IL2223 - Embedded Intelligence
Project Report

Joshua Sadiq

Contents

Task 1 - Time-series prediction with neural networks **2**
Task 1.1 - Prediction with MLP, RNN, LSTM using synthetic series 2
Task 1.2 - Predict white noise, random walk, an ARMA process using neural networks 8
Task 1.3 - Comparison with ARIMA-based modeling and prediction 11

Task 2 - Decomposition-based anomaly detection **15**
Task 2.1 - Anomaly identification in global land temperature changes 15

Task 3 - Prediction-based anomaly detection **17**
Task 3.1 - Anomaly detection for uni-variate series with ARIMA . . 17
Task 3.2 - Anomaly detection with LSTM in ECG signals 20

Task 4 - Clustering-based Anomaly detection **23**
Task 4.1 - Anomaly detection in a bivariate series 23

Task 5 - Summary **25**

Introduction

Anomaly detection is an important tool which can be applied to data series. Used in fields of economics, medicine, industry, and more, understanding how and where to use it is key to pre-processing data and removing anomalous data. Identifying anomalies is not a simple task. There is a plethora of methods that are useful, depending on the data set.
In the final project of the course *IL2233 - Embedded Intelligence*, offered at KTH, the students are to understand how anomalies can be detected and removed using neural networks, ARIMA processes, and clustering.
Task 1 aims to introduce time-series prediction using neural networks and ARIMA-based modeling. *Task 2* will introduce the first anomaly-detection method, namely the decomposition-based method. *Task 3* handles prediction-based anomaly-detection. Finally, *task 4* teaches clustering-based anomaly detection. The summary, found under *task 5* ties the bag together.

Task 1 - Time-series prediction with neural networks

Task 1 aims to introduce simple time-series prediction using neural networks. It also aims to utilize neural networks to predict a variety of random time series. Finally, a comparison between the neural networks and ARIMA-based modelling will be discussed.

Task 1.1 - Prediction with MLP, RNN, LSTM using synthetic series

Task 1.1 is divided into four (4) subtasks. In the first subtask, an equal-difference series $\in [0, 1)$ with 200 steps was generated. Following that, a one-step prediction MLP with a single hidden layer, RELU-activation function, and 4-10-1 nodes (input, hidden, and output layers) was designed. The data was transformed into a supervised learning data set, with a training-to-test ratio of 8:2. For the loss function, MSE was chosen as this was a regression problem, and the famous Adam optimizer fit the solution well. Once the model was trained, a prediction test series was generated and compared to the test data set. An MSE-value around 0.000700 was found, and the results can be seen in fig. 1 below. Overall, the neural network achieved a favourable prediction.

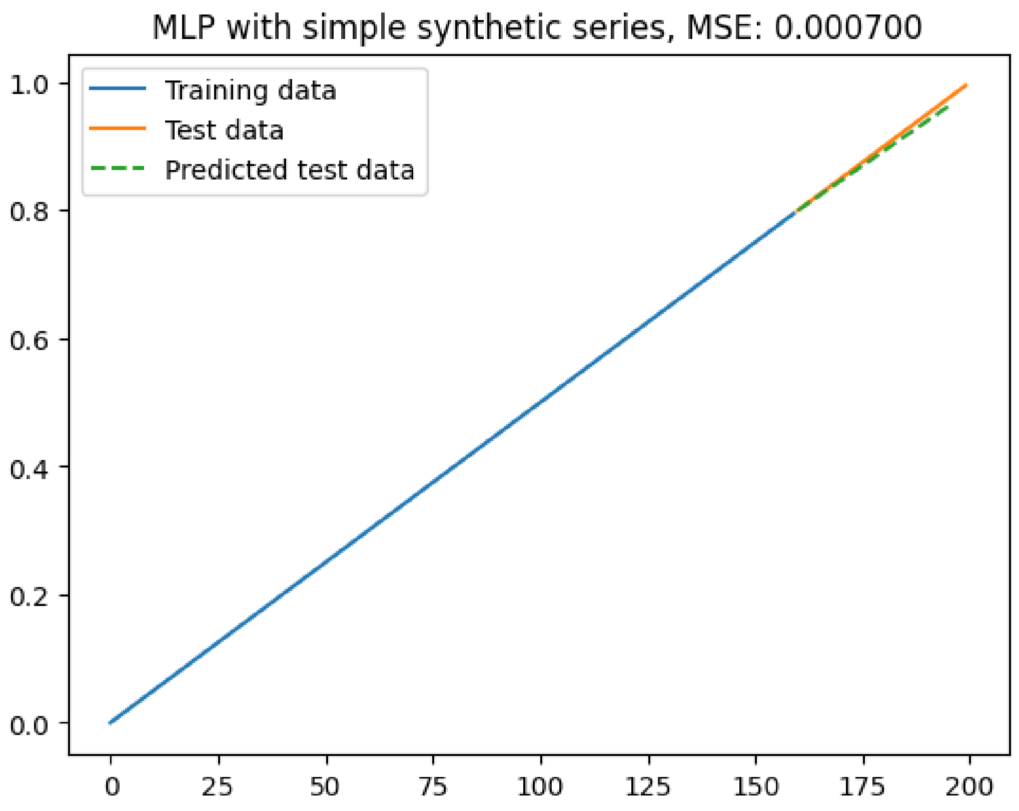

Figure 1: MLP predicting simple synthetic series

For the second subtask, the data set requirement added an additional white noise series added unto the synthetic series. Following that, the procedure to predict the data set remained the same, namely using an MLP. Recycling the same model from previous subtask, the results for the prediction can be seen in fig. 2 below. The neural network managed to predict the trend of the data set quite favourably here as well, resulting in an MSE value of 0.007. Although the MSE was a degree smaller than for the simple synthetic series, predicting random noise is not an easy task, and the achieved value is favourable in this case.

The third subtask changed method and data set. Here, a deterministic series sampled from a sinusoid wave was required. Period time was 20 seconds and a sample rate at 100 Hz. Furthermore, both a Recurrent Neural Network (RNN) and a Long Short-Term Memory (LSTM) models were to be used for the prediction. Finally, a two-step output vector was to be used.

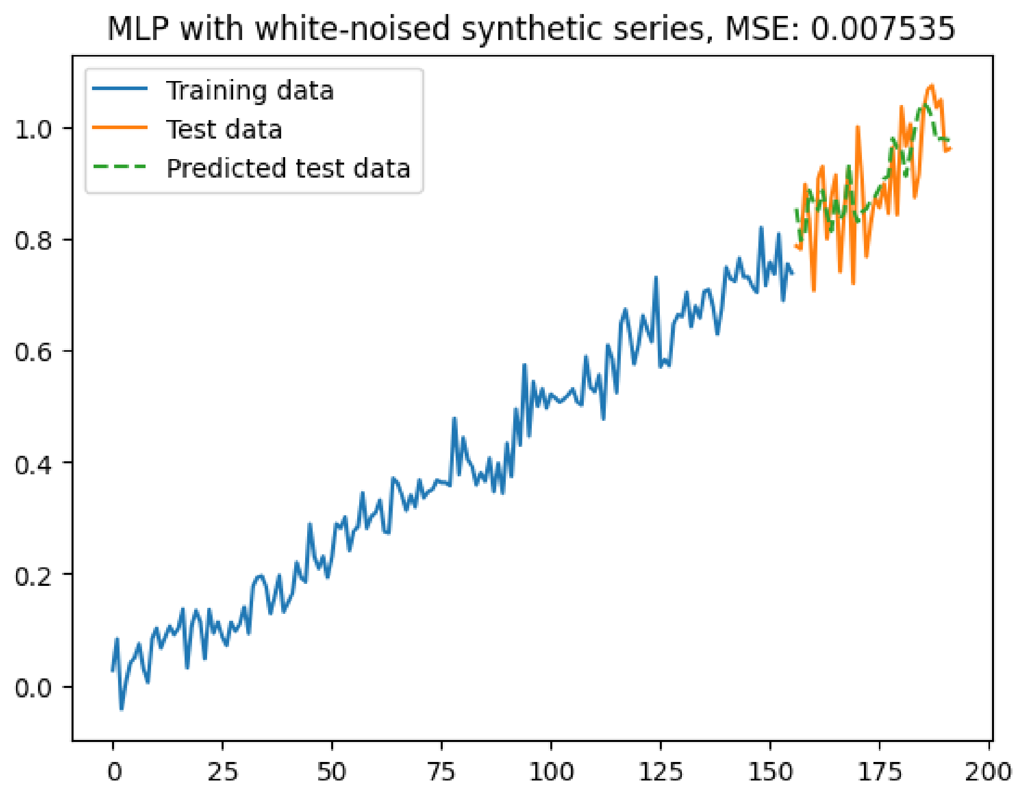

Figure 2: MLP predicting simple synthetic series with white noise

For the RNN, a simple five-unit dense layer was used. As the sinusoid data set pendles between positive and negative values, a RELU-activation function would not work optimally[1]. The same limitation would arise with the Sigmoid function. For that reason, Tanh was chosen as the activation function. MSE and Adam were once again chosen as the loss function and optimizer, respectively. The final predicted test data achieved an MSE value of 0.0013, which is favourable. In fig. 3 visualizes the predicted test data. Fine tuning the model could most likely achieve a visually near-perfect prediction.

An LSTM was also modeled and trained to predict the same data set. With a similar single-dense-layer, five-unit composition as the RNN the second model achieved an MSE value of 0.00093 - a degree better than the previous model. As can be seen in fig. 4, the prediction fits much better to the data set. This could indicate that the LSTM is a better-suited model for this form

[1]The Rectified Linear Unit activation function can only work on positive values.

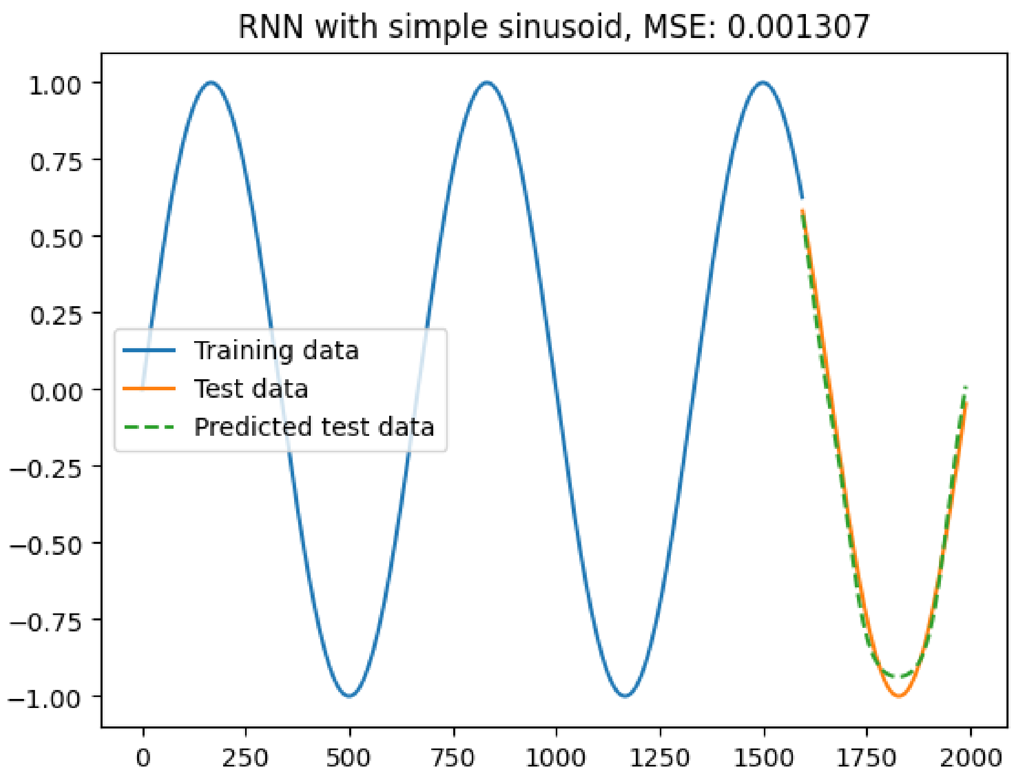

Figure 3: RNN predicting simple sinusoid series

of regression.

For the final subtask, similarly to subtask two a random white noise was added on top of the previous data set. Here, as well, an RNN and an LSTM models were to predict the series. The models were not changed to fit the new series, but the MSE values were lower than for subtask three. This is to be expected, as white noise introduces a more-challenging data set to predict. Results can be see in figs. 5 and 6, respectively.

Discussion

When designing each neural network, simplicity has been the key. Keeping number of layers and nodes as low as possible allows faster computation times and reduced complexity, all the while maintaining good metrics. For that reason, no network has more than one layer. The amount of hidden nodes also remains low, between five and ten. There was no specific reason for these particular values, and further experimentation could most likely find

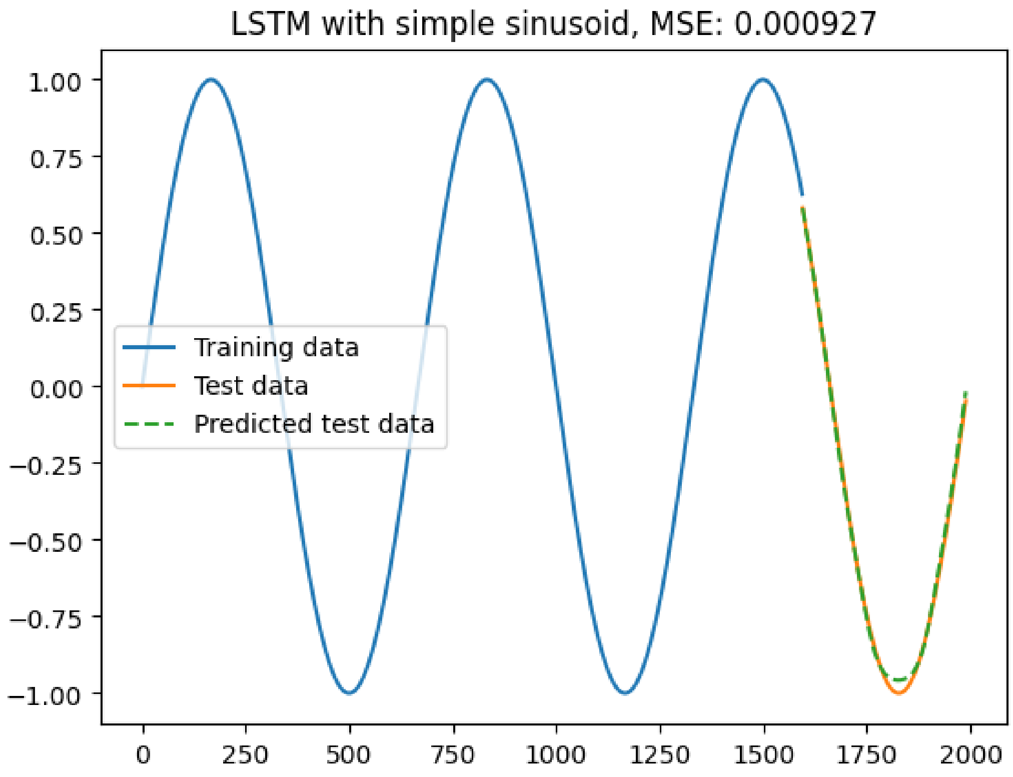

Figure 4: LSTM predicting simple sinusoid series

a more-optimal amount. For the first two subtasks, the RELU was a given choice due to its simplicity. When RELU could no longer achieve favourable results (for subtasks three and four), the change for Tanh was motivated.

For the hyperparameters, a learning of 0.01 was sufficient for the first subtask, and 0.001 for the second (due to its slightly more complex nature). A smaller learning rate results in longer computation times, but also incorrect fitting. Between 10 and 100 epochs were enough to train the model.

As demonstrated, the neural networks fit to the data set quite well and achieved favourable values for the accuracy metrics. Optimizing the models and achieving even lower metrics is possible.

As seen in subtask three, the LSTM model seemed to achieve much better results when predicting the simple sinusoid series. For subtask four, there was no comparable difference in the accuracy metric between the RNN and the LSTM models. This leads me to believe that white noise ”throws a wrench

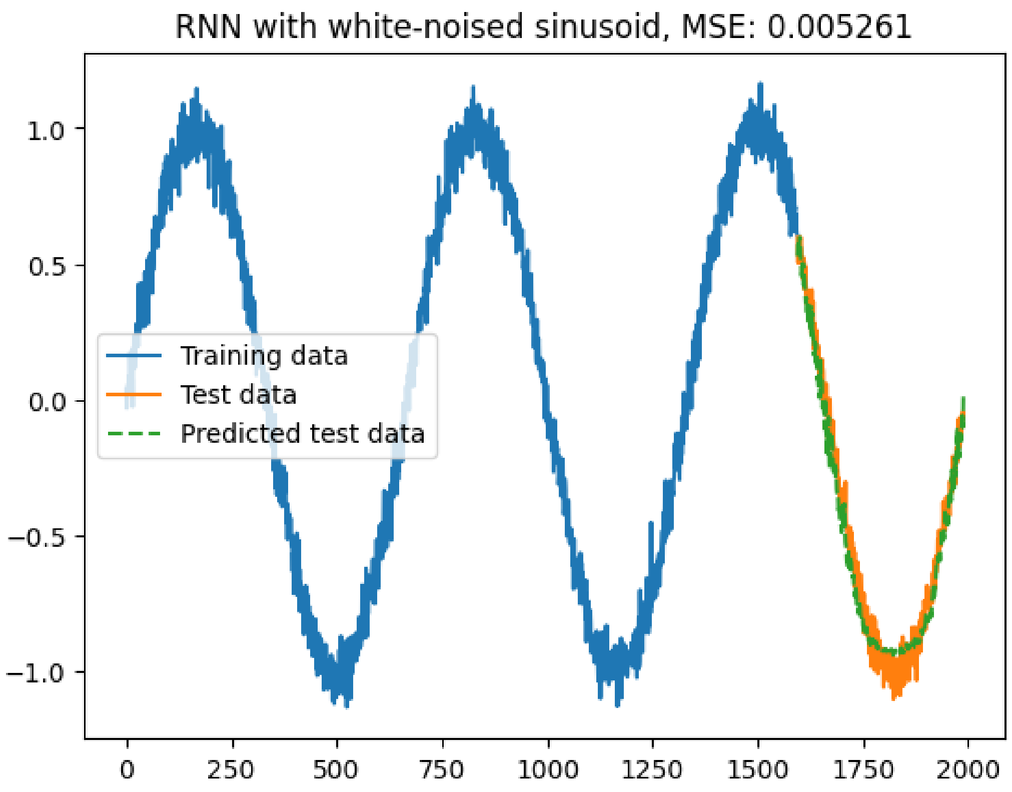

Figure 5: RNN predicting simple sinusoid series with white noise

in the system" for the LSTM model, but that the RNN model is more rigid against such challenges.

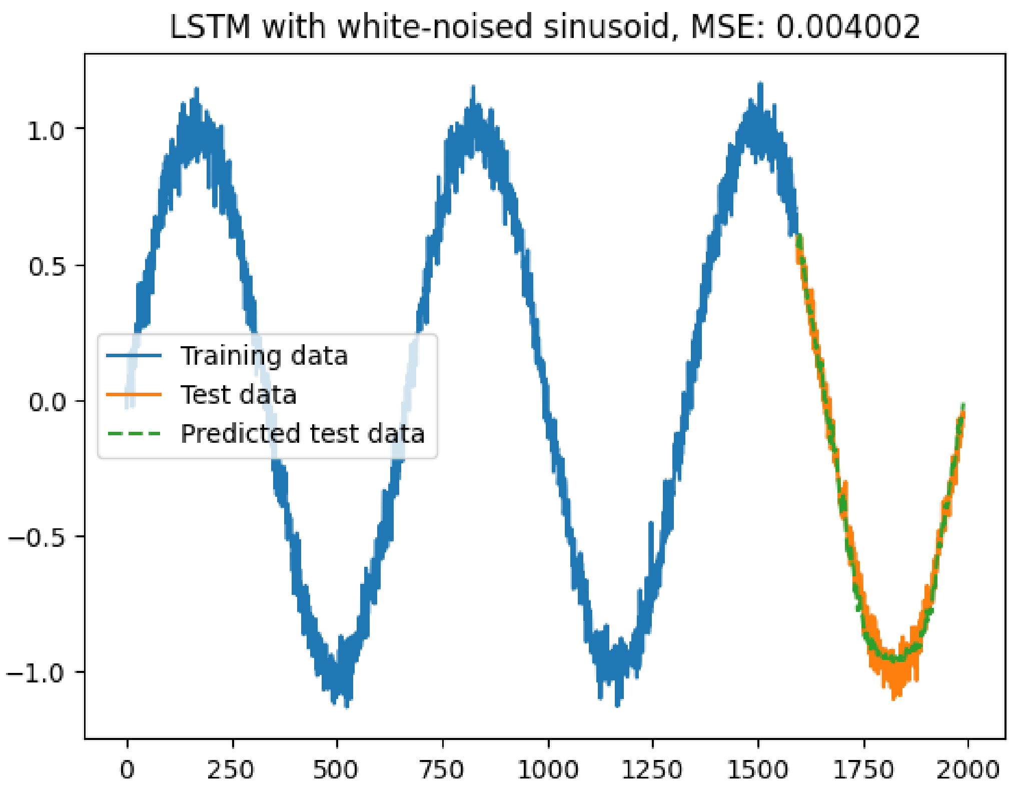

Figure 6: LSTM predicting simple sinusoid series with white noise

Task 1.2 - Predict white noise, random walk, an ARMA process using neural networks

For task 1.2, the goal is to predict a white noise series, a random walk series, and an ARMA process using a neural network of choice. For this scenario, an LSTM model was created with 100 units and a dense layer for the output. For the activation function, Tanh was used once more due the nature of the data sets. Finally, MSE and Adam were used as loss function and optimizer, respectively. All three series were synthetically generated and used, one after the other, to generate a prediction series. The results can be found in fig. 7 below. Using the metric scores, it seems that the LSTM did not achieve as favourable results as previous tasks did.

Discussion

The neural network was designed as an LSTM model, with 100 nodes for its single hidden layer. The output layer was designed as a dense two-node

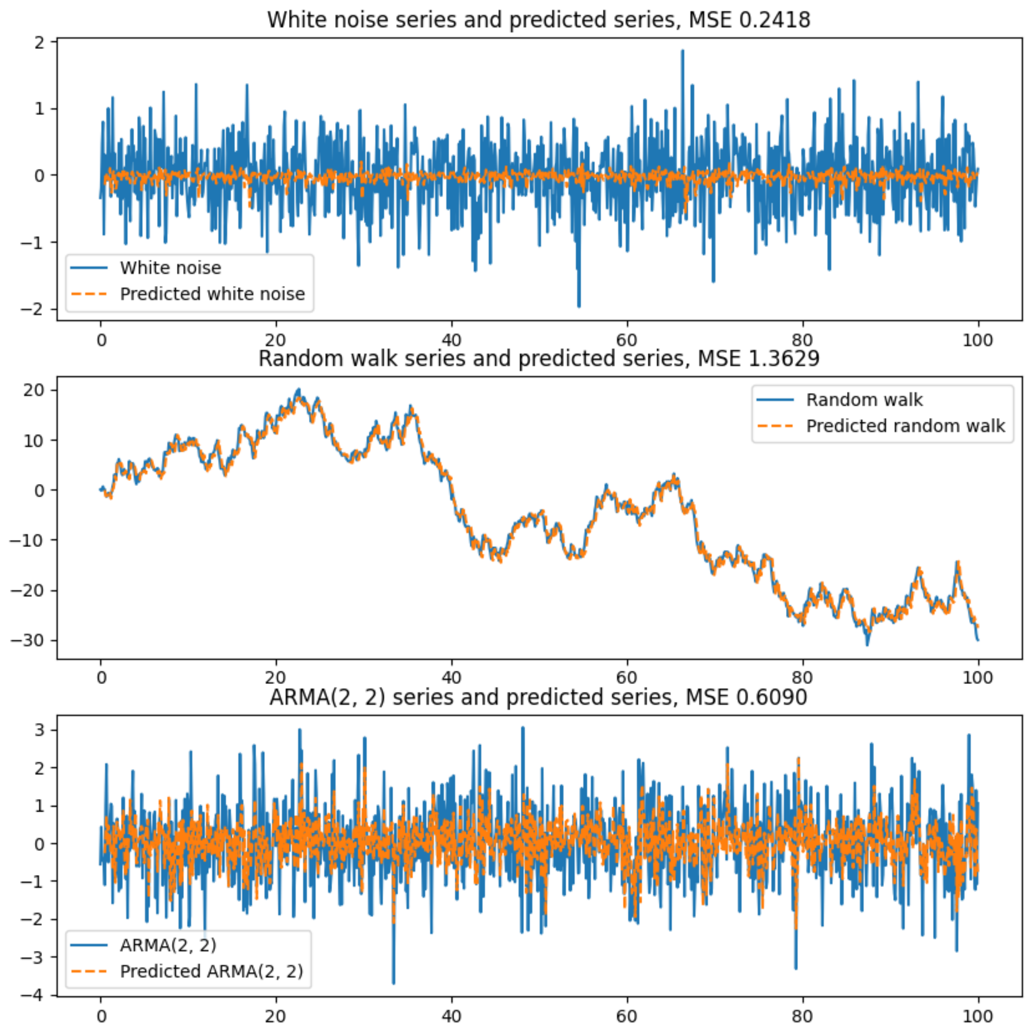

Figure 7: LSTM predicting three synthetic series

layer. A Tanh-activation function was used, and MSE loss function as well as Adam optimizer were implemented. For each data set, 50 epochs were used to train the model. A lower amount would suffice, but to attempt achieving better MSE values a higher amount of epochs were used.

The neural network did, ultimately, fit reasonably well to the random walk series and the ARIMA series upon visual inspection. The white noise series did not achieve as good results. Despite this, as the MSE values indicate, the neural network was not well-fitted to the data set. Closer inspection on the generated data and the predicted data will most likely indicated a larger discrepancy than can be seen from the figure. The reason for the network not fitting well to the data is most likely due to the random nature of these data sets. There are no clear patterns or trends which creates a challenge for the network to predict the random series.

It can be noted that the MSE losses for various methods were not comparable. In the case for the random walk series, it achieved the largest MSE score of 1.3629. In comparison, the ARMA(2, 2) series achieved a much lower MSE value of 0.6090. Despite white noise series not working as intended, it achieved the lowest MSE score of 0.2418. This would indicate that the white noise series came the closest to achieving a satisfactory result, but as can be visually inspected this is a result of the smaller series amplitude which white noise utilizes. In comparison, the random walk series and the ARMA(2, 2) series accomplishes a much more fitting predicted series.

Task 1.3 - Comparison with ARIMA-based modeling and prediction

In this task, the goal is to generate a Fibonacci-series with a given length (e.g. 50 steps), add noise to the data set (around 19:1 signal:noise ratio), then predict it using MLP, RNN, LSTM, and finally an ARIMA process. Finally, a comparison between the various models is to be made using Mean Square Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

The generated Fibonacci sequence and its white-noise counterpart can be seen in fig. 8 below. The data set is logarithmic to allow a better visual representation. Note that the noise is around 5%.

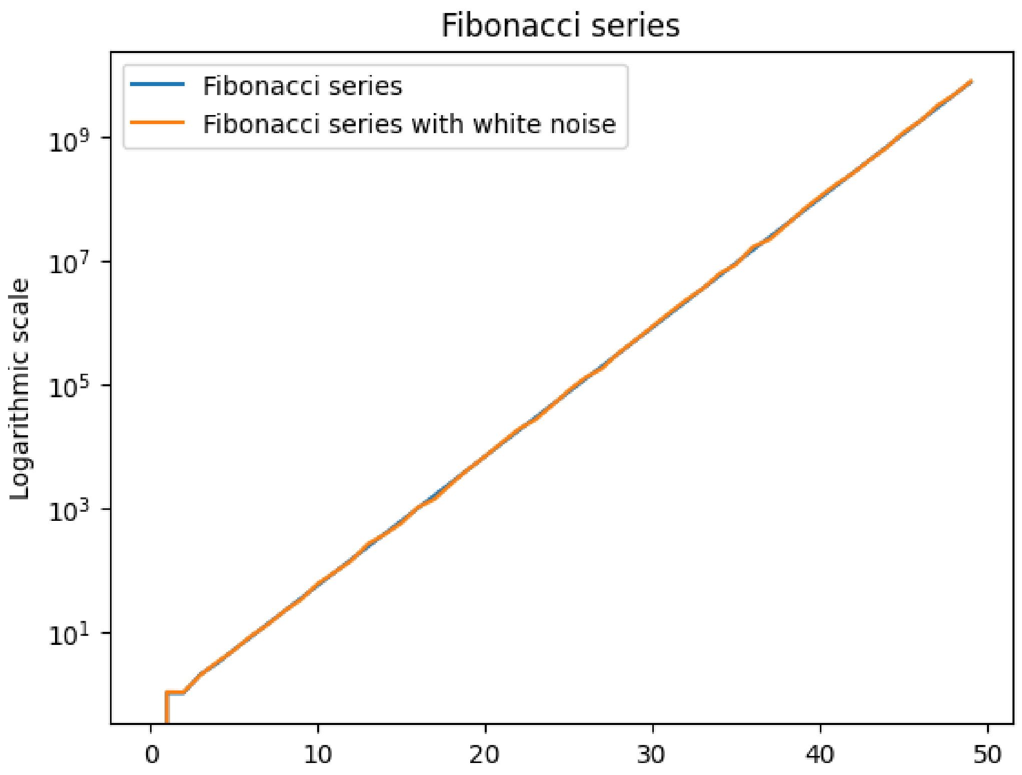

Figure 8: Fibonacci series and white-noise counterpart

The MLP was designed with a 4-100-2 pattern (input-hidden-output nodes). Given the data sets positive values, RELU was chosen as the activation function. MSE and Adam were, as usual, the loss function and optimizer. The model was given 10000 epochs to train. Results can be seen in the table below.

MSE	0.2742
MAE	0.4696
MAPE	0.0214

For the RNN, a 5-2 (input-output) model was designed with a dense layer. RELU, MSE and Adam were used here as well. The training ran for 100 epochs. The results can be seen in the table below.

MSE	103.3880
MAE	10.1509
MAPE	0.4649

The LSTM model was designed similarly to the RNN model. Here, a 100 epochs were used as well to train the data and achieved the following results.

MSE	476.3396
MAE	21.8175
MAPE	1.0000

Finally, an ARIMA model was created using the Box-Jenkins method. ARIMA requires a stationary data set and thus the series was first logarithmized[2] (note to achieve similar methodology, the dataset for the neural networks were also logarithmized before training) and then differenced once. The ADF statistic and its p-value achieved acceptable levels, which resulted in a stationary series. Following that, ACF and PACF graphs were used to identify AR(p) and MA(q) orders, which seemed to be around 3 for both. Finally, a grid-based search using the AIC criterion was conducted to find a fitting model for the differenced data set. This resulted in an ARIMA(3, 0, 1) process.

Upon fitting the model with the data set and using it to predict the sequence, the metrics were quite favourable and are displayed below.

MSE	0.0022
MAE	0.0379
MAPE	0.0835

[2]A transformation function to achieve linearity of an exponential trend.

A side-by-side comparison of all the models can be seen below

	MLP	RNN	LSTM	ARIMA
MSE	0.2742	103.3880	476.3396	0.0022
MAE	0.4696	10.1509	21.8175	0.0379
MAPE	0.0214	0.4649	1.0000	0.0835

A final comparison is to be made where the signal:noise ratio is increased. Using a 3:1 signal:noise ratio, the following metrics were attained

	MLP	RNN	LSTM	ARIMA
MSE	1.1361	106.7416	571.9409	0.0030
MAE	0.9516	10.2942	15.0245	0.0451
MAPE	0.0442	0.4763	1.0000	0.1012

Discussion
As aforementioned, between 100 and 10000 epochs were used to train the MLP, RNN, and LSTM models in an attempt to achieve more favourable scores. A quick overview of the various epochs did not seem to affect the accuracy metrics considerably. A learning rate of 0.01 was used. MSE was used as a loss function and Adam as the optimizer.

The designed neural networks did not conform to the data set well enough and resulted in terrible performance. Upon later discussion during the workshop, it was revealed that my models were most likely flawed in design. There would be no need for a four-input model, as only the previous two values are required to account for the next value in a Fibonacci series[3], but I was unable to replicate better values despite using a two-step input, one-step output. Overall, as can be seen from the metrics, none of the models achieved good enough results. On the other hand, the ARIMA process managed very favourable results. This indicates that neural networks are not always the best method to solve a regression problem. Furthermore, this is most likely due to ARIMA being a purely mathematical function and thus is better suited to predicting mathematical models, such as the Fibonacci sequence.
ADDENDUM: Upon further revision based on the feedback from the course team, new results have been presented. These results have overwritten the previous, faulty numbers but the author has chosen to keep the paragraph as it showcases the thought process which followed in regards to the workshop.

[3] $F_n = F_{n-1} + F_{n-2}$

Upon logarithmizing the dataset prior to feeding it to the neural networks (which was not done before), the achieved results were much more reasonable. the RNN and LSTM did not achieve nearly as well results as the MLP did. The MLP method did well enough to be considered satisfactory, but compared to the ARIMA model it is still somewhat disappointing. As mentioned before, the ARIMA model is a suitable method for mathematical models and thus not unreasonable to achieve such favourable scores.

It is apparent that the neural networks have similar results despite increased noise. The ARIMA model, on the other hand, seems to maintain a slightly lesser accuracy. Despite this, it attains favourable results.

Task 2 - Decomposition-based anomaly detection

In this task, the objective is to detect anomalies in time series through the use of decomposition. Decomposition is a technique where, much like the name suggests, the time series is decomposed into a trend, seasonal, and residual series. From these decomposed series, one may extract features through visually inspecting the various series. It is notable that decomposition comes with a weakness - namely, it is limited to detecting anomalies in in-sample predictions. Future data points are required to be included into the pre-existing data set, and the entire process run again. This is quite costly and thereby the method is limited to existing samples.

Task 2.1 - Anomaly identification in global land temperature changes

In task 2.1, a data set for global land temperature changes ranging years $\in [1880, 2020]$ is provided. The first assignment is to inject one or several anomalies into the existing time series. As can be seen in fig. 9 below, two points have been artificially revised to detail extreme variation, easing the process of detecting anomalies.

Upon decomposing the limited time series using a seasonal decomposition, trend, seasonal, and residual series can be seen in fig. 10. The uppermost graph depicts the full series. Note that the residual series (bottom) clearly indicates the two anomalous data points.

Finally, anomalies can be detected using the z-score and box plot as well. The z-score is a statistical method to determine how many standard deviations an element in the time series is off by in a normal distribution. This is done by calculating the distance from the mean of the series, and standardizing using the standard deviation:

$$z = \frac{x - \mu}{\sigma}$$

The score of the element is then compared to a threshold, and should fall outside the threshold, it is considered an anomaly. Typically, a score of $> ||3||$ can be considered anomalous, as this corresponds to a 99.7% confidence interval. This was the threshold for the experiment as well, which resulted in

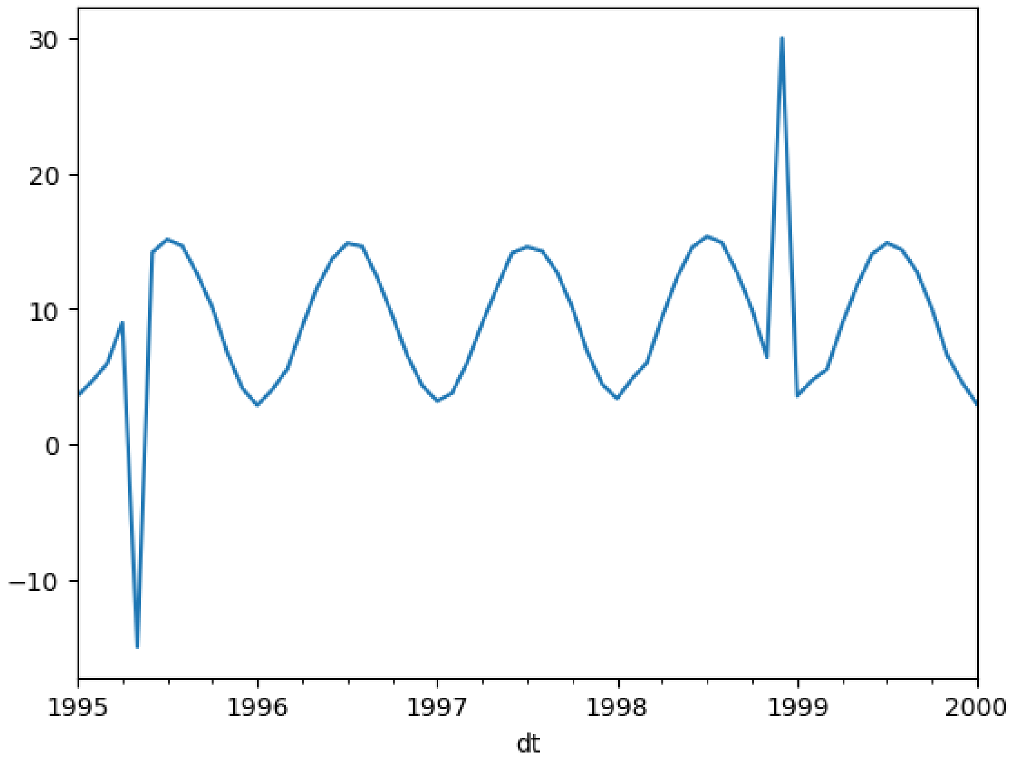

Figure 9: Average land temperature change with injected anomalies, limited [1995 : 2000]

two anomalous data points being detected at the expected positions (namely, the injected datetime positions).

A box plot can indicate anomalous data points as well. It is a visualization of statistical method, where the mean, median, and Q1-Q4 percentiles of the normal distribution are presented. Should a data point fall outside the Q1 and Q4 limits, it will be clearly indicated in the plot. These points are clearly shown in fig. 11 below.

Discussion

Decomposition seems to clearly separate trend, season, and remainder components. This pertains to data with the previously-mentioned components. Were a white-noise series to be decomposed, there would not be clear components from which features could be extracted. It is not clear either whether there is a general rule that determines which part belongs to the various

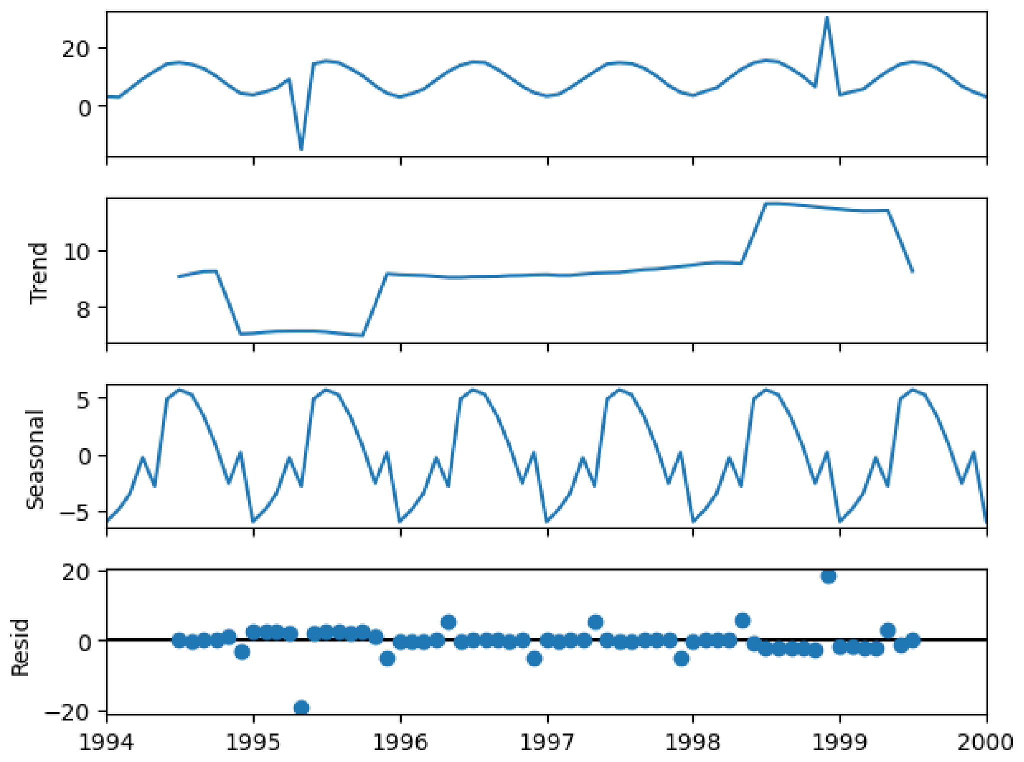

Figure 10: From top to bottom, original limited series, trend, seasonal, and residual decompositions

decomposed series, but that it depends on then various algorithms. Finally, there is a small but clear upwards trend in the overall land average temperature. This can be detected in the decomposed trend series of the complete time series, which has not been presented in this report.

Task 3 - Prediction-based anomaly detection

Task 3 pertains to detecting anomalies using a prediction-based method. The method depends on generating a prediction using neural networks or an ARIMA process, then calculating the residual series. Using the residual series, one may find the anomalies and eliminate them from the original time series.

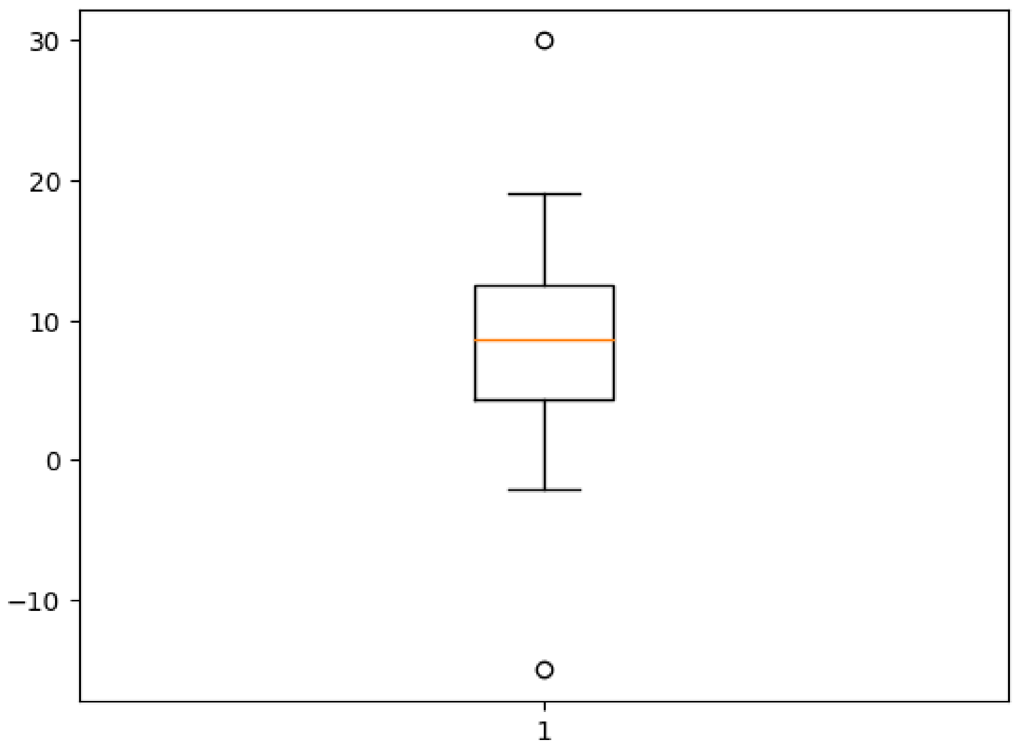

Figure 11: Box plot of the average land temperature series, indicating two anomalous data points

Task 3.1 - Anomaly detection for uni-variate series with ARIMA

In subtask 3.1, the objective is to detect anomalies in a uni-variate series using an ARIMA process. The data set pertains to global land temperature anomalies from 1880 to 2015. The data set is visualized using EDA in fig.12 below.

Mean and standard deviation of the data set can be seen in the table below.

Mean:	0.0149
Standard deviation:	0.1996

Using the Box-Jenkins method, an ARIMA(2, 1, 4) model was constructed and fitted to the data. Following that, in-sample predictions were generated and a prediction errors series could be calculated. Both are presented in figs.

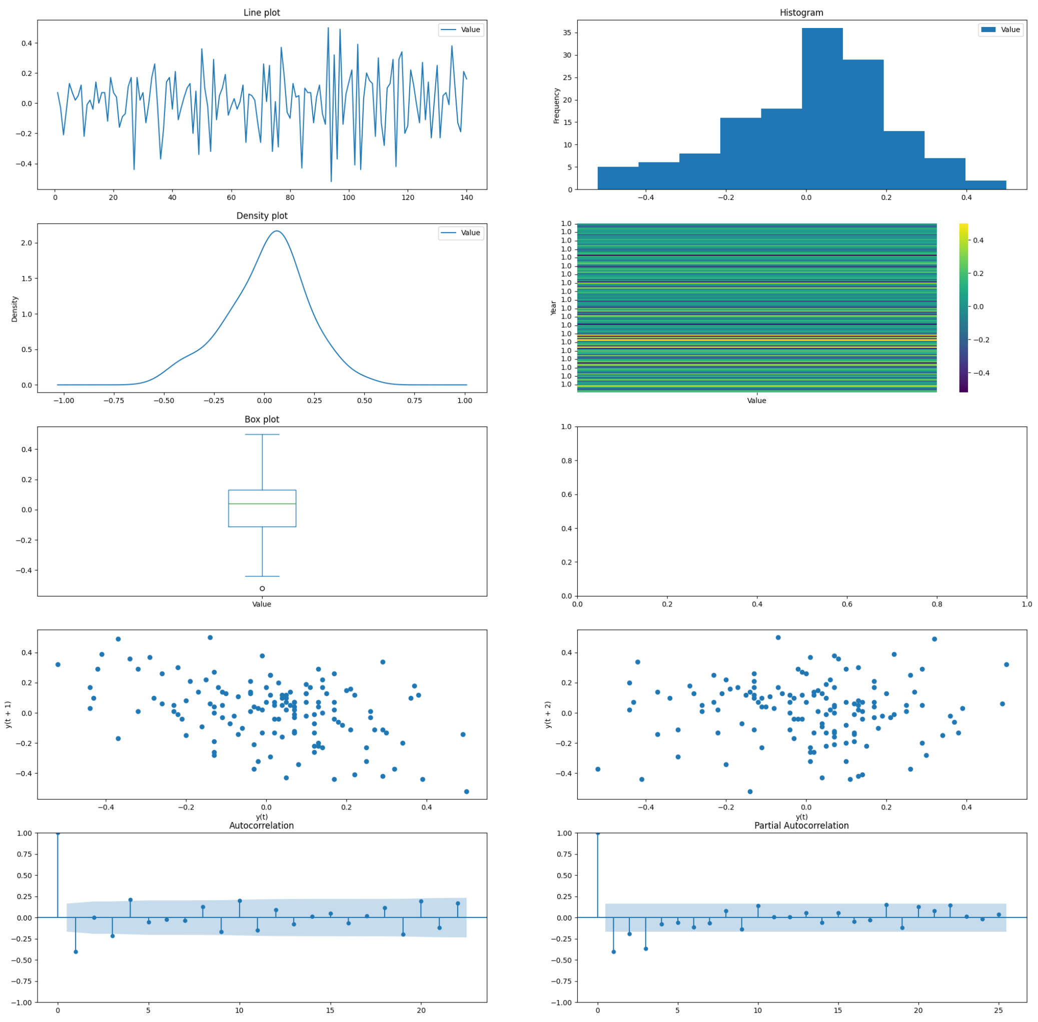

Figure 12: Line, density, box, lag-1 and lag-2 plots as well as histogram and heat map of global land temperature anomalies.

13 and 14 below. Utilizing a Ljungbox test, the residual was determined to be random.

Using z-score (confidence interval 98% for 2% anomaly ratio) and box plot, several anomalous data points could be detected. The z-score method found one additional anomaly which the box plot did not indicate. The data points can be found plotted in fig. 15 below.

Discussion

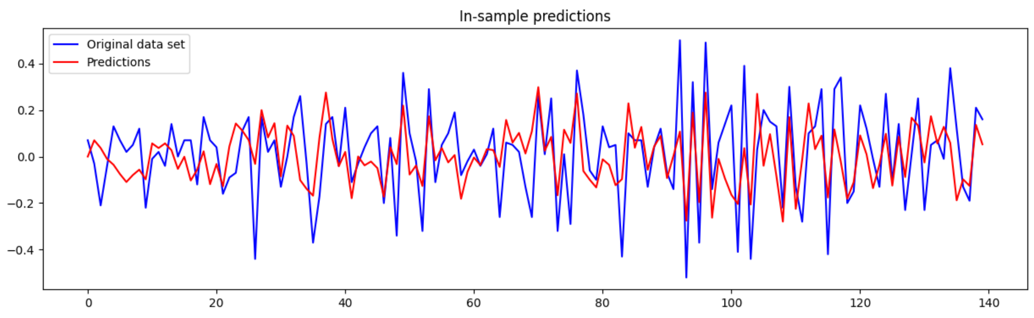

Figure 13: In-sample predictions using the ARIMA model and the original data set.

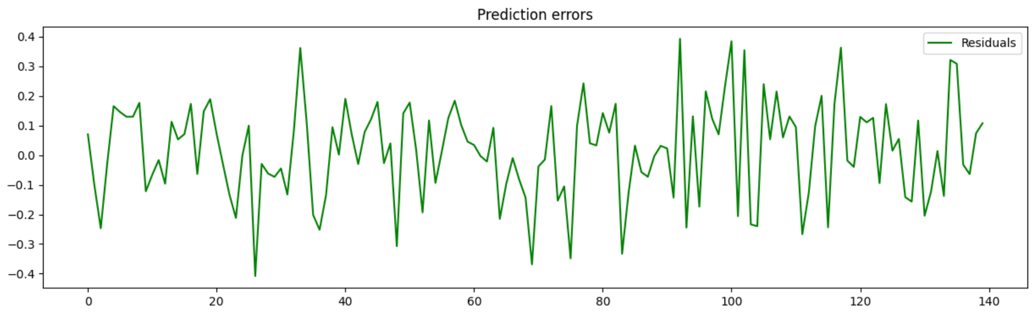

Figure 14: Residual series (prediction errors series) calculated from original data set and prediction data set.

This method achieves anomaly detection quite differently from decomposition-based anomaly detection. The largest benefit this method achieves over decomposition-based anomaly-detection is that it can allow for out-of-sample anomaly detection. Furthermore, decomposition seems to rely mostly on visually identifying anomalies. This can be useful for gaining intuition and insight into the data, as visually perceiving the data is easier for humans than relying on complicated formulas. In the end, they achieve the same result. But this only applies for the same data set, and not for future data points. The method of achieving the results also differ, which may play into strengths and weaknesses of the researcher/engineer. Finally, as mentioned the confidence interval was set to be at 98% to correspond with the 2% anomaly ratio as specified in the instructions. This results in a z-score ≈ 2.05.

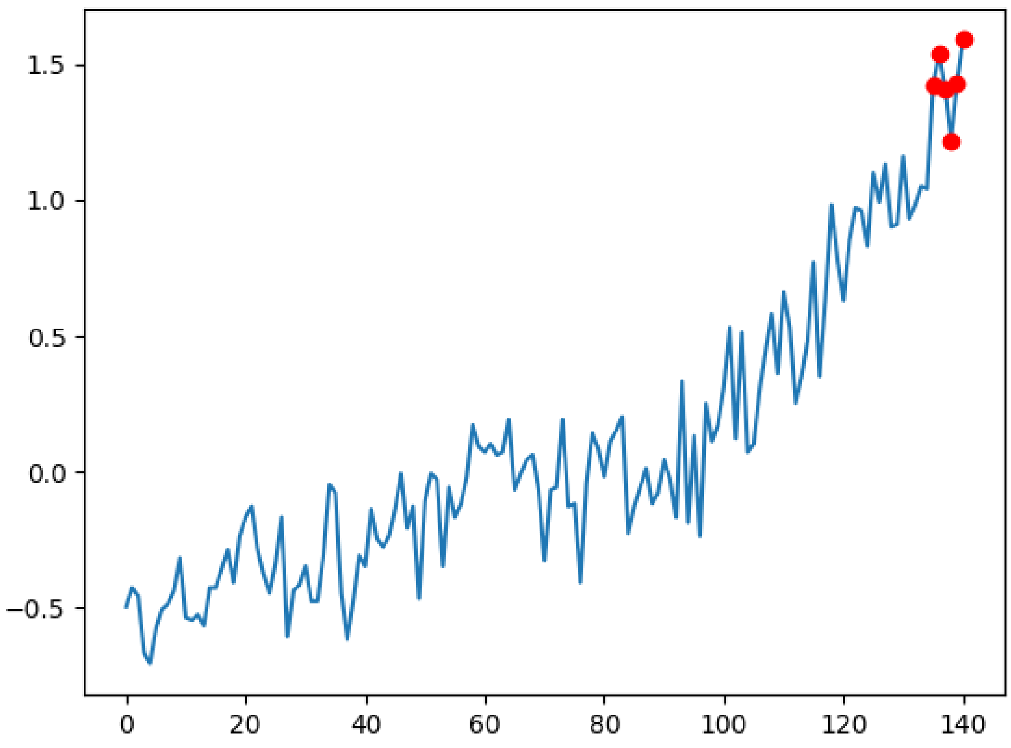

Figure 15: Anomalies marked in red on original data series.

Task 3.2 - Anomaly detection with LSTM in ECG signals

In task 3.2, the goal is to detect anomalies using LSTM. The data set contains signals from an ECG, in the form of two leads - MLII and V5. The instructions specify to handle two separate cases, (1) with two uni-variate series (MLII and V5 separately) and (2) with a bi-variate series. Due to time limitation, (2) was not accomplished. The course team are aware of this.

For starters, a simple EDA with line, lag-1, and lag-2 plots was conducted for the uni-variate series. Results can be seen in fig. 16 below.

Following this, the data was split into a training and testing set with a typical 8:2 ratio. For the LSTM model, a INPUT-5-2 (input size, hidden nodes, output) design was chosen. For this task, the input is specified to be 4, 8, and 16. For this reason, there are three iterations of the LSTM model where both series are run through it. In total, the LSTM model is trained six times

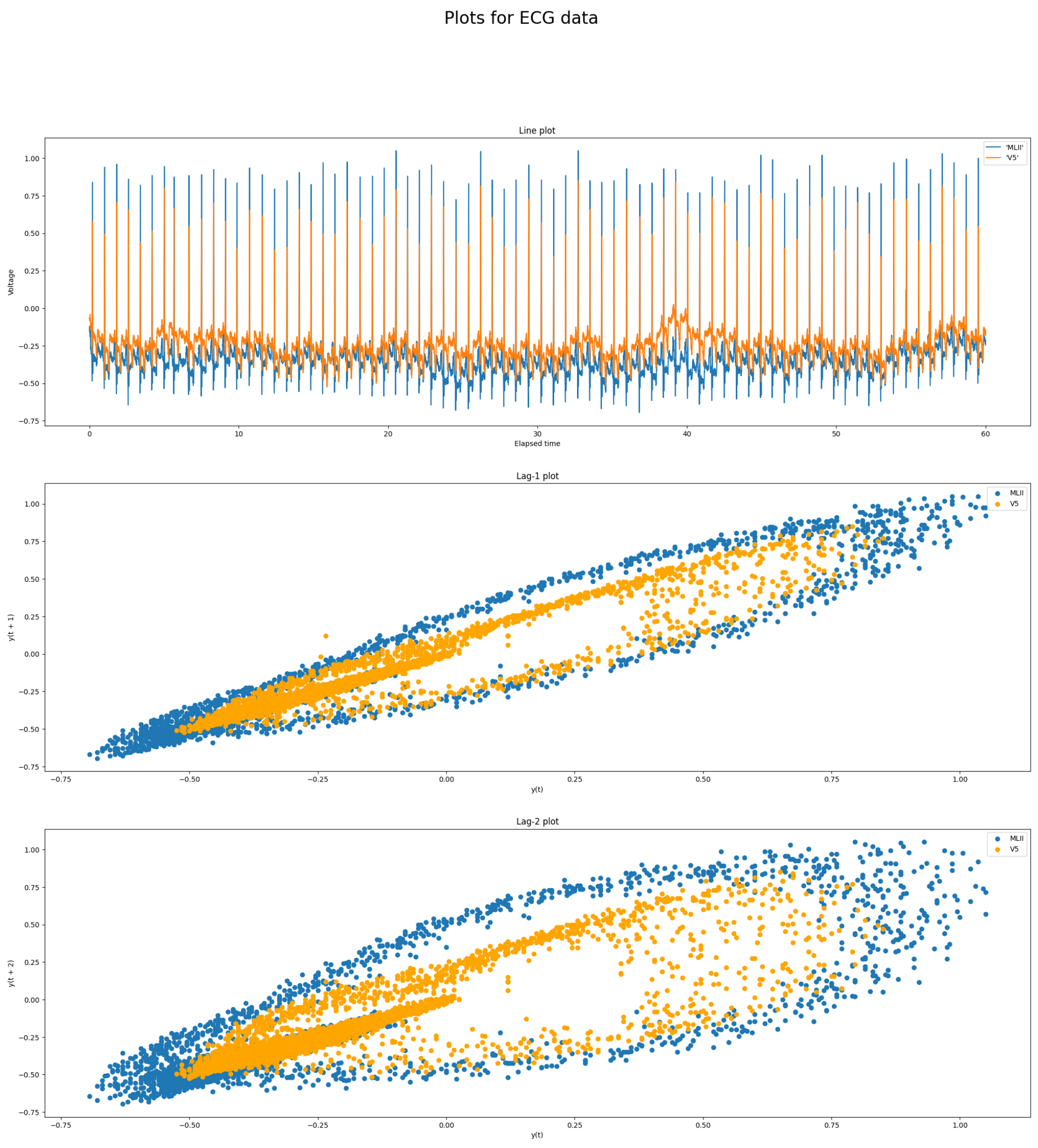

Figure 16: Line, lag-1, and lag-2 plots of the ECG signals.

over. To determine the validity and accuracy of each model, the MSE, MAE, and MAPE are noted and compared. See the results in the table below. It is notable that the input size did not affect the accuracy in a valuable matter.

	Input size	MLII	V5
	4		
MSE		0.0085	0.0093
MAE		0.1755	0.1846
MAPE		67.0519	111.9436
	8		
MSE		0.0084	0.0092
MAE		0.1921	0.1882
MAPE		69.1599	111.0437
	16		
MSE		0.0075	0.0103
MAE		0.1787	0.1980
MAPE		65.8397	118.4074

Due to time limitation, the anomaly detection was not fully implemented. The course team have been made aware of this.

Discussion
As mentioned, the LSTM network was designed with a single hidden layer of 5 nodes and a dense output layer with 2 nodes. The loss function utilized MSE and an Adam optimizer was utilized. A learning rate of 0.001 was used and 5 epochs were deemed enough to achieve favourable accuracy. Judging by the accuracy metrics, the model fit the uni-variate ECG signals quite well in all iterations. As aforementioned, and somewhat surprisingly, the chosen input sizes did not affect the accuracy metric in any noticeable way.

Task 4 - Clustering-based Anomaly detection

The final method for detecting anomalies is a clustering-based one. Clustering is a method of setting data points into different groups according to their closeness (based on some distance metric) to the center of the cluster. The strength of this method is that it can be done to unsupervised learning data, which is useful when a ground truth is missing. On the other hand, it may not be obvious how accurate the clustering is. There are two common methods for clustering - K-means and Self-Organizing Maps. Due to time limitation, only K-means was explored. The course team are aware of this.

Task 4.1 - Anomaly detection in a bivariate series

For task 4.1, a two-variable series is generated. Both variables are generated from a Gaussian distribution of (mean, standard variation) = (0, 2) and (1, 2), respectively, with 200 data points. Furthermore, a 2% anomaly threshold is set. This means that 4 anomalies should be found in the total data set. Due to misunderstanding the lab instructions, a 2% anomaly threshold for *each cluster* was implemented (which results in 5 anomalies). In principle, the method will be the same, but there will be more anomalies.

Upon generating the bi-variate series, a line plot and a scatter plot was created to allow for EDA. This can be seen in fig. 17 below.

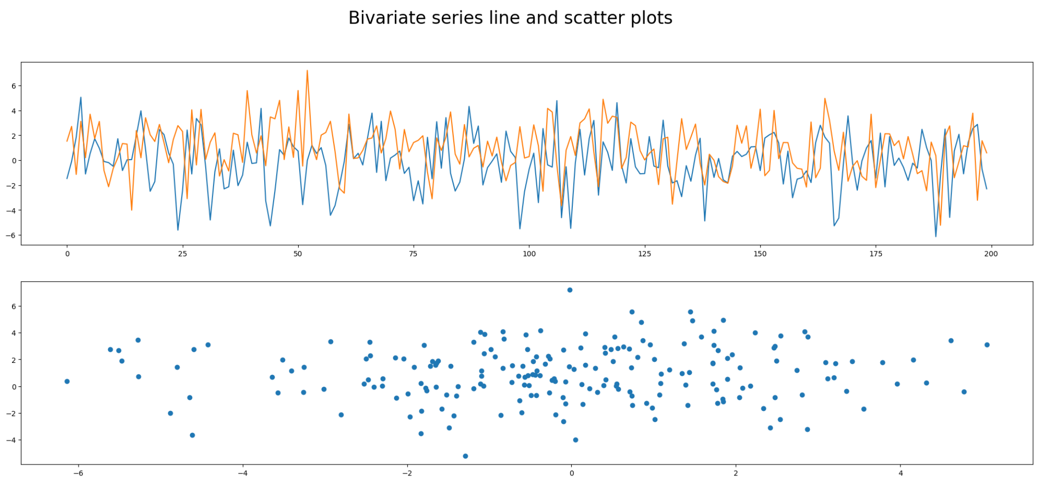

Figure 17: Line and scatter plots for the bi-variate series.

Following this, a reasonable deduction was made to determine the number of clusters to be set as five. The clustering using K-Means generated the following results:

For each element, the distance from the element to its respective cluster was calculated and stored. The distance metric used was euclidean distance. The distance list for each centroid was sorted, and the furthest 2% of the data points were considered anomalous. Using this information, circles for each cluster were drawn to visualize which points belonged inside the cluster and which were considered outliers. See fig. 19.

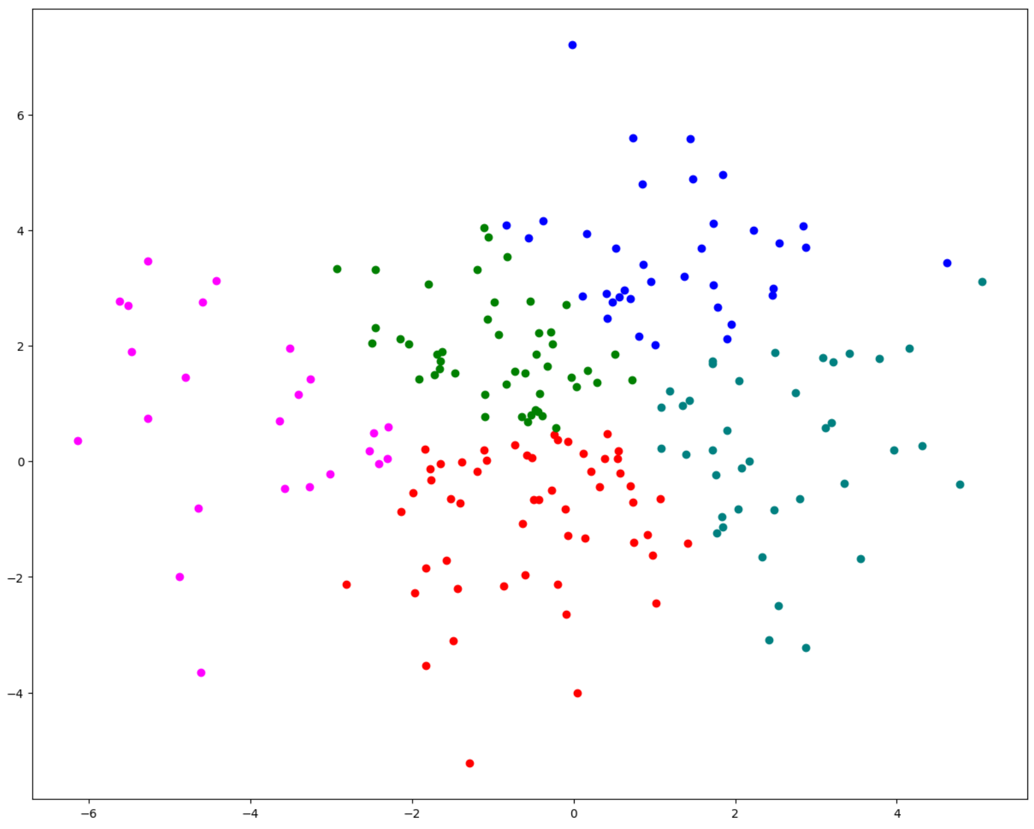

Figure 18: Line and scatter plots for the bi-variate series.

Discussion

Deciding the number of clusters is not a simple choice. Often it relies on experience, which was lacking here. It is clear that a single cluster defeats the purpose of clustering, and two clusters would not assist considerably. One could argue that three clusters would suffice, but I believe the circles would become too large and cover too much white space. Four or five clusters were reasonable following this line of thought.

As mentioned, the Euclidean distance metric was used. DTW was not chosen as both series are equally as long and there is no clear time warping. The Manhattan distance metric was also a possibility.

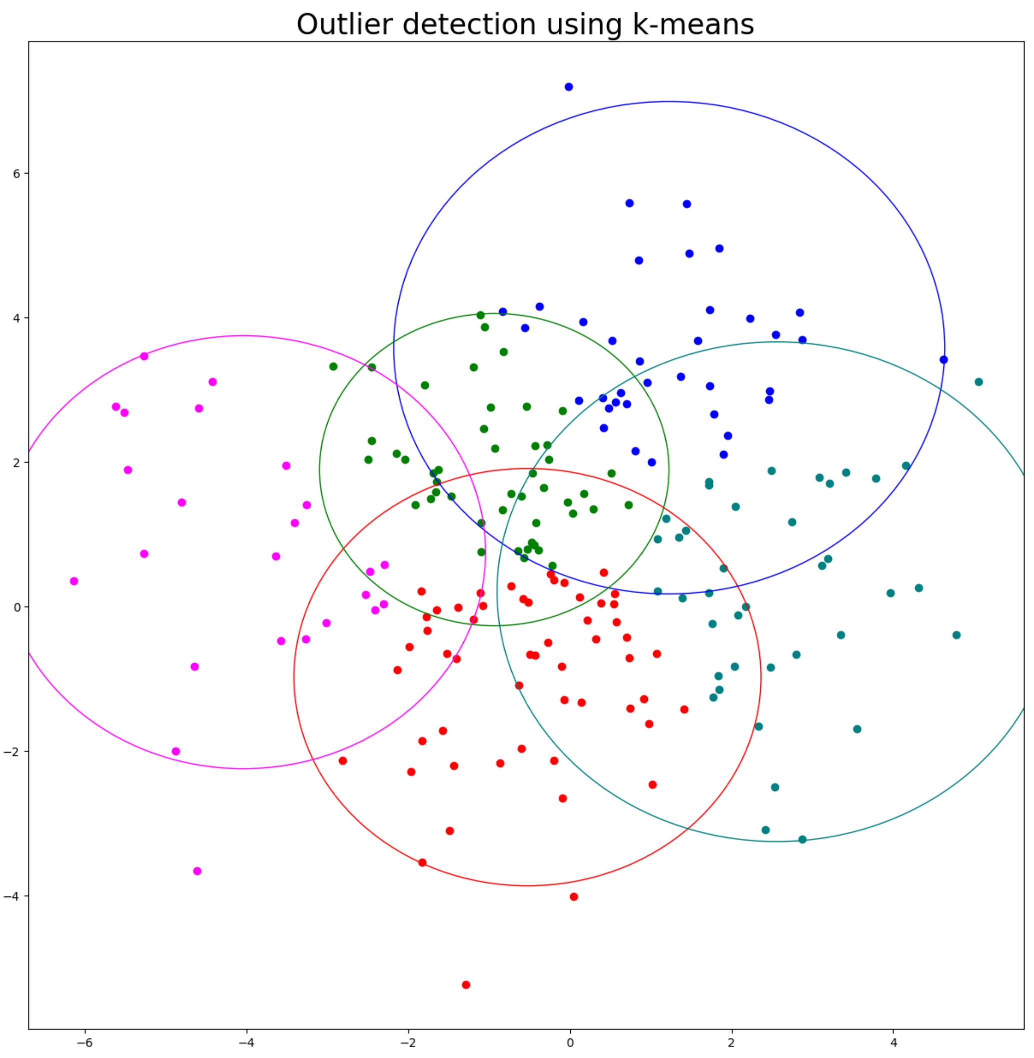

Figure 19: Outliers detected using K-Means.

Task 5 - Summary

To conclude this report, a summary of the project is written. In the first part, we will discuss the the two different approaches - neural networks and ARIMA - to time series modeling and prediction. In the second part, the three different anomaly detection methods will be processed.

Time series prediction

Neural networks and the ARIMA process are two wholly different approaches to the same problem. Neural networks (specifically MLP, RNN, and LSTM) work by training on a data set and correcting their weights over a several iterations, in an attempt to reduce the loss value to a favourable level of accuracy. Once the neural network has been trained sufficiently, predictions are generated and compared to ground truth (for in-sample predictions). ARIMA, on the other hand, is trained different. It requires a fitting model to be found, which can often begin with finding the separate components' values (using the Box-Jenkins method), then combining the components but running a grid-based search using some criterion such as AIC, to find a fitting model with both components combined. Once this is done, the model is *then* trained on the data and fitted to it. Only then can the model generate predictions.

When comparing neural networks such as MLP, RNN, and LSTM to ARIMA there are some clear differences that come to mind. From the earliest tasks with synthetic time series, it has been obvious that not all series can be predicted by neural networks alone. For purely mathematical functions, the ARIMA process seems a better fit to achieve favourable accuracy. This is a reasonable inference given its statistical nature. Despite this, the ARIMA process requires considerable pre-work prior to fitting the model. The Box-Jenkins method requires achieving stationarity of the data, checking for seasonality, conducting EDA to find values for the AR- and MA-components. Finally, the quite resource-costly grid-based search to find a fitting model is cumbersome. Compared to ARIMA, neural networks require lesser preprocessing to transform the data into supervised learning data, and to create a model. Here, the engineer's understanding of what makes a good model plays more into the role than some brute-force method.

Approach	Assumption	Modeling & Prediction	Strength	Weakness
ARIMA	Stationarity, Auto-Correlation	Find stationarity, consider seasonality, find (p, d, q) values, conduct grid-based search, train the model, extract predictions	Statistical approach, good for pure mathematical functions	Resource-intensive, not as well-fit for complex mathematical functions
NNs	Supervised learning data, experience with modeling NNs	Pre-process data, model NN, find good hyperparameters, train the model, compare to ground truth, evaluate accuracy	Works well with non-mathematical data sets	Not as strong for purely mathematical functions

Anomaly detection
For anomaly detection, three different methods have been considered and experimented with. These are the decomposition-based, prediction-based, and clustering-based anomaly-detection methods.
Decomposition works by decomposing a series into three components: Trend, seasonal, and residual series. From the residual series, it is often apparent where the anomalous data points arise. **Prediction-based** approach works by training a model to create an in-sample prediction which is then used to find residual series. Following that, anomalous data points can then be found in the residual series, using some anomaly threshold. **Clustering** works by clustering data points, then finding those furthest out according to some threshold and labeling them as outliers.

While decomposition allows for easy visualization of anomalous data points (which gives good intuition), it is limited to the existing data set. Future data points (out-of-sample predictions) are costly to find anomalies in using decomposition, as it would require the process to run again. In contrast, due to the nature of the prediction-based approach, finding out-of-sample anomalies is quite a simple process. A weakness for prediction-based approach is of course the need for ground truth. This is covered by the clustering-based anomaly detection, as it requires no ground truth to find anomalous points. At the same time, using unsupervised learning data set is also clustering's weakness, as one may not be certain if the outliers truly are anomalous or not.

Method	Anomaly detection	Strength	Weakness
Decomposition	Decompose time series, residual reveals anomalies	Visual and intuitive	Limited to existing data set
Prediction	Generate prediction using NN or ARIMA, calculate residual, find anomalies given threshold	Allows for out-of-sample anomalies	Requires ground truth
Clustering	Cluster data set, set a threshold, consider data points farthest from centroid given threshold as outliers	No ground truth required	Uncertainty whether outliers are anomalous or not
Z-score	Calculated z-score for the data point and compare it to threshold to determine anomaly	Mathematically intuitive	Uncertainty regarding threshold, depends on use case