Obtaining Cycle Breakpoints and Predicting Next Cycles: An Exploratory Analysis of Generative Programming and Time Series Forecasting to Derive Insight for Artificial Muscle Movement

Navraj Narula
Creative Machines Lab
Computer Science Department
nnn2112@columbia.edu
December 22, 2017
Fall 2017

Abstract

Abhijit Naskar, author of *The Art of Neuroscience in Everything*, once said: "Artificial intelligence can be a supplement to human insight, not substitute." Working with a dataset exceeding a hundred thousand rows can be troublesome for any human to tackle in a timely manner; however, given the fact that a human has made a computer aware of meaning associated with the unfamiliar content it needs to parse, a machine can deliver outputs both quickly and continuously.

During the course of this semester, I worked to help the programs I built understand the meaning behind a large dataset containing information pertaining to the time and load of 47 cycles related to the movement of a soft, artificial muscle.

In mathematically breaking down the dataset, I was able to obtain the start point for each cycle, end point for each cycle, the total time per cycle, the heating time per cycle, and the cooling time per cycle. After providing this narration for my dataset, I was then able to think more critically about how to predict the next three cycles in my dataset. By applying the concept of time series forecasting, I was able to build both an autoregression model and a persistence model in order predict next values.

Despite improvements to be made in regards to precision, my results for each task indicate a close correlation with the actual results.

Goals

The objective of the project is to take advantage of machine learning and artificial intelligence techniques to better understand the underlying data related to the movements of a soft, artificial muscle. Beyond exploratory analysis, another objective of the project is to not only assess existing algorithms related to data prediction, but also experiment with creating algorithms that could assist in the prediction of further cycles pertaining to the given dataset.

Background

In order to better understand the project, you may like to familiarize yourself with the overall research as it currently stands at the Creative Machines Lab as well as with mathematical and algorithmic concepts used in this paper to retrieve both the necessary calculations and models needed. You may refer to *Appendix A* for resources most pertinent to the lab, while the remainder of this paper will discuss the background necessary to understand algorithms applied to the dataset of cycles.

Literature Review

While the problem I intend to solve in regards to cycle breakpoints is isolated, the problem in regards to cycle prediction, or rather prediction of next values in a dataset, is not. Advanced machine learning algorithms that may take advantage of Bayesian or regressive concepts may be used as a baseline in regards to forecasting, but such algorithms themselves do not predict next values in a dataset. For instance, a Naive Bayes classifier will require that only one attribute in a dataset be predicted upon whereas when it comes to forecasting, the prediction required is that of an entire instance given previous instances. We can adopt the concept of training and testing sets from such algorithms; however, must also find a way to extend predictions beyond single attributes and beyond the length of a given dataset as well.

This is where time series forecasting comes into play. According to Brockwell and Davis, author of *Time Series: Theory and Methods*, a time series is a set of observations x_i with each one being recorded at a specified time *t* [1]. The dataset we have containing 47 cycles matches this definition in the sense that each instance corresponds to a specific recording in time. Aforementioned machine learning approaches can be applied to such datasets representing a time series problem.

Hassan and Nath, researchers at the University of Melbourne, developed a Hidden Markov Model-based tool for time series forecasting in regards to stock market forecasting. Four input features for a stock were considered: the opening price, the closing price, the highest price, and the lowest price of the stock. The target price is the next day's closing price. Hassan and Nath's tool yielded an R^2 value of 87.5%, indicating that the model adequately fit their given dataset [2].

In addition to Markov incorporation into time series analysis, Chris Chatfield, author of *The Analysis of Time Series*, expands on an another approach in regards to prediction: constructing an autoregression model. He discusses how its outputs, or predictions, are based on a linear combination of input values. In its base case, the model assumes that the observations made at the previous time steps are useful in predicting values in the next time steps as well. Since our dataset is linearly separable and continuous, applying an autoregression model might be a good start in regards to making predictions on top of the data that already exists. [3]

Jason Brownlee, the founding researcher at Machine Learning Mastery, introduces a predictive approach involving a persistence algorithm in which predictions are based on simply the given value in a previous time step rather than on multiple previous values. In predicting the monthly number of shampoo sales over a three year period, Brownlee was able to retrieve next rows for his given dataset that mimicked previous rows already present [4]. Such a model may also be fitting for our particular dataset of cycles.

Methodology, Technical Approach, and Results

In this section, I will be discussing my methodology in regards to tackling both problems given the dataset containing information related to the 47 cycles: one in regards to obtaining breakpoints for each cycle, and the other in regards to predicting next cycles given values already present in the dataset containing all 47 cycles. Before diving into the technical approach as well as the results retrieved for each method, I will also provide a high-level explanation of the dataset.

Dataset Information

Courtesy of Aslan Miriyev, a post-doctoral research scientist in the Creative Machines Lab, I was provided with a dataset containing information pertaining to the movement of a soft, artificial muscle [5]. The dataset itself contains 47 cycles in total; however, these cycles do not represent a single instance (i.e. row) in our dataset. Instead, a single cycle can span over a couple hundred of rows before terminating. The entire dataset itself actually contains 170,101 rows. Only two columns are present, both of numerical value: one being the time represented in seconds, and the other being the load (i.e. force of the muscle) represented in Newtons.

The first time value in the dataset starts at zero seconds, and following this it goes on to 0.0018 seconds and then, 0.1 seconds. After these initial values, each value in the time column now increments by 0.1 seconds. The very last value in the time column, signaling the end of 47 cycles, is 16770.9 seconds.

The first load value in the dataset starts at 0.06729 Newtons. This represents the minimum value starting the first cycle. Values following this will increase until a maximum value of about 130 Newtons is reached. After this value is reached, the pressure drops until the value once again becomes close to zero; thus, starting the next cycle. This occurs 47 times in our dataset, with a load value ending at 0.09558.

Obtaining Breakpoints for Each Cycle

As alluded to earlier, our dataset is continuous in regards to time; however, segmented in regards to load. Therefore, in order to determine the start and end points for each cycle, I zoomed in on the load column.

With prior knowledge that the pressure for each cycle starts around 0 and ends around 130, I can presume that each cycle has a starting point and an ending point. In mathematical terms, I can seek to find the local minima and local maxima for each cycle. According to Tri Lai, a postdoctoral associate at the Institute for Mathematics and its Applications at the University of Minnesota, a function has a local minimum at a certain point, *p*, if the function of *p* is less than or equal to the values of the function for points near *p*. The same concept applies to the local maximum; however, in this case, the function of *p* would be greater than or equal to the values of the function for points near *p* [6].

Within programming itself, the most intuitive way to retrieve such values is to use a built-in function. Python's NumPy library has functions to derive local minima and local maxima using a call to *extrema*, which retrieves the relative extrema in a dataset [7]. Applying this function to our load values returns misleading results, as seen in *Appendix B*.

I then began to write a function on my own to calculate the local minima and maxima. I tried varying approaches, all of which included iteration, conditionals, and self-defined functions. The most optimal approach I could come up with ran in $\Theta(n)$, or rather, linear time. The function is a generative function that seeks to group results based on minimum and maximum values as defined by conditionals which examine both the current value and the next value of the list passed through. You may view the function in its entirety in *Appendix C*.

When passing in the load parameter to the function, however, the values returned still did not represent the true minimum and maximum values. Though the function itself is implemented correctly and returns expected values according to the rules implemented, the output is incorrect because the load values in the dataset themselves oscillate. What is expected is that the first value should start at zero and continually increase until 130. Then, continually decrease from 130 until zero. However, values tend to increase and decrease along the way.

For instance, take the first three values in our load column: 0.06729, 0.07128, and 0.05453. Supposedly, the pressure value is supposed to increase after 0.06729; however, due to oscillation, the values start to decrease well before reaching 130 Newtons at 0.05453. Below are the first three outputs of the function:

```
Cycle: 1
Minimum value: -0.05887
Maximum value: 0.07505

Cycle: 2
Minimum value: 0.07159
Maximum value: 132.97716

Cycle: 3
Minimum value: 5.30435
Maximum value: 5.31258
```

As shown above, the values do not represent the true minimum and maximum values. In order to fix this error, I attempted multiple experiments in regards to rounding values in Excel. The best outcome that neared expected results turned out to be rounding values to the nearest whole number. This was accomplished by using Excel's ROUND function, specifying a value of zero. Below are some arbitrary values from the load column and the values they round to, according to the aforementioned function used:

0.51372 → 1
0.46894 → 0
2.49806 → 2
2.51084 → 3

By applying my self-constructed function to this new column and keeping track of the index of the rounded values, I was able to map the corresponding index to the original values in the load column. Below are the returned local minimum and local maximum results for the first three cycles:

```
Cycle: 1
Minimum value: 0.51372
Maximum value: 132.54321
```

```
Cycle: 2
Minimum value: 0.50146
Maximum value: 132.57836

Cycle: 3
Minimum value: 0.50416
Maximum value: 131.58451
```

The minimum and maximum values for the above points are closer to the known minimum value of zero and maximum value of 130 as they should be, so improvements have certainly been made. After retrieving the minimum and maximum values for each cycle, I proceeded to calculate the total time, heating time, and cooling time per cycle. The heating time for each cycle starts at the same time index for the minimum load value and ends at the same time index for the maximum load value. The cooling time for each cycle starts at the same time index for the maximum load value and ends at the same time index for the minimum load value. This pattern continues, cycle after cycle. The total time per cycle is simply the sum of the cooling time and the heating time. Below I have included all criterias for the first three cycles:

```
Cycle: 1
Minimum value: 0.51372
Maximum value: 132.54321
Total time per cycle: 242.29998
Cooling time per cycle: 161.29998
Heating time per cycle: 80.9

Cycle: 2
Minimum value: 0.50146
Maximum value: 132.57836
Total time per cycle: 334.6
Cooling time per cycle: 262.80003
Heating time per cycle: 71.7

Cycle: 3
Minimum value: 0.50416
Maximum value: 131.58451
Total time per cycle: 289.2
Cooling time per cycle: 235.8
Heating time per cycle: 53.29994
```

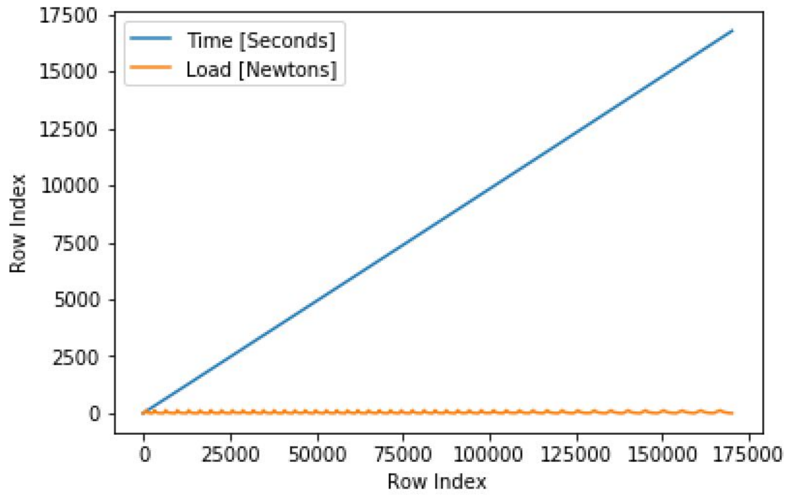
You may find all criterias for all 47 cycles in *Appendix D*.

Predicting Next Cycles: Time Series Forecasting in the Context of our Dataset

After obtaining breakpoints for each of the 47 cycles, the following task was to predict the next three cycles. Miriyev then provided me with a dataset containing actual data for all 50 cycles, in addition to the data I already have of the 47 cycles [8]. In order to predict the values for the next three cycles, I employed two methods related to time series forecasting to build both an autoregression model and a persistence model.

In order to perform time series forecasting, we must first have a dataset that is time-relevant. Time series adds an "explicit order dependence" between observations: a time dimension. This extra dimension can be considered both a constraint and a structure that provides extra source of information [9]. I believe that the dataset we have for our 47 cycles is a time-relevant dataset because it contains a sequence of observations that have taken place sequentially in time. The only caveat is that prediction of the next instance may not be easy to come by as each next point represents an instance in time over the actual timespan for a given cycle.

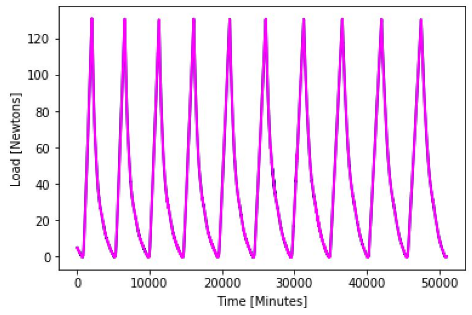
Our data for the 47 cycles currently looks like (note that both the x-axis and y-axis here represent row indices up to 170,101 since this is the length of our dataset):

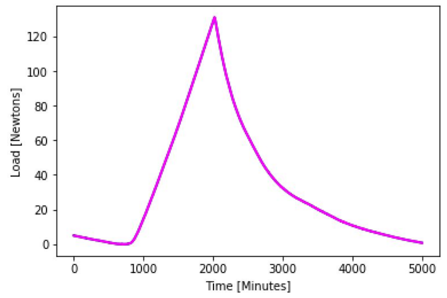


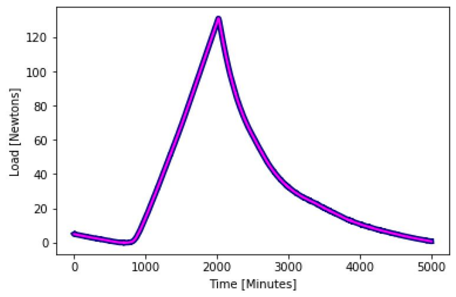
Observing the above graph, we can see that the time column increments at each row, so it makes sense that the time value is always increasing upward. We can see "peaks" and "valleys" in the values for the load column. This also makes sense due the fact that we know that these values oscillate between their minimum and maximum points.

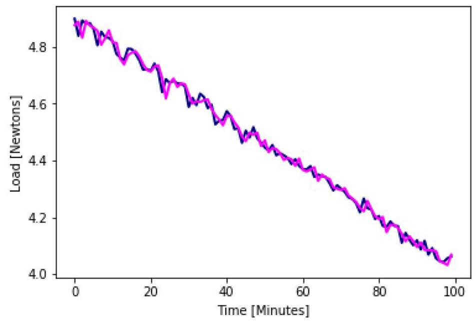
__Predicting Next Cycles: An Autoregression Model__

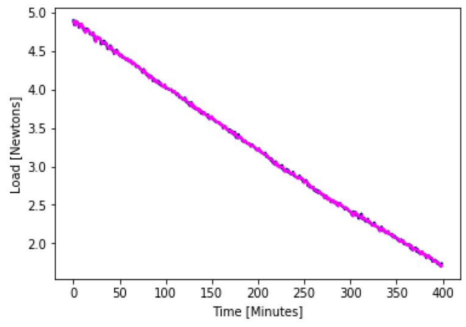
I selected an autoregression model, a forecasting model, to assist in making predictions. An autoregression model is viewed as a representation of a type of random process and can be used to describe time-varying processes in a dataset [3]. I have chosen to train my model on 70% of the data in the load column. When I increase or decrease this value, my model is not exactly representative. At each timestep, a prediction is made using a fitted model specified on the load parameter. I have included several visualizations below that present the results of my model. The blue line represents the actual values in the test set whereas the pink lines represent the predicted values. In each graph, the x-axis represents the row index and the y-axis represents the load.

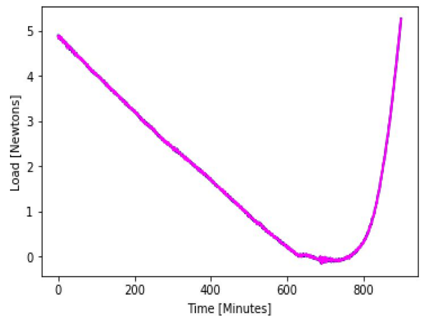












The first visualization at the top-left corner shows all 51,030 values in the test set plotted as well as all 51,030 values in the predicted output plotted. The second visualization in the top-middle displays the curve of one cycle, extracted from the previous graph. Given the fact that the overlap among the testing set and the predicted set is quite close, the third visualization at the top-right corner displays the same graph as the second visualization albeit with varying line thickness for the two sets. The fourth visualization in the bottom-left corner displays the first 100 values in both groups. Although the overlap between both groups may indicate that each predicted point matches an actual point, the visualization in this case shows that an exact matching for every point is not the case. In the following visualizations remaining, I expand from 100 points to 400 points in the fifth visualization and from 400 points to 900 points in the sixth visualization to indicate a closer merge of points as row indices increase.

Lag values are one of the few parameters that can be tweaked in our model. In the final model, a lag value of 30 was used; tweaking it to the extreme on both ends did little to improve our model as a whole. As a final method of evaluation, I computed the mean-squared error for my model which yielded a value of 0.001. The mean-squared error measures the average of the squares of the errors or deviations (i.e. the difference between the estimator and what is estimated). It measures the quality of an estimator, or in our case, the predicted set. Since values closer to zero indicate a better model, we can conclude that our model–with a mean-squared error of 0.001–is a good model.

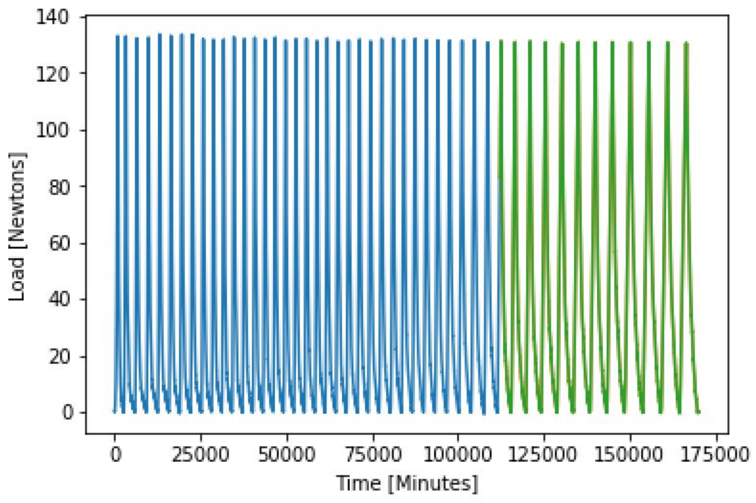
Predicting Next Cycles: A Persistence Model

The autoregression model does well to predict values concurrently on top of values in our actual test set. However, in regards to predicting next values beyond the values already in our dataset, we may want to explore other forecasting models. I will be using the persistence algorithm to implement one. According to Brownlee, the persistence algorithm uses values at a previous time step to predict the expected outcome at the next time step [4]. In order to illustrate the algorithm more clearly, I have included an example of the first few load values of our dataset and their predicted values at the following time step.

```
       L-1        L
0      NaN  0.06729
1  0.06729  0.07128
2  0.07128  0.05453
3  0.05453  0.00621
4  0.00621  0.00352
```

The column L contains the first few values in our load column pertaining to our dataset of 47 cycles. The column L-1 contains values that are one previous "time-step" ahead of L. In row 0, there are no previous values that occur before 0.06729. Therefore, the previous value is NaN (i.e. none). In row 1, the previous value before 0.07128 is 0.06729. Therefore, the 0.06729 is indicated as the previous value in the column, L-1. The column L-1 is what is used to predict the next rows in our dataset.

After delegating 66% of the dataset to be trained, predictions were made using the remaining test set of 34%. I have graphed the results below, where the blue lines indicate the actual data and the green lines indicate the predicted next rows:



We can intuitively conclude that the model is naive given that the predicted values are based on the L-1 values; however, a similar pattern to the actual data is certainly achieved as seen in the visualization above. The mean-squared error was 0.006, which is decent metric considering this naive approach.

In order to accomplish the task at hand of only predicting the next three cycles for our dataset of 47 cycles, examination of the prediction outputs as well as the dataset was required.

The last row of our dataset containing only the 47 cycles ended with these values:

Time: 16770.9
Load: 0.09558

In my algorithm above, my persistence model starts making predictions based on a random point in the dataset (i.e. the starting point for the test set, which includes 34% of the data). This random starting point happens to be 83.06224 for the load value, which is quite far off from 0.09558. In order to know exactly where each cycle begins and ends, I have to know where each minimum and maximum cycle begins and ends.

In using my generative function to determine the start and end points for each minimum and maximum within the outputted predictions, I received a misleading output:

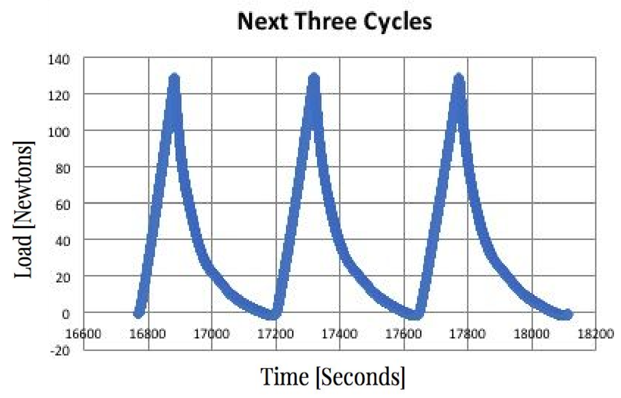
Cycle: 1
Minimum value: 83.06224
Maximum value: 99.87193

```
Cycle: 2
Minimum value: 100.04444000000001
Maximum value: 126.81127

Cycle: 3
Minimum value: 126.80696
Maximum value: 131.14229
```

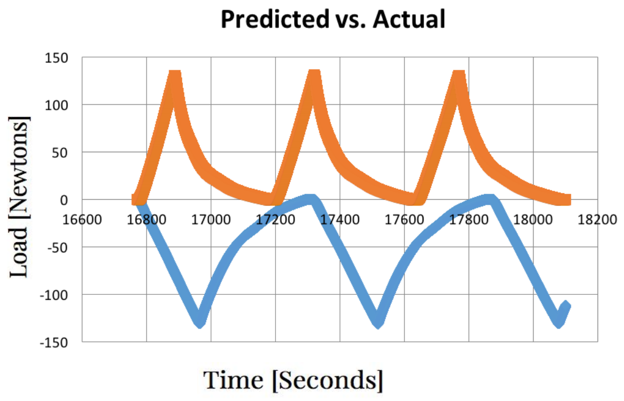
The values above do not indicate the correct points, from what we already know of our dataset. So, I decided to manually modify the returned predictions to indicate a start value closer to 0.09558 and not 83.06224. The first value immediately following 0.09558 was 0.97445, so I ran my algorithm on the prediction set starting from this value. Again, results were not ideal:

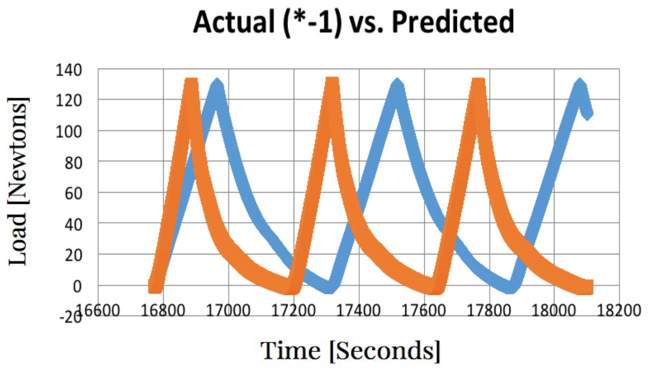
```
Cycle: 1
Minimum value: 0.97445
Maximum value: 1.23071

Cycle: 2
Minimum value: 1.21348
Maximum value: 6.9386399999999995

Cycle: 3
Minimum value: 6.93382
Maximum value: 6.93382
```

After much trial and error, I decided to manually obtain the three cycles by going through the data on my own. It took some time, but I was able to obtain the three cycles in this manner. The final graph for the next three predicted cycles is below, where the x-axis represents time and the y-axis represents load.



In comparing the predicted output against the remaining three cycles in our dataset containing 50 cycles overall, the results indicated a similar pattern as seen in the visualization below to the left. However, the cycles are inverted at most points along the horizontal axis. The blue line indicates the actual results from the dataset of the remaining three cycles, while the orange line indicates the predicted outputs for the aforementioned cycles. Similar to the above visualization, the x-axis represents time and the y-axis represents load. The graph below on the right shows the actual values multiplied by -1 against the predicted values, indicating a close–but not exact–overlap against the points pertaining to the actual three cycles.





The mean-squared error still remains at 0.006, despite the curve for the predicted results arriving a little earlier in time than the curve for the actual results.

Conclusions and Future Work

Whereas the problem in regards to obtaining cycle breakpoints has been addressed, there is still much work to be done in regards to predicting next values in our dataset so as to truly bring artificial, muscle movement to life rather than leave it as stationary. The models presented in this paper that fall under the category of time series forecasting yielded results that closely matched our actual data points with a minimal mean-squared error; however, next values predicted either overlap the test set (as seen using the autoregression model) or replace it entirely in the output (as seen using the persistence model).

In comparing the autoregression model to the persistence model, it is difficult to say which one exceeds the other in performance given a difference of only 0.005 in regards to the mean-squared error. In the future, we may or may not want to incorporate both models. Nonetheless, it is necessary to not only tweak these models so as to achieve a higher accuracy, but also look more closely into combining these models against other methods that seek to add values onto our dataset rather than on top of a test set or in replacement of a test set. Despite such improvements to be made, the work in this paper provides a starting point for further exploring data prediction by means of time series forecasting.

Bibliography

[1] Brockwell, Peter J., Davis, Richard A. "Examples of Time Series." *Time Series: Theories and Methods*. Springer. 1987. Retrieved: Dec-21-2017.

[2] Md. Rafiul Hassan., Baikunth Nath. "Stock Market Forecasting Using Hidden Markov Model: A New Approach." *Intelligent Systems Design and Applications*. 2005. Retrieved: Dec-21-2017.

[3] Chatfield, Chris. "Fractional Differencing and Long-Memory Models." *The Analysis of Time Series*. 2003. Taylor & Francis Group. Retrieved: Dec-21-2017.

[4] Brownlee, Jason, "How to Make Baseline Predictions for Time Series Forecasting with Python." *Machine Learning Mastery*. Retrieved: Dec-18-2017.

[5] *Provided by*: Miriyev, Aslan. "47_cycles.csv." Retrieved: Dec-18-2017.

[6] Lai, Tri. "Local Maxima and Minima." *University of Minnesota*. Retrieved: Dec-18-2017.

[7] "scipy.signal.argrelextrema." *SciPy.org*. Retrieved: Dec-18-2017.

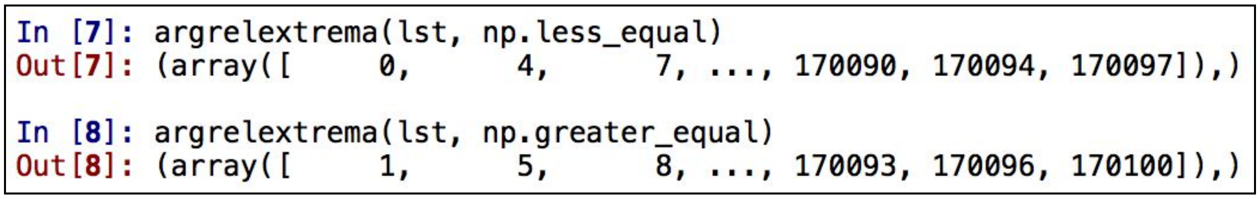
[8] *Provided by*: Miriyev, Aslan. "50_cycles.csv." Retrieved: Dec-18-2017.

[9] Brownlee, Jason. "What is Time Series Forecasting?" *Machine Learning Mastery*. Retrieved: Dec-18-2017.

Appendix A

1. Creative Machines Lab point people
    a. Hod Lipson - hod.lipson@columbia.edu
    b. Aslan Miriyev - aslan.miriyev@columbia.edu
2. Paper point person
    a. Navraj Narula - nnn2112@columbia.edu
3. Soft material for soft actuators
    a. This is a paper by Aslan Miriyev, Kenneth Stack, and Hod Lipson recently published in the *Nature Communications* journal. In order to get a better idea of the material used to design the muscle as well as learn about challenges in soft robotics, it would be good to review this paper for background knowledge–especially to understand the context behind the given dataset discussed in this paper.

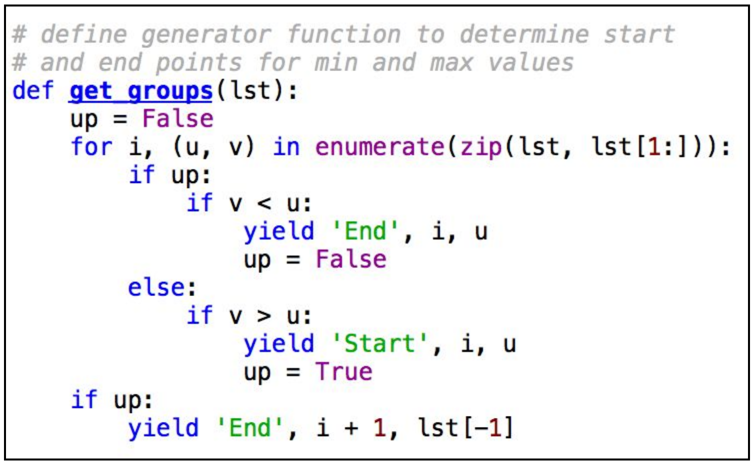
Appendix B



```
In [7]: argrelextrema(lst, np.less_equal)
Out[7]: (array([     0,      4,      7, ..., 170090, 170094, 170097]),)

In [8]: argrelextrema(lst, np.greater_equal)
Out[8]: (array([     1,      5,      8, ..., 170093, 170096, 170100]),)
```

The values returned represent array indices. Given such indices returned, we know that using NumPy's built-in extrema function is not a valid option for finding local minimums and maximums since they should be at least 130 values apart.

Appendix C



```python
# define generator function to determine start
# and end points for min and max values
def get_groups(lst):
    up = False
    for i, (u, v) in enumerate(zip(lst, lst[1:])):
        if up:
            if v < u:
                yield 'End', i, u
                up = False
        else:
            if v > u:
                yield 'Start', i, u
                up = True
    if up:
        yield 'End', i + 1, lst[-1]
```

My generator function to retrieve local minimum and maximum values is displayed in the image above. In our case, *lst* would be the list containing load values to be passed in as a parameter to the function.

Appendix D

```
Cycle: 1
Minimum value: 0.51372
Maximum value: 132.54321
Total time per cycle: 242.29998
Cooling time per cycle: 161.29998
Heating time per cycle: 80.9


Cycle: 2
Minimum value: 0.50146
Maximum value: 132.57836
Total time per cycle: 334.6
Cooling time per cycle: 262.80003
Heating time per cycle: 71.7

Cycle: 3
Minimum value: 0.50416
Maximum value: 131.58451
Total time per cycle: 289.2
Cooling time per cycle: 235.8
Heating time per cycle: 53.29994

Cycle: 4
Minimum value: 1.50691
Maximum value: 131.54046
Total time per cycle: 344.39994
Cooling time per cycle: 249.29988
Heating time per cycle: 95.0

Cycle: 5
Minimum value: 0.50848
Maximum value: 133.5142
Total time per cycle: 322.99988
Cooling time per cycle: 250.69988
Heating time per cycle: 72.2

Cycle: 6
Minimum value: 0.50238
Maximum value: 132.6618
Total time per cycle: 323.2
Cooling time per cycle: 251.2
Heating time per cycle: 71.89988

Cycle: 7
Minimum value: 0.52059
Maximum value: 132.69964
Total time per cycle: 313.00012
Cooling time per cycle: 252.90012
Heating time per cycle: 60.0

Cycle: 8
Minimum value: 0.5009
Maximum value: 132.63109
Total time per cycle: 311.3
Cooling time per cycle: 260.8
Heating time per cycle: 50.4

Cycle: 9
Minimum value: 0.52352
Maximum value: 131.53477
Total time per cycle: 274.0

Cooling time per cycle: 224.0
Heating time per cycle: 49.9

Cycle: 10
Minimum value: 0.50309
Maximum value: 131.55767
Total time per cycle: 291.3
Cooling time per cycle: 222.3
Heating time per cycle: 68.9

Cycle: 11
Minimum value: 0.52487
Maximum value: 131.57019
Total time per cycle: 292.60025
Cooling time per cycle: 222.80025
Heating time per cycle: 69.70025

Cycle: 12
Minimum value: 0.53089
Maximum value: 132.52823
Total time per cycle: 297.59975
Cooling time per cycle: 226.0
Heating time per cycle: 71.5

Cycle: 13
Minimum value: 0.50404
Maximum value: 131.56165
Total time per cycle: 280.0
Cooling time per cycle: 208.5
Heating time per cycle: 71.39975

Cycle: 14
Minimum value: 1.51977
Maximum value: 131.67785
Total time per cycle: 308.20025
Cooling time per cycle: 217.8
Heating time per cycle: 90.3

Cycle: 15
Minimum value: 0.52461
Maximum value: 131.58631
Total time per cycle: 293.5995
Cooling time per cycle: 220.5995
Heating time per cycle: 72.9

Cycle: 16
Minimum value: 0.50027
Maximum value: 131.61126
Total time per cycle: 296.7

Cooling time per cycle: 223.5
Heating time per cycle: 73.1

Cycle: 17
Minimum value: 0.50253
Maximum value: 130.64782
Total time per cycle: 295.4
Cooling time per cycle: 221.2
Heating time per cycle: 74.0995

Cycle: 18
Minimum value: 0.5016
Maximum value: 131.55186
Total time per cycle: 300.8
Cooling time per cycle: 225.4
Heating time per cycle: 75.3

Cycle: 19
Minimum value: 0.50355
Maximum value: 131.64112
Total time per cycle: 301.1
Cooling time per cycle: 223.8
Heating time per cycle: 77.1995

Cycle: 20
Minimum value: 0.50285
Maximum value: 130.7148
Total time per cycle: 278.5
Cooling time per cycle: 201.5995
Heating time per cycle: 76.8005

Cycle: 21
```

```
Minimum value: 1.50497
Maximum value: 131.60804
Total time per cycle: 339.3995
Cooling time per cycle: 247.3995
Heating time per cycle: 91.8995

Cycle: 22
Minimum value: 0.50203
Maximum value: 130.68189
Total time per cycle: 232.3995
Cooling time per cycle: 173.0
Heating time per cycle: 59.2995

Cycle: 23
Minimum value: 4.50012
Maximum value: 130.50085
Total time per cycle: 364.5

Cooling time per cycle: 230.7005
Heating time per cycle: 133.7

Cycle: 24
Minimum value: 0.51336
Maximum value: 131.53273
Total time per cycle: 310.5
Cooling time per cycle: 226.9
Heating time per cycle: 83.5

Cycle: 25
Minimum value: 0.52311
Maximum value: 130.55855
Total time per cycle: 316.3
Cooling time per cycle: 231.5
Heating time per cycle: 84.6995

Cycle: 26
Minimum value: 0.50607
Maximum value: 131.51486
Total time per cycle: 328.0
Cooling time per cycle: 241.3005
Heating time per cycle: 86.6

Cycle: 27
Minimum value: 0.52793
Maximum value: 131.57064
Total time per cycle: 311.7
Cooling time per cycle: 220.7
Heating time per cycle: 90.9

Cycle: 28
```

```
Minimum value: 0.50275
Maximum value: 131.50829
Total time per cycle: 300.599
Cooling time per cycle: 210.699
Heating time per cycle: 89.7995

Cycle: 29
Minimum value: 1.51489
Maximum value: 131.51
Total time per cycle: 334.899
Cooling time per cycle: 228.699
Heating time per cycle: 106.1

Cycle: 30
Minimum value: 0.50042
Maximum value: 131.50921
Total time per cycle: 326.5

Cooling time per cycle: 228.2
Heating time per cycle: 98.2

Cycle: 31
Minimum value: 0.51261
Maximum value: 130.71489
Total time per cycle: 323.2
Cooling time per cycle: 222.0
Heating time per cycle: 101.1

Cycle: 32
Minimum value: 1.53942
Maximum value: 130.54335
Total time per cycle: 359.301
Cooling time per cycle: 240.801
Heating time per cycle: 118.401

Cycle: 33
Minimum value: 0.51392
Maximum value: 130.5877
Total time per cycle: 342.8
Cooling time per cycle: 232.0
Heating time per cycle: 110.7

Cycle: 34
Minimum value: 1.5089
Maximum value: 131.5352
Total time per cycle: 400.6
Cooling time per cycle: 272.3
Heating time per cycle: 128.199

Cycle: 35
```

```
Minimum value: -0.49218
Maximum value: 130.50038
Total time per cycle: 361.301
Cooling time per cycle: 255.5
Heating time per cycle: 105.701

Cycle: 36
Minimum value: 0.51553
Maximum value: 130.51911
Total time per cycle: 398.3
Cooling time per cycle: 271.3
Heating time per cycle: 126.9

Cycle: 37
Minimum value: 0.51303
Maximum value: 130.56107
Total time per cycle: 422.199

Cooling time per cycle: 287.5
Heating time per cycle: 134.6

Cycle: 38
Minimum value: 0.5018
Maximum value: 130.53618
Total time per cycle: 441.0
Cooling time per cycle: 297.101
Heating time per cycle: 143.8

Cycle: 39
Minimum value: 0.5113
Maximum value: 130.53745
Total time per cycle: 459.9
Cooling time per cycle: 309.101
Heating time per cycle: 150.7

Cycle: 40
Minimum value: 0.51186
Maximum value: 129.57251
Total time per cycle: 466.4
Cooling time per cycle: 305.7
Heating time per cycle: 160.599

Cycle: 41
Minimum value: 0.51316
Maximum value: 130.57625
Total time per cycle: 486.099
Cooling time per cycle: 318.0
Heating time per cycle: 168.0

Cycle: 42
```

```
Minimum value: 0.50586
Maximum value: 130.56697
Total time per cycle: 510.2
Cooling time per cycle: 335.5
Heating time per cycle: 174.599

Cycle: 43
Minimum value: 0.51402
Maximum value: 130.55333
Total time per cycle: 477.2
Cooling time per cycle: 320.2
Heating time per cycle: 156.9

Cycle: 44
Minimum value: 1.50876
Maximum value: 130.53639
Total time per cycle: 521.201

Cooling time per cycle: 321.201
Heating time per cycle: 199.901

Cycle: 45
Minimum value: 1.50364
Maximum value: 130.5965
Total time per cycle: 540.301

Cooling time per cycle: 333.5
Heating time per cycle: 206.7

Cycle: 46
Minimum value: 0.5051
Maximum value: 130.5785
Total time per cycle: 562.2

Cooling time per cycle: 358.8
Heating time per cycle: 203.299

Cycle: 47
Minimum value: 0.50993
Maximum value: 129.5121
Total time per cycle: 531.501
Cooling time per cycle: 347.5
Heating time per cycle: 183.9
```