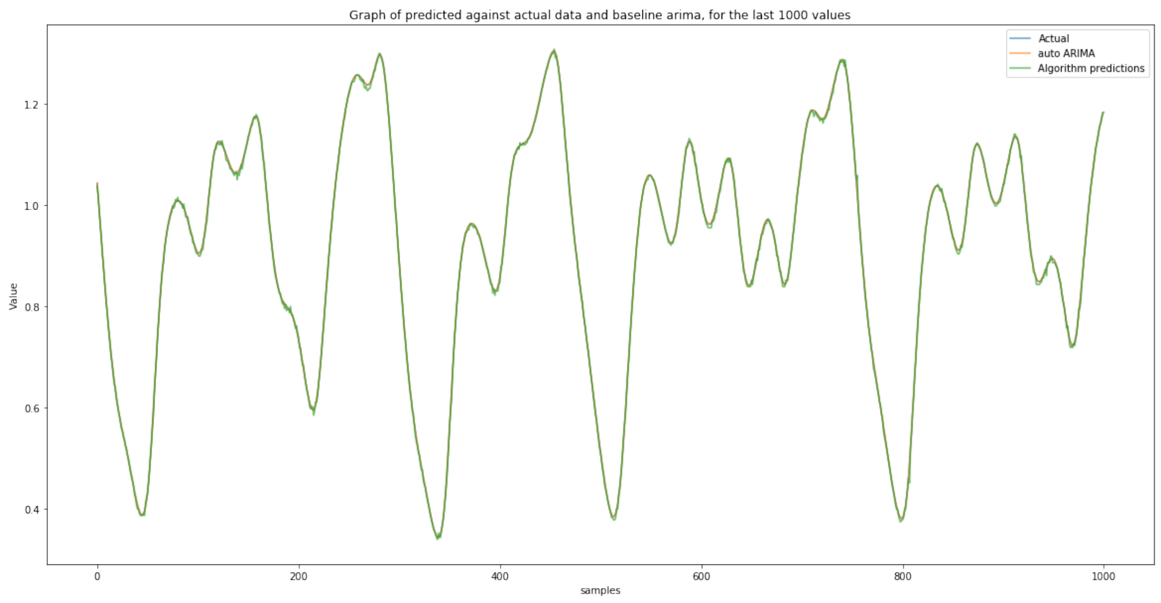


**Figure B.1:** Graph comparing Alpex's predictions and ARIMA's model against the original dataset for the Mackey Glass System.

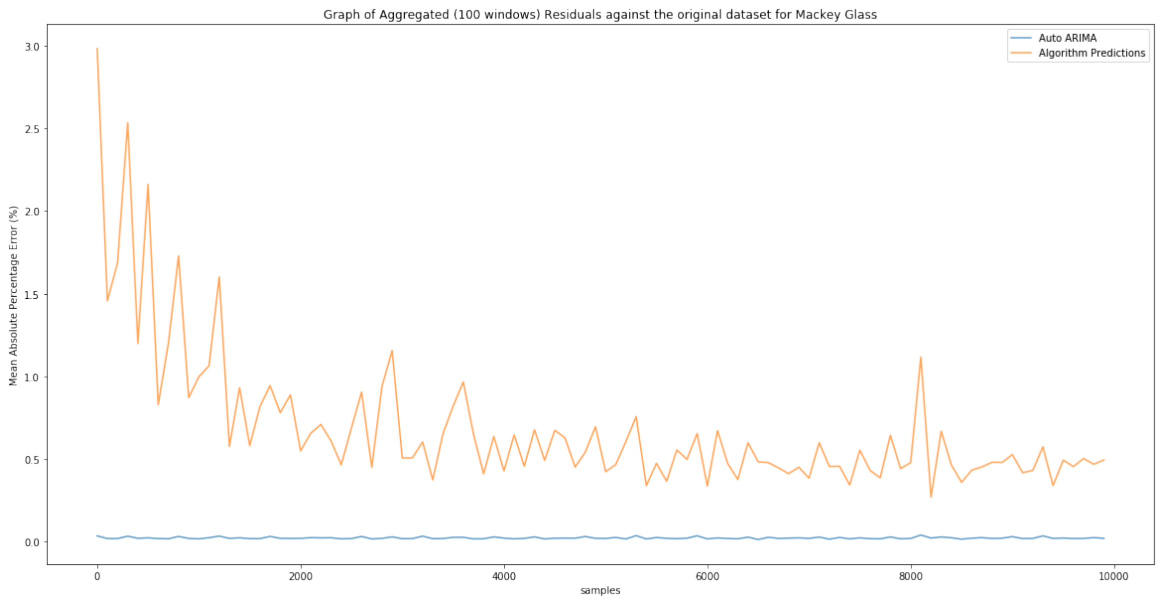


**Figure B.2:** Graph showing the residuals as a percentage of the values, taken as an average with 100 windows for the Mackey Glass System.

## B.2 Logistic Map Experiment Results

| Sigmas | MAE | MAPE (%) | CODE | $r^2$ |
|---|---|---|---|---|
| 0.00 | 0.011001 | 2.227099 | 2.271062 | 0.992374 |
| 0.10 | 0.017103 | 3.329940 | 2.515263 | 0.988114 |
| 0.25 | 0.032979 | 6.351746 | 3.663004 | 0.963898 |
| 0.50 | 0.060108 | 11.493736 | 4.371184 | 0.893001 |
| 0.75 | 0.089300 | 17.105186 | 7.521368 | 0.782894 |
| 1.00 | 0.117613 | 22.225216 | 10.134310 | 0.631381 |
| 2.00 | 0.222222 | 44.652586 | 24.102564 | -0.051092 |
| 4.00 | 0.382510 | 181.212732 | 39.511600 | -0.755665 |
| 7.00 | 0.592416 | 886.128677 | 47.594628 | -0.955527 |

**Table B.1:** Metric scores for predicting the Logistic Map Dataset.

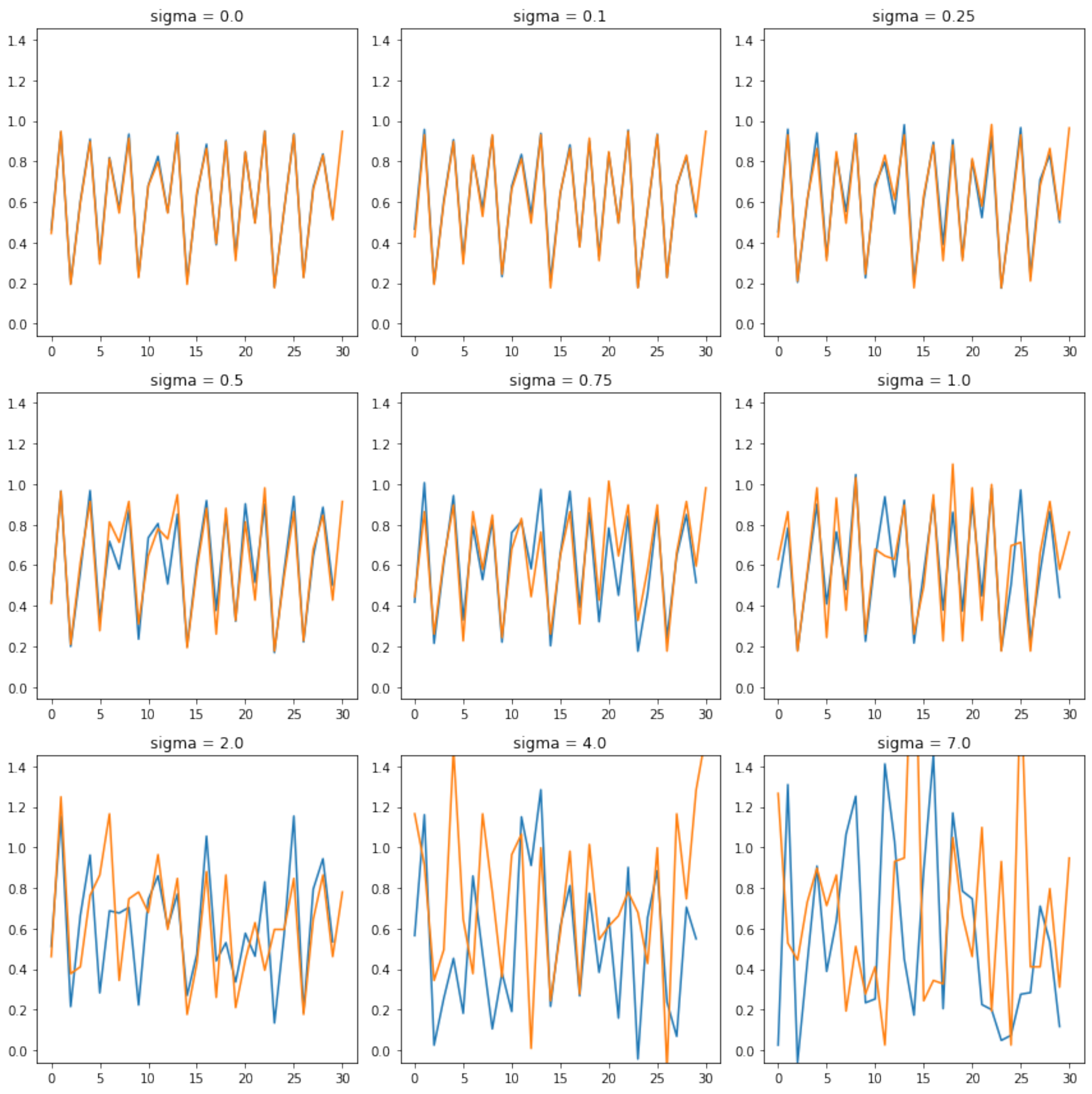


**Figure B.3:** A zoomed in view of the last 30 values of the predicted (orange) vs actual (blue) Logistic Map dataset at different sigma values.

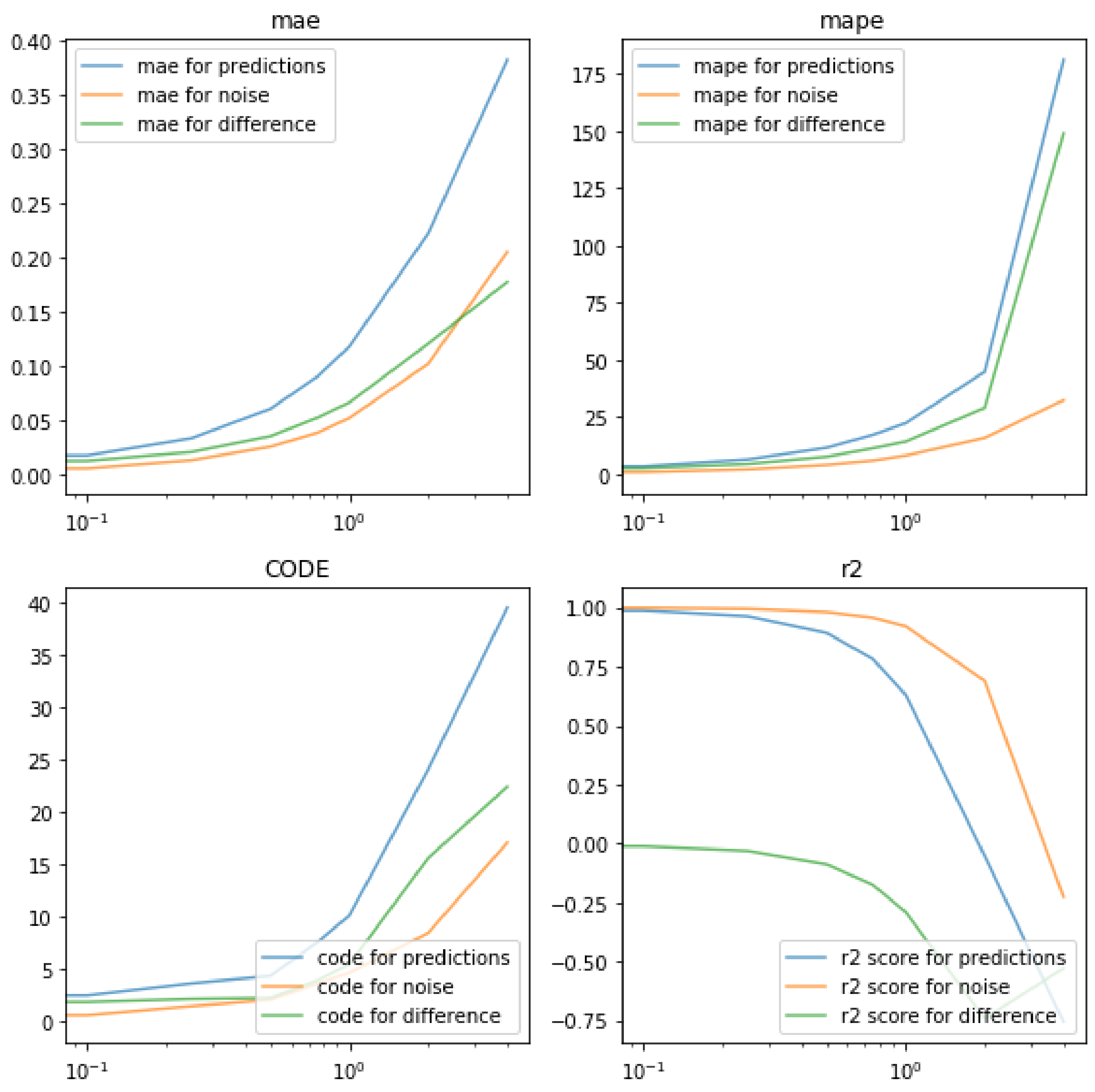


**Figure B.4:** Graphs comparing metric values against sigma for the Logistic Map Dataset as well as the metric values comparing the noisy dataset with the actual dataset.

## B.3 Metrics for other Prediction Methods

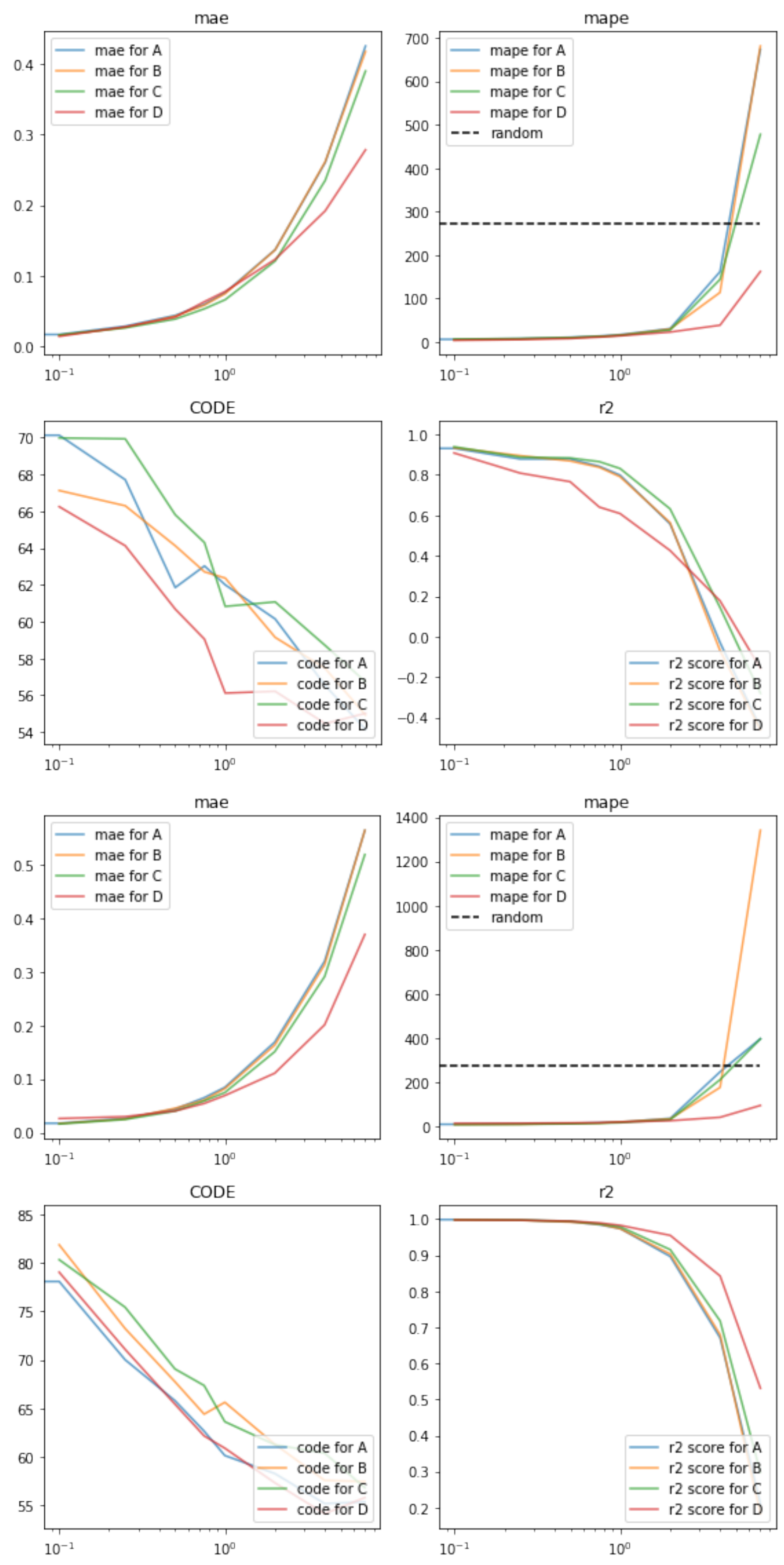


**Figure B.5:** Metrics for the Sawtooth(top) and cos(bottom) curve using different prediction methods,
where:

A = Normally trained predictions    B = Trained on clean data first
C = Trained on progressively noisy data    D = Trained only on clean data

## B.4 Noise Experiments

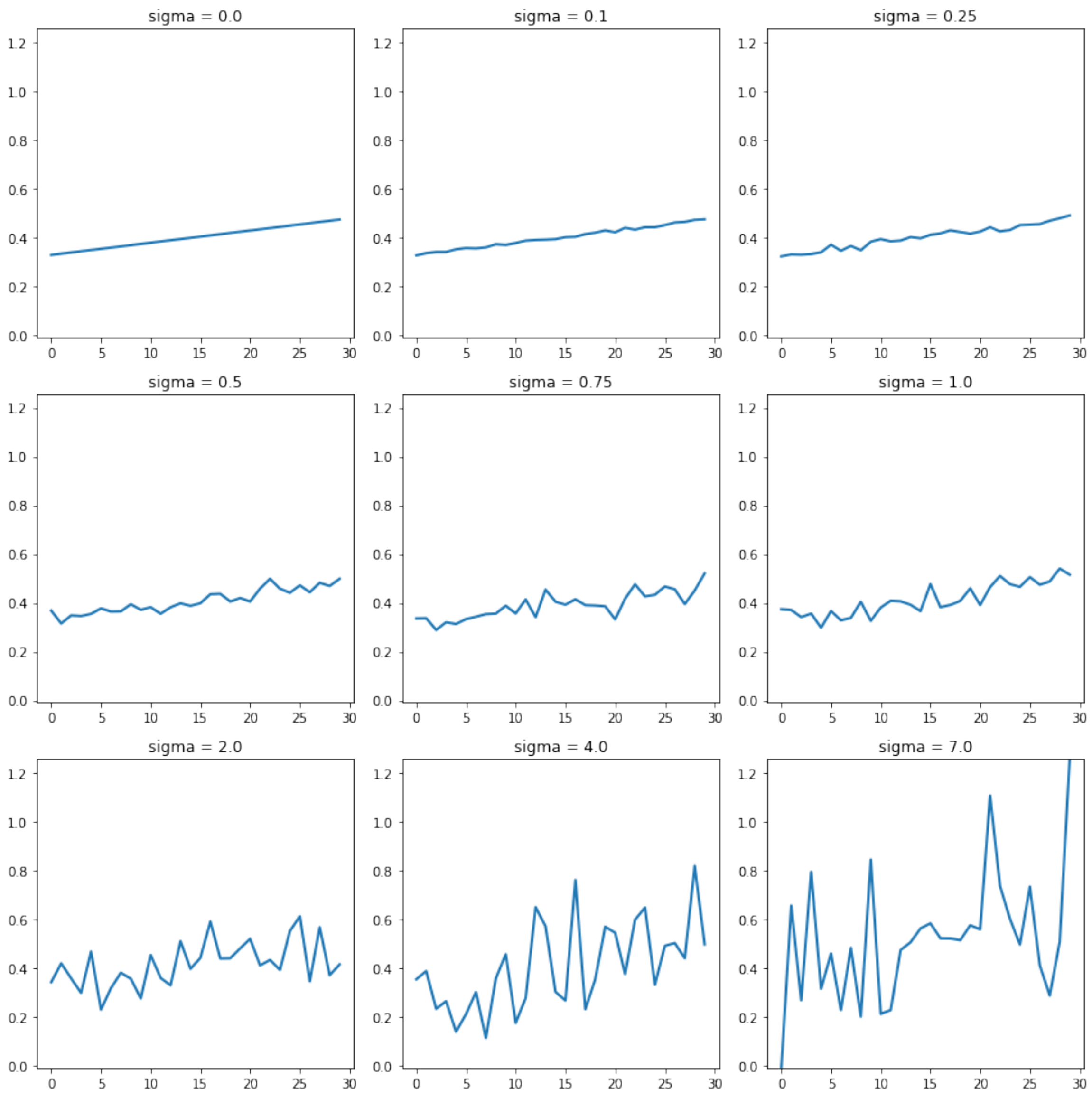


**Figure B.6:** A zoomed in view of the last 30 values of the sawtooth dataset at different sigma values.

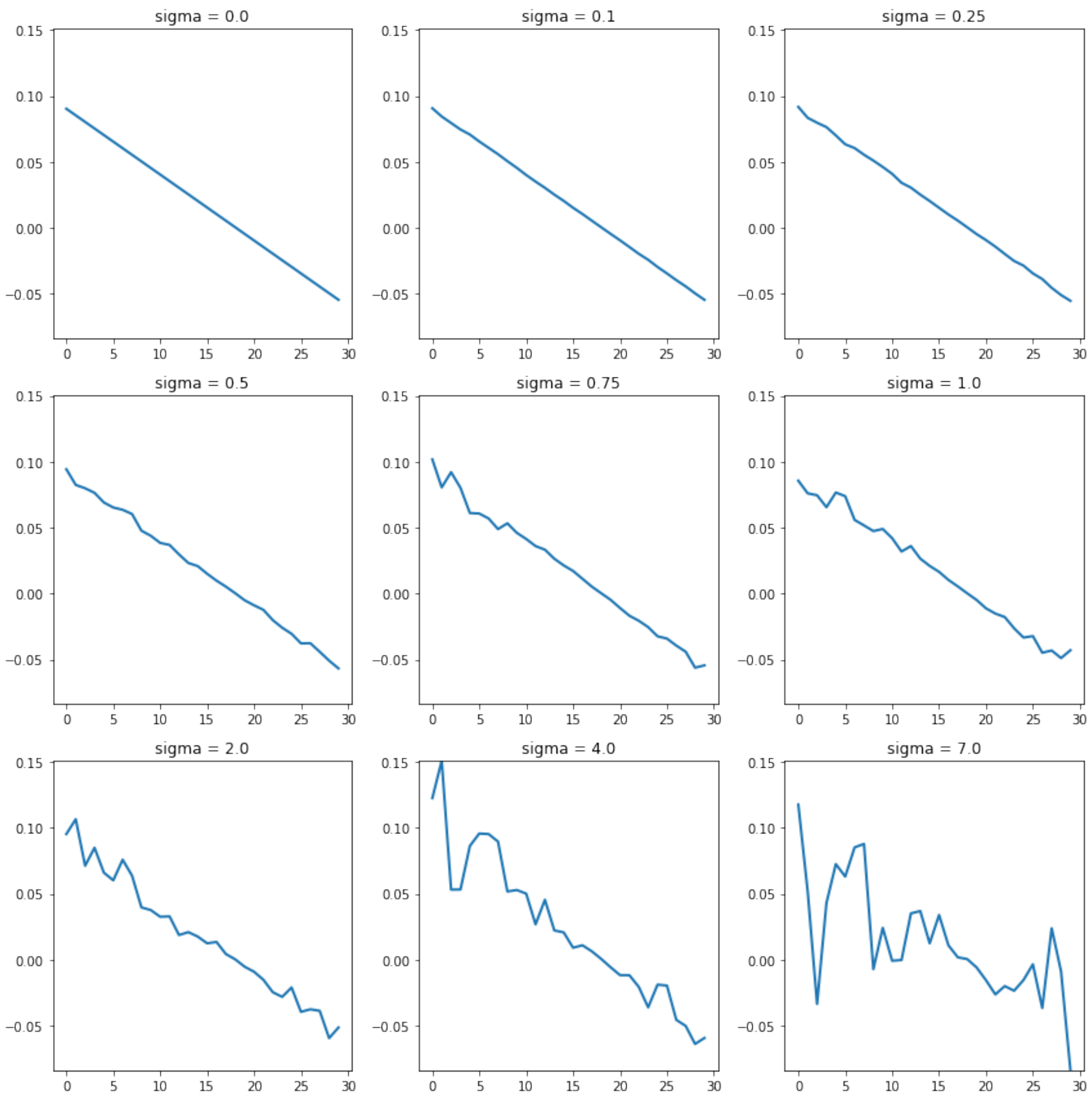


**Figure B.7:** A zoomed in view of the last 30 values of the cos dataset at different sigma values.

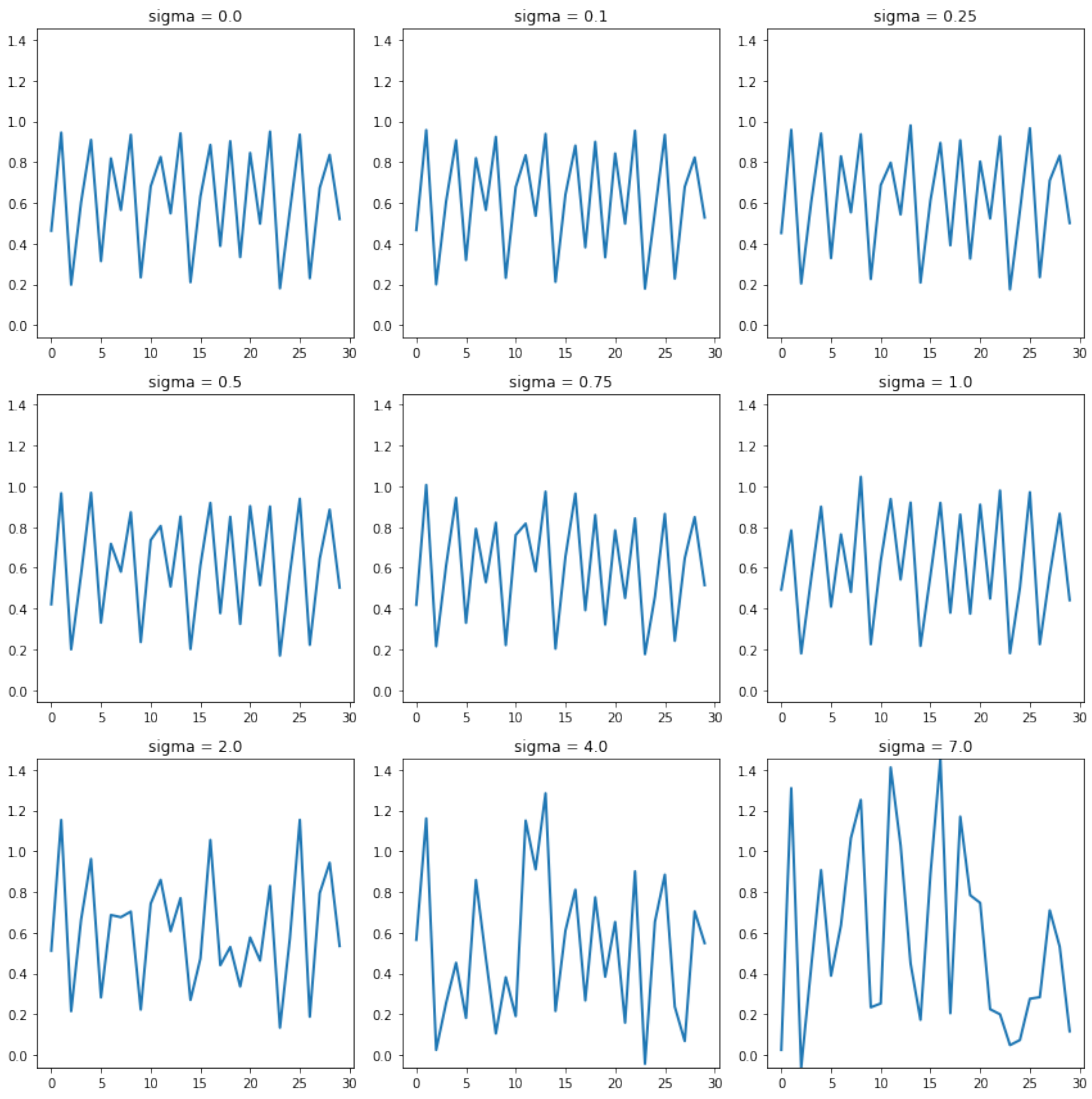


**Figure B.8:** A zoomed in view of the last 30 values of the Logistic Map dataset at different sigma values.

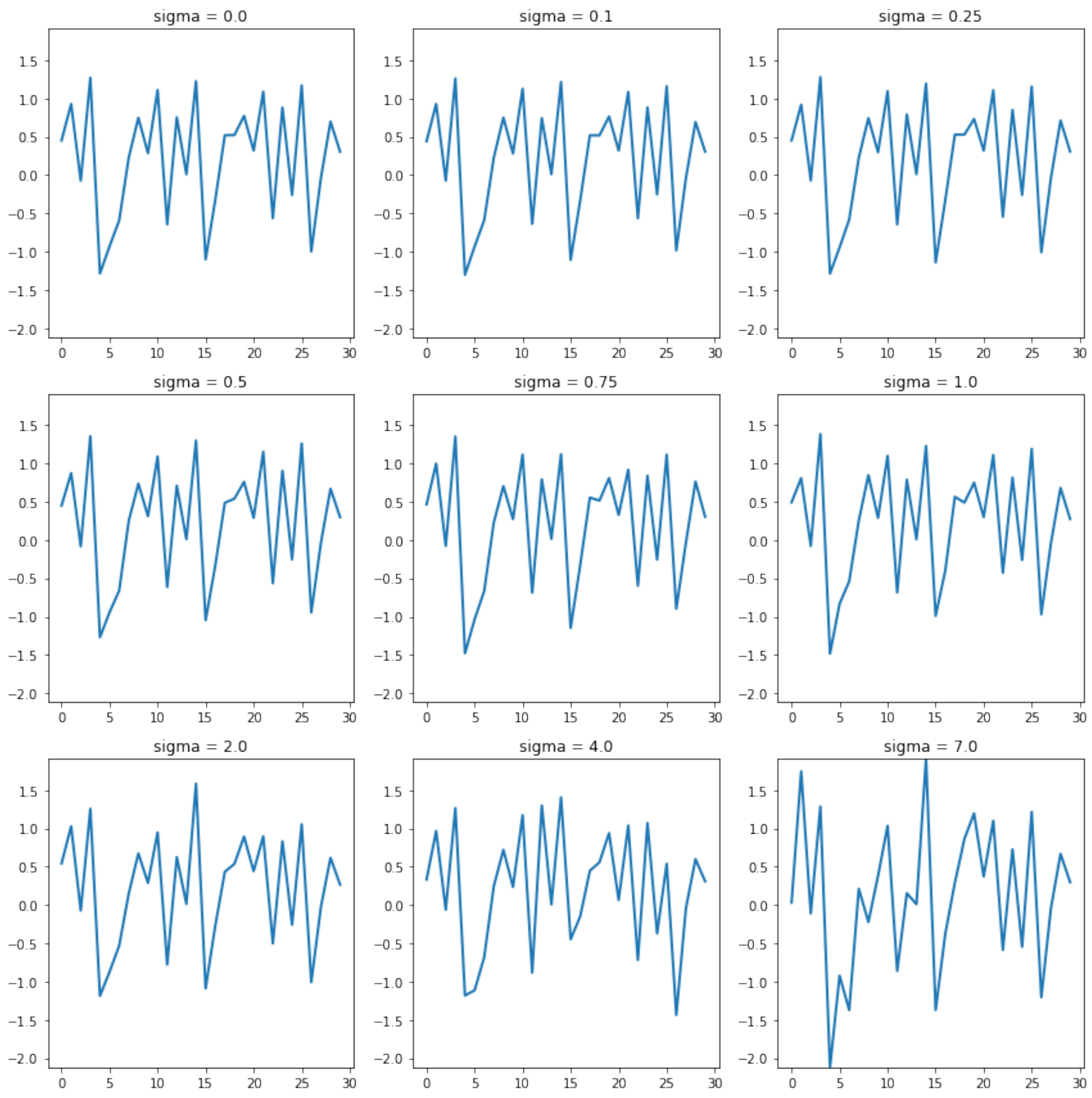


**Figure B.9:** A zoomed in view of the last 30 values of the Henon Map dataset at different sigma values.

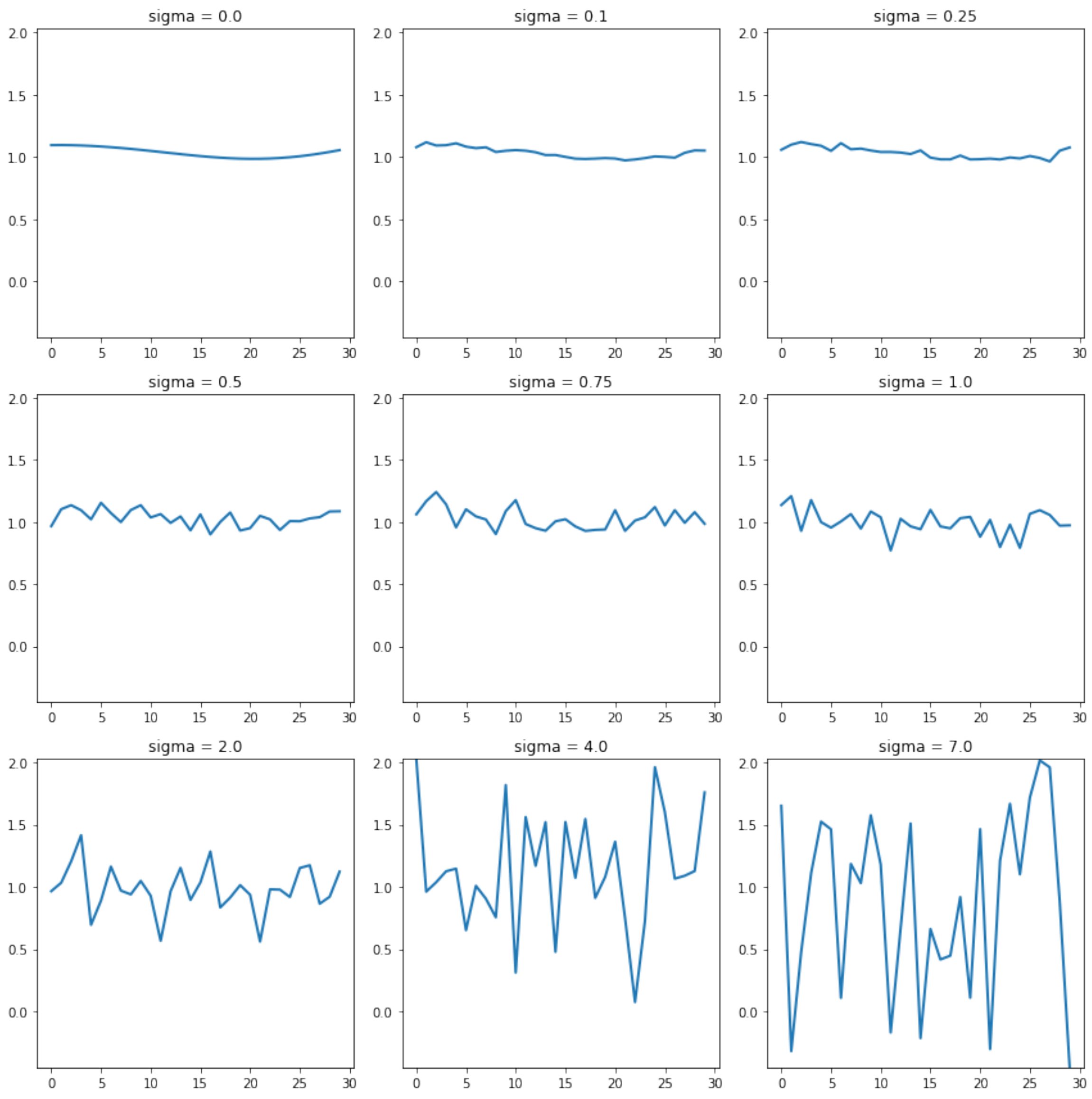


**Figure B.10:** A zoomed in view of the last 30 values of the Mackey Glass dataset at different sigma values.

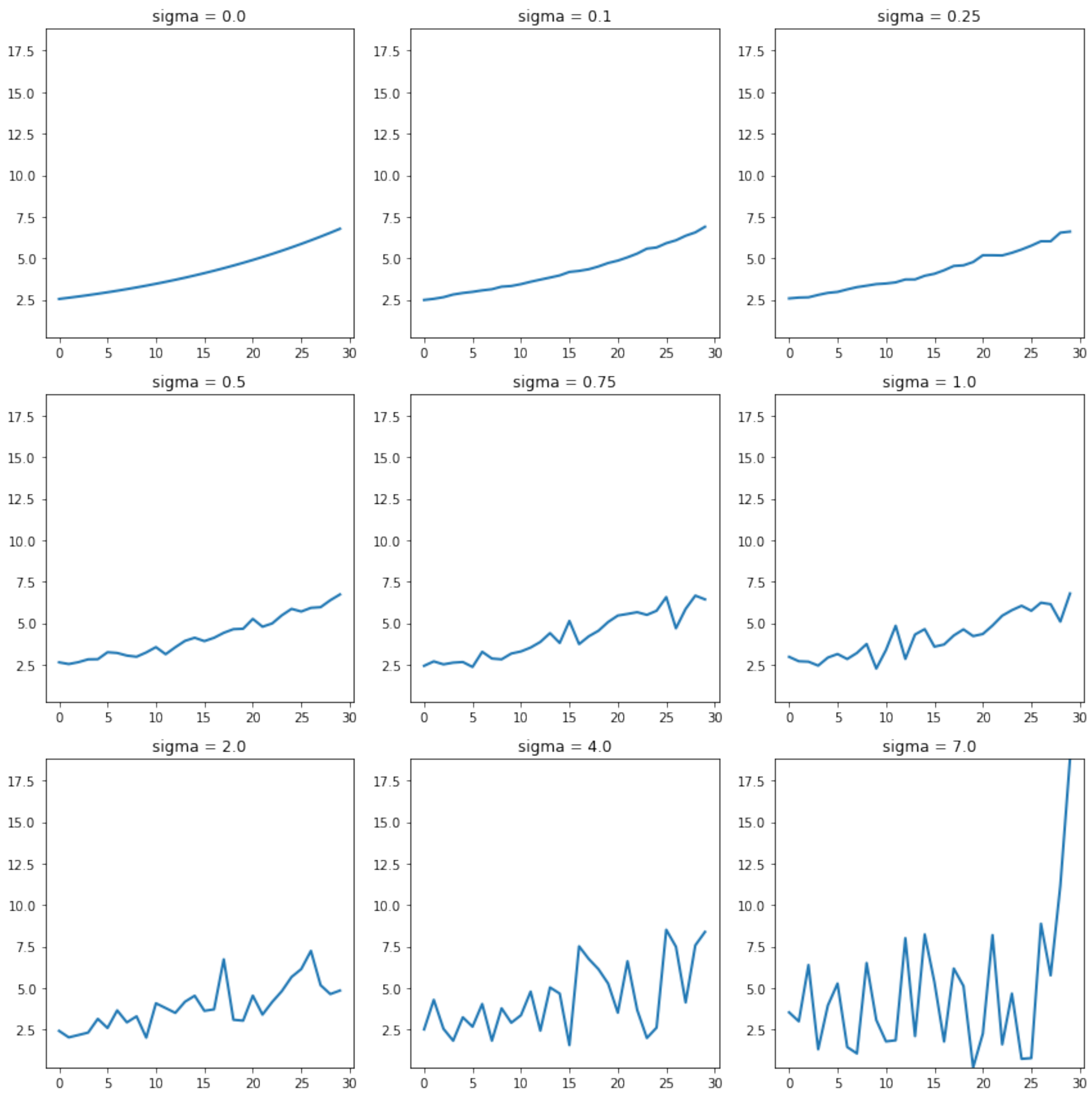


**Figure B.11:** A zoomed in view of the last 30 values of the Lorenz System dataset at different sigma values.

# Appendix C

# Real World Analysis

## C.1 Lagged Predictions

### C.1.1 Venice Hourly Water Level

### C.1.2 Belgium Offshore Generation

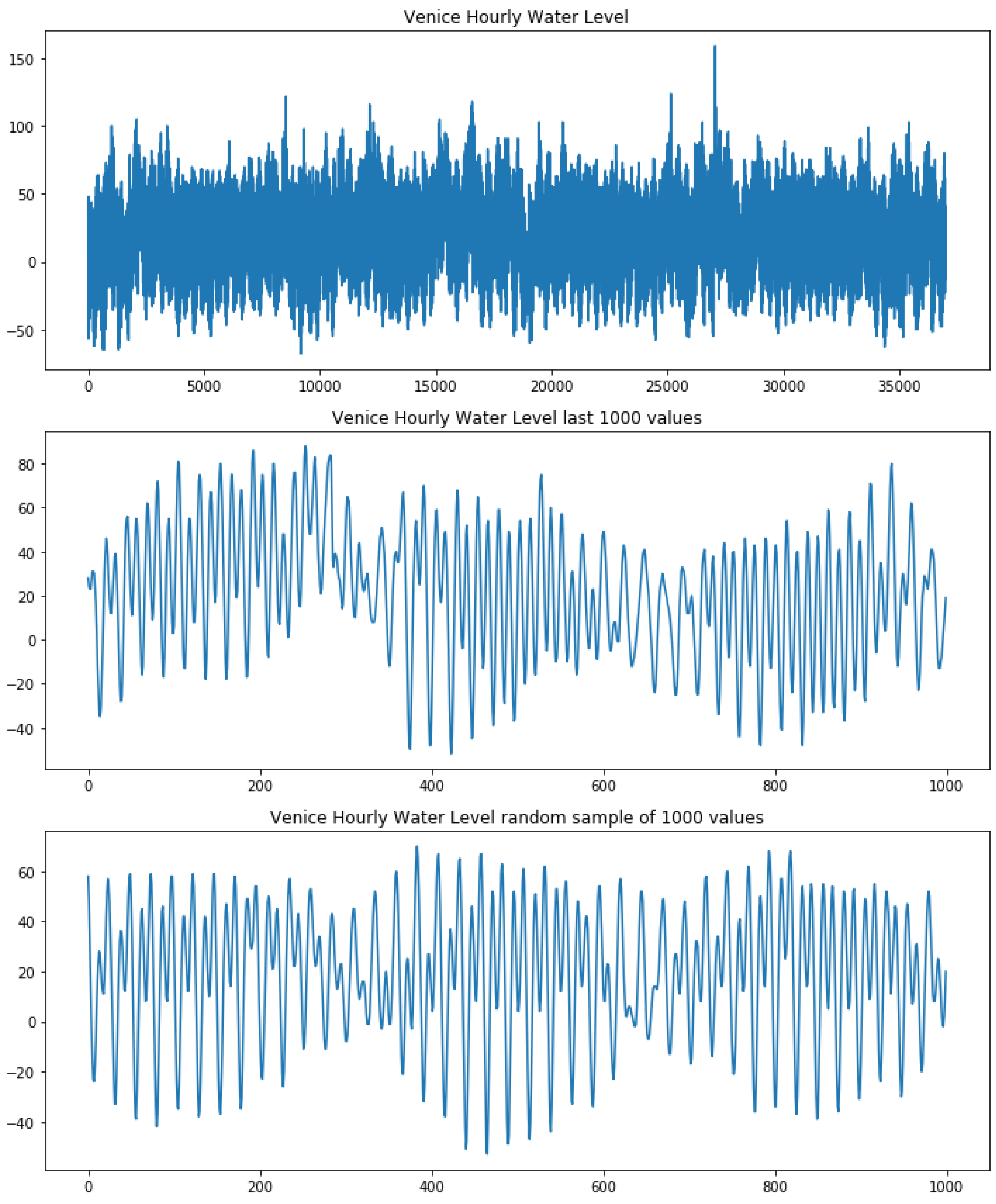


**Figure C.1:** Graphs showing Hourly Water Levels in Venice.

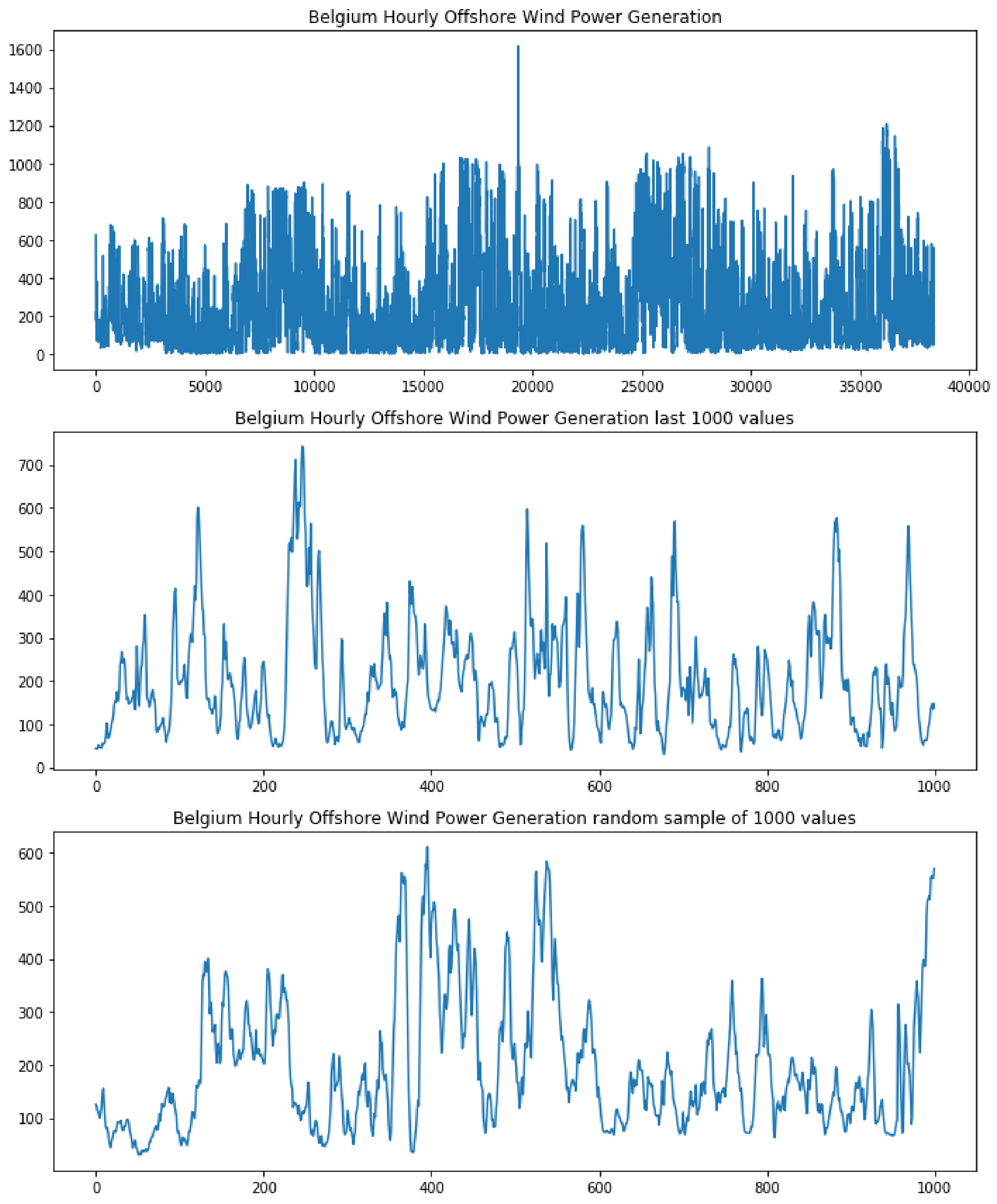
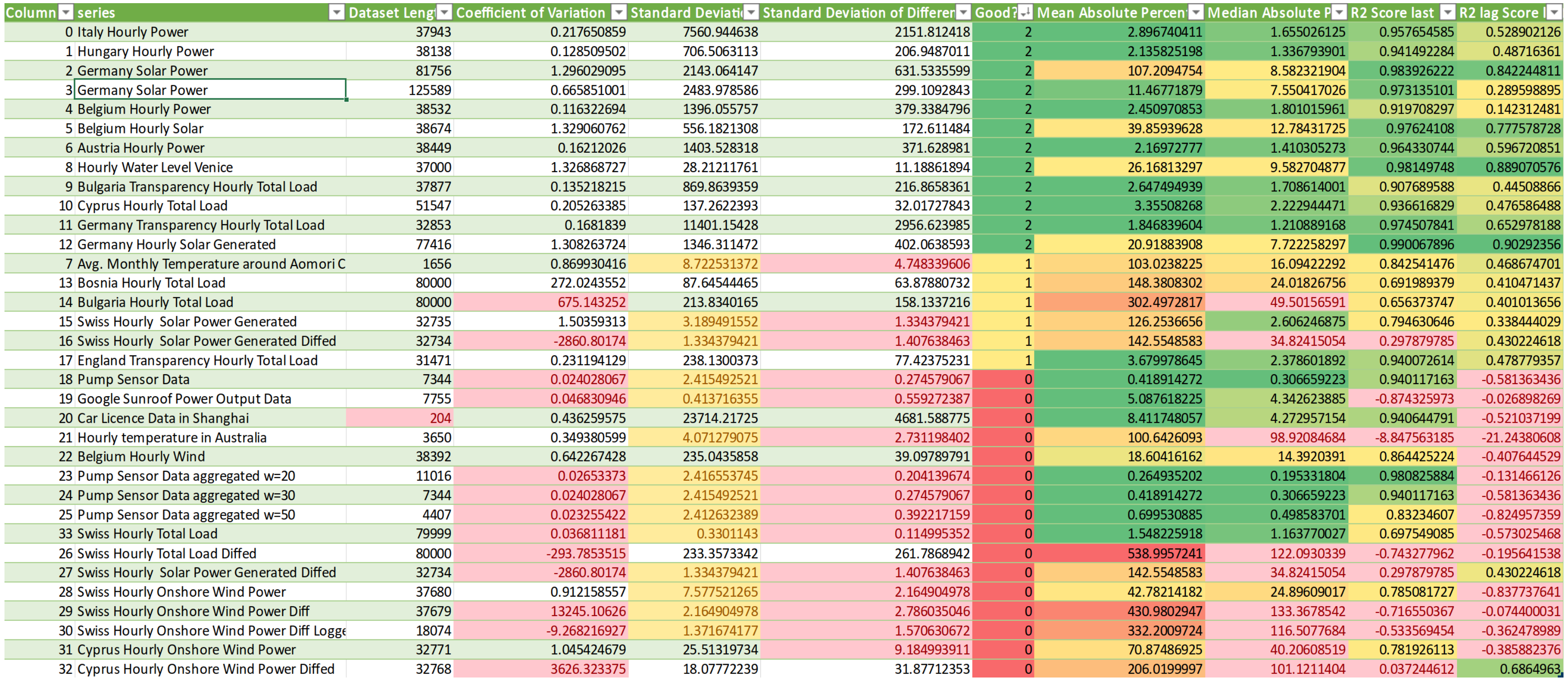


**Figure C.2:** Graphs showing Hourly Belgium Offshore Power Generation.

## C.2 Table of Time Series Properties



| Column | series | Dataset Leng | Coefficient of Variation | Standard Deviati | Standard Deviation of Differen | Good? | Mean Absolute Percen | Median Absolute P | R2 Score last | R2 lag Score |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Italy Hourly Power | 37943 | 0.217650859 | 7560.944638 | 2151.812418 | 2 | 2.896740411 | 1.655026125 | 0.957654585 | 0.528902126 |
| 1 | Hungary Hourly Power | 38138 | 0.128509502 | 706.5063113 | 206.9487011 | 2 | 2.135825198 | 1.336793901 | 0.941492284 | 0.48716361 |
| 2 | Germany Solar Power | 81756 | 1.296029095 | 2143.064147 | 631.5335599 | 2 | 107.2094754 | 8.582321904 | 0.983926222 | 0.842244811 |
| 3 | Germany Solar Power | 125589 | 0.665851001 | 2483.978586 | 299.1092843 | 2 | 11.46771879 | 7.550417026 | 0.973135101 | 0.289598895 |
| 4 | Belgium Hourly Power | 38532 | 0.116322694 | 1396.055757 | 379.3384796 | 2 | 2.450970853 | 1.801015961 | 0.919708297 | 0.142312481 |
| 5 | Belgium Hourly Solar | 38674 | 1.329060762 | 556.1821308 | 172.611484 | 2 | 39.85939628 | 12.78431725 | 0.97624108 | 0.777578728 |
| 6 | Austria Hourly Power | 38449 | 0.16212026 | 1403.528318 | 371.628981 | 2 | 2.16972777 | 1.410305273 | 0.964330744 | 0.596720851 |
| 8 | Hourly Water Level Venice | 37000 | 1.326868727 | 28.21211761 | 11.18861894 | 2 | 26.16813297 | 9.582704877 | 0.98149748 | 0.889070576 |
| 9 | Bulgaria Transparency Hourly Total Load | 37877 | 0.135218215 | 869.8639359 | 216.8658361 | 2 | 2.647494939 | 1.708614001 | 0.907689588 | 0.44508866 |
| 10 | Cyprus Hourly Total Load | 51547 | 0.205263385 | 137.2622393 | 32.01727843 | 2 | 3.35508268 | 2.222944471 | 0.936616829 | 0.476586488 |
| 11 | Germany Transparency Hourly Total Load | 32853 | 0.1681839 | 11401.15428 | 2956.623985 | 2 | 1.846839604 | 1.210889168 | 0.974507841 | 0.652978188 |
| 12 | Germany Hourly Solar Generated | 77416 | 1.308263724 | 1346.311472 | 402.0638593 | 2 | 20.91883908 | 7.722258297 | 0.990067896 | 0.90292356 |
| 7 | Avg. Monthly Temperature around Aomori C | 1656 | 0.869930416 | 8.722531372 | 4.748339606 | 1 | 103.0238225 | 16.09422292 | 0.842541476 | 0.468674701 |
| 13 | Bosnia Hourly Total Load | 80000 | 272.0243552 | 87.64544465 | 63.87880732 | 1 | 148.3808302 | 24.01826756 | 0.691989379 | 0.410471437 |
| 14 | Bulgaria Hourly Total Load | 80000 | 675.143252 | 213.8340165 | 158.1337216 | 1 | 302.4972817 | 49.50156591 | 0.656373747 | 0.401013656 |
| 15 | Swiss Hourly  Solar Power Generated | 32735 | 1.50359313 | 3.189491552 | 1.334379421 | 1 | 126.2536656 | 2.606246875 | 0.794630646 | 0.338444029 |
| 16 | Swiss Hourly  Solar Power Generated Diffed | 32734 | -2860.80174 | 1.334379421 | 1.407638463 | 1 | 142.5548583 | 34.82415054 | 0.297879785 | 0.430224618 |
| 17 | England Transparency Hourly Total Load | 31471 | 0.231194129 | 238.1300373 | 77.42375231 | 1 | 3.679978645 | 2.378601892 | 0.940072614 | 0.478779357 |
| 18 | Pump Sensor Data | 7344 | 0.024028067 | 2.415492521 | 0.274579067 | 0 | 0.418914272 | 0.306659223 | 0.940117163 | -0.581363436 |
| 19 | Google Sunroof Power Output Data | 7755 | 0.046830946 | 0.413716355 | 0.559272387 | 0 | 5.087618225 | 4.342623885 | -0.874325973 | -0.026898269 |
| 20 | Car Licence Data in Shanghai | 204 | 0.436259575 | 23714.21725 | 4681.588775 | 0 | 8.411748057 | 4.272957154 | 0.940644791 | -0.521037199 |
| 21 | Hourly temperature in Australia | 3650 | 0.349380599 | 4.071279075 | 2.731198402 | 0 | 100.6426093 | 98.92084684 | -8.847563185 | -21.24380608 |
| 22 | Belgium Hourly Wind | 38392 | 0.642267428 | 235.0435858 | 39.09789791 | 0 | 18.60416162 | 14.3920391 | 0.864425224 | -0.407644529 |
| 23 | Pump Sensor Data aggregated w=20 | 11016 | 0.02653373 | 2.416553745 | 0.204139674 | 0 | 0.264935202 | 0.195331804 | 0.980825884 | -0.131466126 |
| 24 | Pump Sensor Data aggregated w=30 | 7344 | 0.024028067 | 2.415492521 | 0.274579067 | 0 | 0.418914272 | 0.306659223 | 0.940117163 | -0.581363436 |
| 25 | Pump Sensor Data aggregated w=50 | 4407 | 0.023255422 | 2.412632389 | 0.392217159 | 0 | 0.699530885 | 0.498583701 | 0.83234607 | -0.824957359 |
| 33 | Swiss Hourly Total Load | 79999 | 0.036811181 | 0.3301143 | 0.114995352 | 0 | 1.548225918 | 1.163770027 | 0.697549085 | -0.573025468 |
| 26 | Swiss Hourly Total Load Diffed | 80000 | -293.7853515 | 233.3573342 | 261.7868942 | 0 | 538.9957241 | 122.0930339 | -0.743277962 | -0.195641538 |
| 27 | Swiss Hourly  Solar Power Generated Diffed | 32734 | -2860.80174 | 1.334379421 | 1.407638463 | 0 | 142.5548583 | 34.82415054 | 0.297879785 | 0.430224618 |
| 28 | Swiss Hourly Onshore Wind Power | 37680 | 0.912158557 | 7.577521265 | 2.164904978 | 0 | 42.78214182 | 24.89609017 | 0.785081727 | -0.837737641 |
| 29 | Swiss Hourly Onshore Wind Power Diff | 37679 | 13245.10626 | 2.164904978 | 2.786035046 | 0 | 430.9802947 | 133.3678542 | -0.716550367 | -0.074400031 |
| 30 | Swiss Hourly Onshore Wind Power Diff Logge | 18074 | -9.268216927 | 1.371674177 | 1.570630672 | 0 | 332.2009724 | 116.5077684 | -0.533569454 | -0.362478989 |
| 31 | Cyprus Hourly Onshore Wind Power | 32771 | 1.045424679 | 25.51319734 | 9.184993911 | 0 | 70.87486925 | 40.20608519 | 0.781926113 | -0.385882376 |
| 32 | Cyprus Hourly Onshore Wind Power Diffed | 32768 | 3626.323375 | 18.07772239 | 31.87712353 | 0 | 206.0199997 | 101.1211404 | 0.037244612 | 0.6864963 |

**Figure C.3:** A table of different time series properties with the corresponding metrics.

# Appendix D

# Future Work

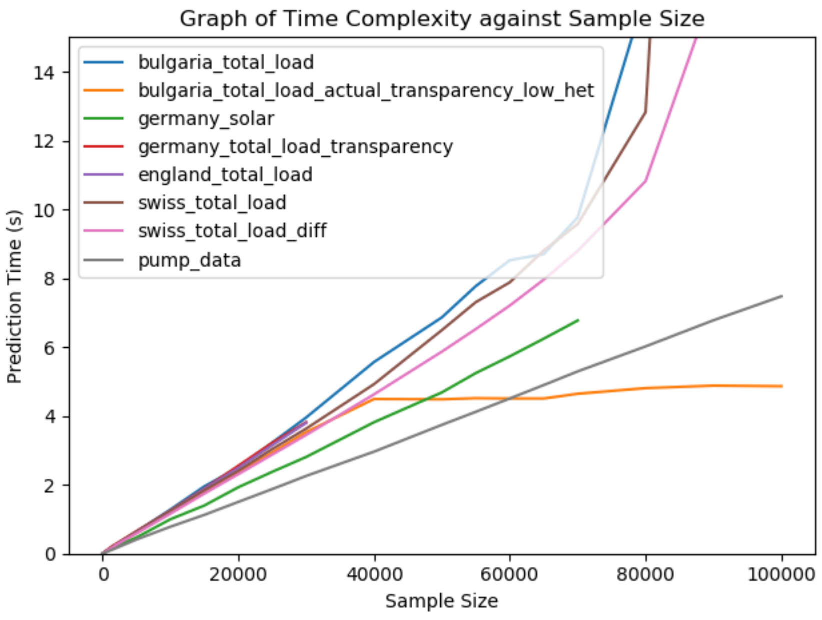


**Figure D.1:** A graph showing runtime against sample size for a number of experiments.

# Appendix E

# Case Study Experiments

## E.1 Case Study A: Southern Italy Power Load

# normal

June 8, 2020

## 1 Experiments into Power Load in Southern Italy

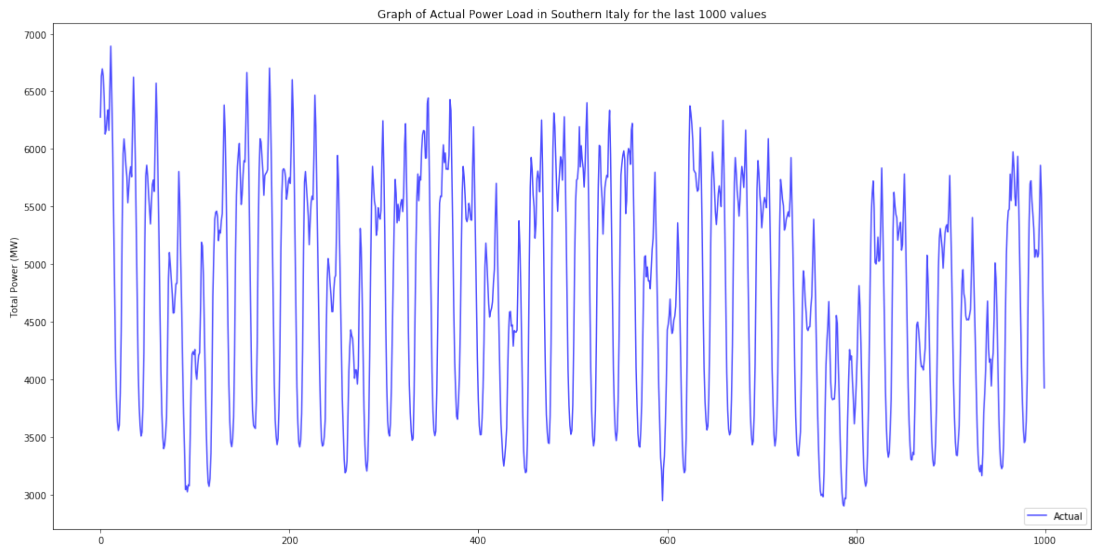
[13]:
```
import pickle
from metrics.metrics import Metrics, print_table
from time_series.graphing import plot_dataset, plot_comparative_datasets,␣
 ↪plot_acf_
from time_series.lag.lag import plot_change_in_metrics_as_grid,␣
 ↪plot_predictions_as_grid

import pandas as pd
from time_series.predictions import test_stationarity, de_trend_with_baseline,␣
 ↪log_dataset, \
    make_predictions_with_normalisation
from time_series.real_world_datasets.power_series.normal.normal import␣
 ↪generate_data_set, generate_arima
```

Load Datasets and Baseline Predictions

[14]:
```
dataset = generate_data_set("IT_CSUD_load_actual_entsoe_transparency")

look_back = 1000
```

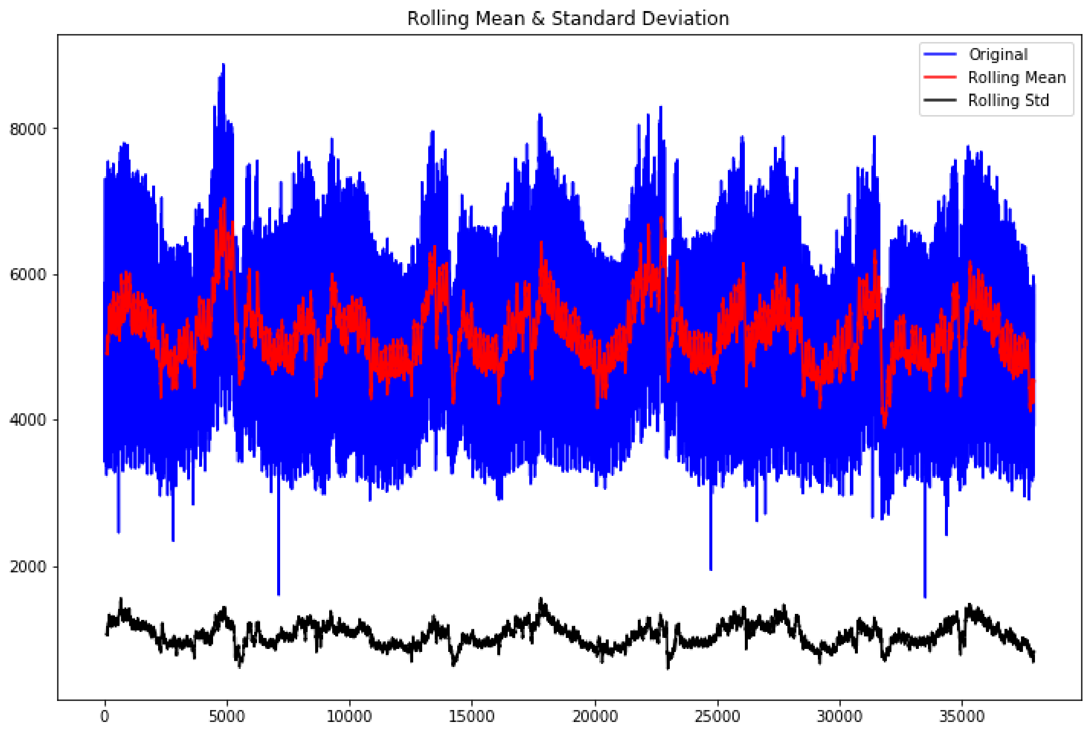
Plot the dataset

[15]:
```
plot_dataset(dataset[-look_back:], "Graph of Actual Power Load in Southern␣
 ↪Italy for the last " + str(look_back) + " values", y_axis_name="Total Power␣
 ↪(MW)")
```



Check for constant Mean and Heteroscedasticity

```
[16]: test_stationarity(pd.Series(dataset), window=100)
```



Set out if we want to remove trend (non constant mean), or log data (non constant variance).

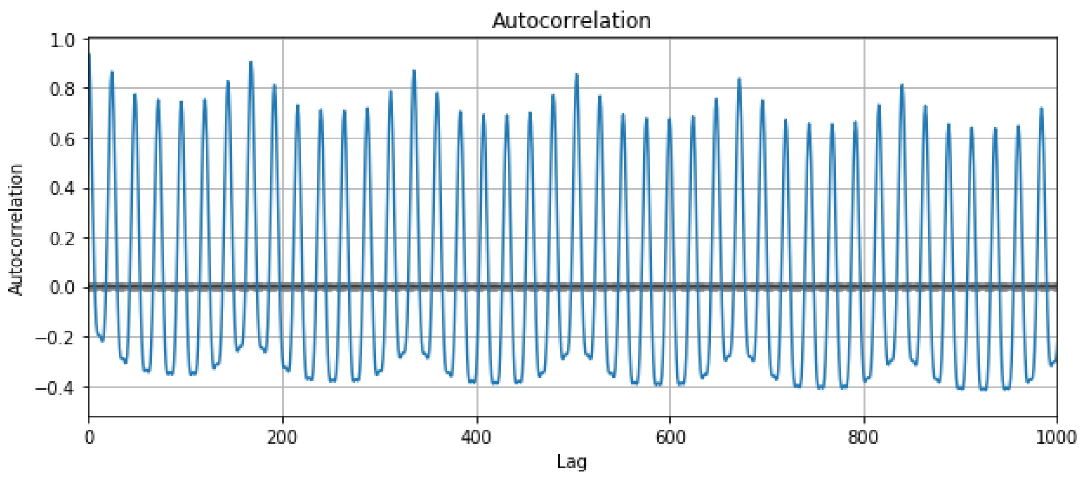
[17]:
```
remove_trend = True
log = True
```

Remove Trend through linear Regression

[18]:
```
if remove_trend:
    dataset, _ = de_trend_with_baseline(dataset, [])

if log:
    dataset = log_dataset(dataset)

generate_arima(dataset)
with open("out/arima.obj", "rb") as f:
    _, auto_arima = pickle.load(f)
```

Plot the Auto Correlation Function for the dataset

[19]:
```
plot_acf_(dataset)
```

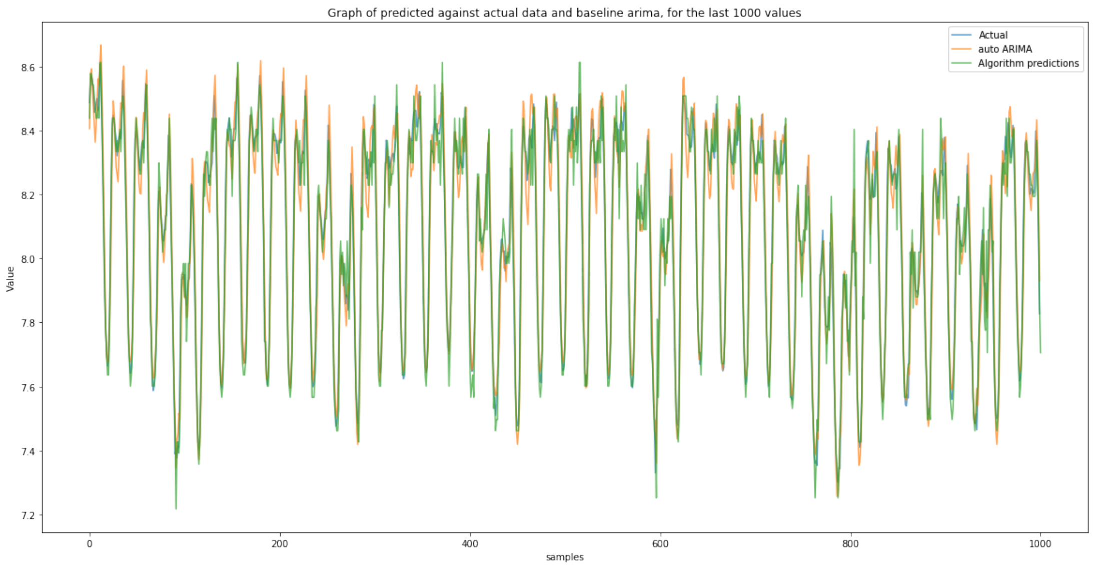


Make Predictions with algorithm

[20]:
```
predictions = make_predictions_with_normalisation(dataset)
```

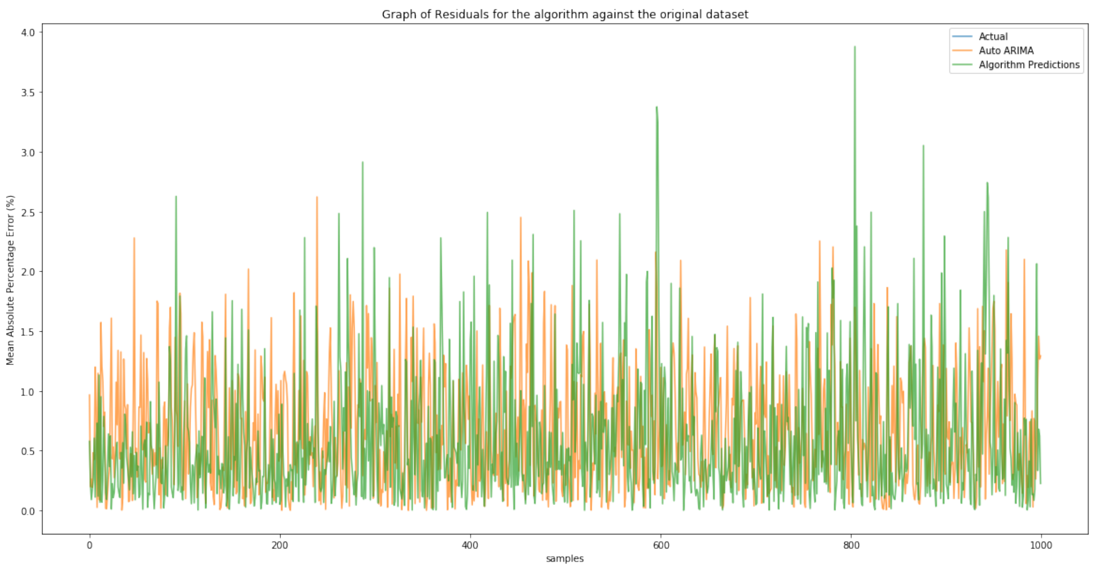
Plot the predictions against the dataset

[21]:
```
auto_arima = auto_arima.fittedvalues
plot_comparative_datasets(dataset[-look_back:],
                          [
                              ("auto ARIMA", auto_arima[-look_back:]),
                              ("Algorithm predictions", predictions[-(look_back+1):
 ↪])
                          ],
                        "Graph of predicted against actual data and baseline␣
↪arima, for the last " + str(look_back) + " values")
```



Plot the residual of the predictions against the dataset

```
[22]: plot_comparative_datasets([],
                                [
                                    ("Auto ARIMA", [100.0 * (abs(a - b)/a) for a, b␣
↪in zip(auto_arima[-look_back:], dataset[-look_back:])]),
                                    ("Algorithm Predictions", [100.0 * (abs(a - b)/a)␣
↪for a, b in zip(predictions[-(look_back + 1):], dataset[-look_back:])])
                                ],
                                "Graph of Residuals for the algorithm against the␣
↪original dataset",
                                y_axis_name="Mean Absolute Percentage Error (%)")
```



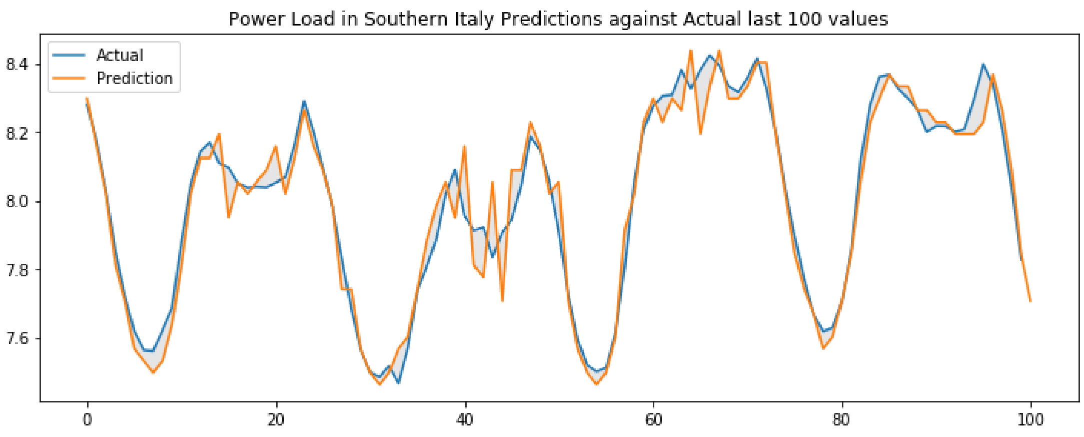
Calculate metrics for the last n values.

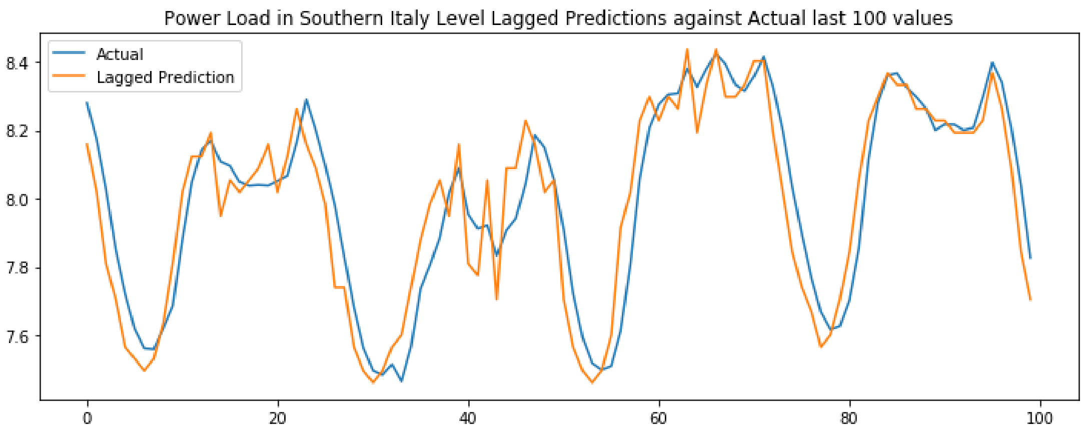
```
[23]: metrics_new_algorithm = Metrics(dataset[-look_back:], predictions[-(look_back +␣
       ↪1):-1])

      metrics_auto_arima = Metrics(dataset[-look_back:], auto_arima[-look_back:])
      print_table([metrics_auto_arima, metrics_new_algorithm], "Algorithm", ["AA",␣
       ↪"Alpex"], ["mape", "mae", "r2", "r2_lag", "med_ape"])
```

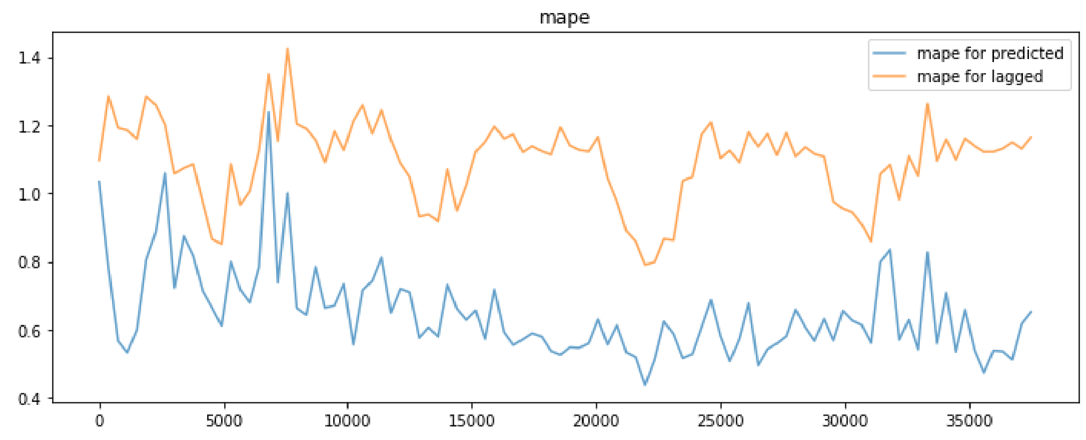
```
  Algorithm       mae      mape        r2    r2_lag   med_ape
0        AA  0.056215  0.693589  0.949401  0.581955  0.603234
1     Alpex  0.048007  0.594459  0.954511  0.624179  0.422753
```

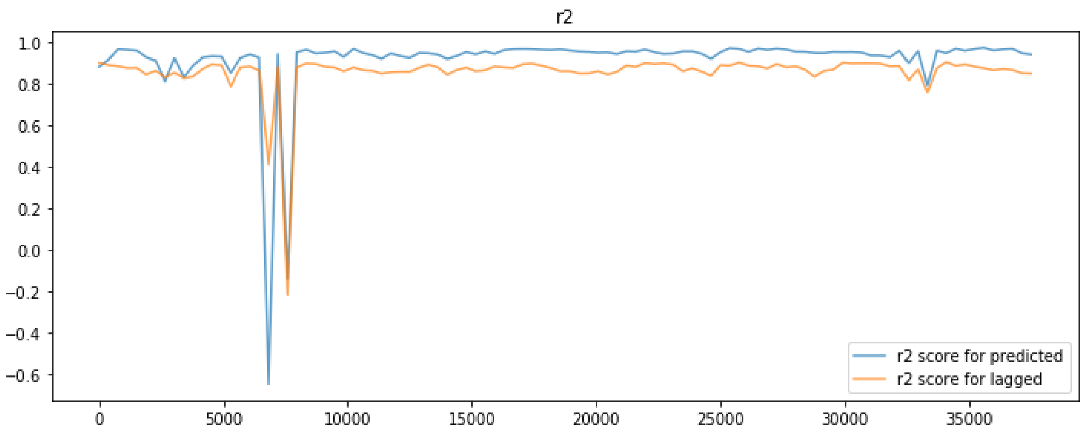
Test for lag

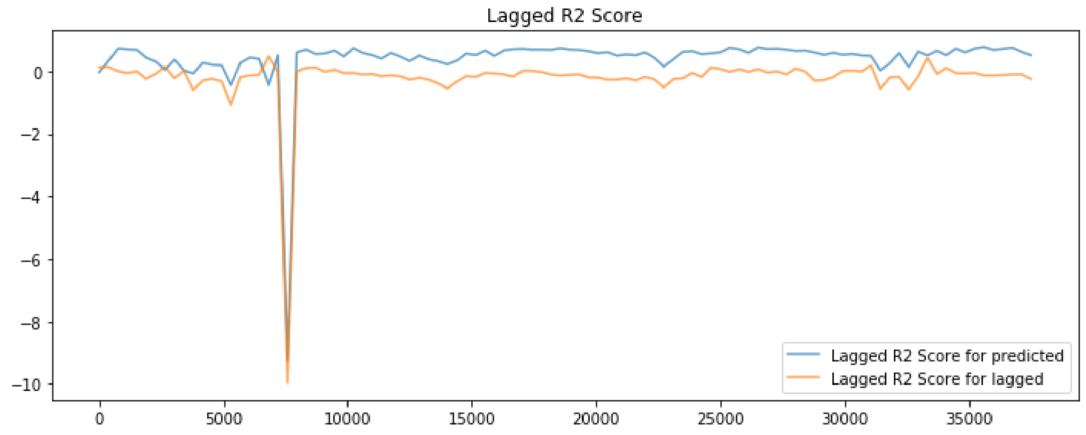
```
[24]: plot_predictions_as_grid(dataset, predictions, name='Power Load in Southern␣
       ↪Italy')

      plot_change_in_metrics_as_grid(dataset, predictions)
```


Power Load in Southern Italy Predictions against Actual last 100 values
Actual
Prediction


Power Load in Southern Italy Level Lagged Predictions against Actual last 100 values
Actual
Lagged Prediction


mape
mape for predicted
mape for lagged
1.4
1.2
1.0
0.8
0.6
0.4
0
5000
10000
15000
20000
25000
30000
35000


r2
r2 score for predicted
r2 score for lagged
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
5000
10000
15000
20000
25000
30000
35000


Lagged R2 Score
Lagged R2 Score for predicted
Lagged R2 Score for lagged
0
−2
−4
−6
−8
−10
0
5000
10000
15000
20000
25000
30000
35000