

Machine Learning Algorithms for Predicting Stellar Light Curves of RR Lyrae Variables

Andrew Ma

International School of Beijing, 100103 Shunyi District Beijing, China

Abstract. The question of why stars are dim and bright has been an important one within the physics realm for decades. This question is thoroughly answered within this paper, which aims to investigate and successfully predict the stellar light curve values, measured in magnitude, of the star, RR Lyrae. The prediction of RR Lyrae light curves are crucial and relevant to astrophysical research, as an accurate prediction allows insight to factors such as cosmic distances. To do so, various computation methods are employed, utilizing Python's Matplotlib for analysis and visualization, specifically consisting of basic mathematical regression models, machine learning algorithms, and neural networks. Hence, my study also aims to highlight the capabilities and limitations of each method, which provides insight into their performance. This will be furthermore measured with evaluation metrics, consisting of mean squared error and mean absolute error, respectively. Thus, the findings from this study will provide insight into the prediction of RR Lyrae's stellar light curve values, and emphasize the importance of selecting appropriate techniques for precise predictions. Moreover, the findings also illustrate the capabilities of various machine learning algorithms within different scenarios and contexts, shedding light into the inherent features and advantages of each one.

Keywords: Stellar Light Curves; Machine Learning Algorithms; regression models; neural networks.

1. Introduction

The variable I will be predicting within this paper is stellar light curves, which put in simpler terms is the graph that displays the brightness of a star over time. This data allows researchers to examine naturally occurring phenomenon of stars. Specifically, the phenomenon includes eclipsing binaries—when another star passes in front of the measured star, causing changes in observable light—, transiting exoplanets—when exoplanets orbiting around the observed star obscure the transmitted light—, and variable stars—stars that vary in brightness over time due to changes in size, temperature, or activity within the star itself—. However, within this paper, to produce accurate predictions and an analysis of the results, I must limit possible influential phenomenon to only one, as multiple phenomena would cause the data to be influenced by more than one measurable independent variable, as shown in literature 1 [1]. Hence, I have chosen RR Lyrae as target star for research, due to it being a variable star, changing in brightness over seasons. Moreover, RR Lyrae does not have any nearby stars nor exoplanets, which eliminate the phenomenon of eclipsing binaries and transiting exoplanets. Thus, with this star, I can effectively use multiple prediction and analysis models to perform the prediction and gain the desired results, which correlates with the methodologies discussed in literature 1 [1].

1.1. Star Background

RR Lyrae are a pivotal class of pulsating stars that are regarded as variable stars within the field of astronomy, as suggested by many sources including literature 1 [1]. The pulsating nature of this star indicates that its light magnitude (stellar light curve data) is solely reliant on itself. Typically, these stars are comprised of hydrogen and helium, and sometimes metal, which means that they self-pulsate, as driven by their helium ionization process. Hence, when predicting the stellar light curves of a star, RR Lyrae stars are a prime candidate, since their light curves have a consistent pattern, and are not usually affect by factors such as orbiting planets, orbiting stars, or stellar flares. Specifically, the light curves of this star have periodicity, in which the pulsating nature of the star creates almost a sine-wave periodicity in which the brightness fluctuates in a pattern day-today. Hence, within this research, this star is used.

1.2. Literature Review

Within the field of astrophysics and research relating to celestial bodies, predicting stellar light curves is something that has been crucial to understanding celestial phenomenon. Hence, research regarding this topic can be found ranging from the likes of NASA all the way to self-published authors. Within this existing research, researchers commonly utilize the following methods:

- **Fourier Analysis**

Description: Breaks down light curves into sinusoidal components.

Applications: Effective for periodic variations like those seen in variable stars.

Limitations: Less effective for irregular or complex light curves.

- **Machine Learning Techniques**

Neural Networks: Used to model complex patterns in light curves.

Support Vector Machines (SVM): Applied for classification of variable stars.

Advantages: Capable of handling large datasets and uncovering non-obvious patterns.

Challenges: Requires extensive training data and computational resources.

- **Autoregressive Models**

Description: Predict future points based on past values in the light curve.

Use Case: Suitable for time-series data with moderate complexity.

Drawbacks: Assumes a stable relationship over time; may struggle with dynamic systems.

- **Gaussian Processes**

Description: A probabilistic approach that models uncertainties in predictions.

Strengths: Provides confidence intervals and handles noise well.

Limitations: Computationally intensive with large datasets.

- **Wavelet Transforms**

Description: Analyzes light curves at multiple scales simultaneously.

Benefits: Captures both frequency and time information, useful for non-stationary signals.

Challenges: Complexity in interpretation and implementation.

1.2.1 Preexisting Literature

Given these precedent methods, the research would likely utilize multiple, to ensure the accuracy of the results, and provide a conclusion to which methods are the most effective. Next, an analysis of pre-existing research related to this field will also help provide precedents and context to the research and results. The research literature in this area explores various methodologies for analyzing stellar light curves. Literature 1 combines LSTM and Transformer frameworks to enhance the capture of temporal and global features in stellar rotation periods from Kepler light curves, outperforming classical methods like the Autocorrelation Function. Literature 2 introduces an unorthodox automated methodology for analyzing eclipsing binary light curves using a fast iterative pre-whitening procedure to extract sinusoids and measure orbital and stellar parameters. Literature 3 discusses a deep learning approach for automated classification of stellar light curves, employing a convolutional neural network to accurately identify different types of variable stars, including eclipsing binaries and pulsating variables. Literature 4 outlines different approaches to time series analysis of stellar light curves, highlighting the scientific applications of each method along with their pros and cons. Literature 5 provides techniques and suggestions to enhance the accuracy of machine learning in analyzing and predicting stellar light curves. Literature 6 examines the scientific applications of stellar light curves, exploring patterns in data and their implications for scientific phenomena, as well as different types of variable stars and influencing factors. Literature 7 is a report analyzing the stellar light curve of R Scuti from 1970 to 2017, offering in-depth scientific and data analysis. Literature 8 is a research-tutorial discussing Python methodologies for analyzing variable stars, detailing parameter adjustments and tuning. Literature 9 applies transfer learning to stellar light curve prediction and classification, with a focus on programming and applications. Lastly, Literature 10 presents a research paper that uses code and mathematics to predict stellar light curve values, similar to the other research discussed.

2. Data Collection



RR Lyrae.csv

```

2451111.506,7.9,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451111.521,7.9,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451111.553,7.8,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451111.577,7.8,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451112.517,7.7,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451112.53,7.7,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451113.496,8.2,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451113.548,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451113.562,8.2,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451114.512,8.2,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451114.526,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451114.539,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451114.554,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451114.67,8.2,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451126.49,8.2,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451126.51,8.0,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451126.52,8.0,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451129.517,7.5,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451129.536,7.4,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451129.555,7.5,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451132.493,7.6,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451134.514,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451134.528,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451135.651,8.0,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451135.663,8.0,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451141.495,8.1,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451142.51,8.0,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451142.524,7.8,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451142.539,7.6,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,
2451142.554,7.3,,,Vis.,SQR,NA,NA,,,Z,,,RR LYR,STD,,,

```

After plotting out the data points, it can be observed that NASA’s dataset has evident errors. Specifically, the dataset from NASA provided many extreme outliers—which were not physically possible, and were likely generated either from incorrect data or the codes’ failure to visualize properly—, observed by the extreme peaks within the visualized data:

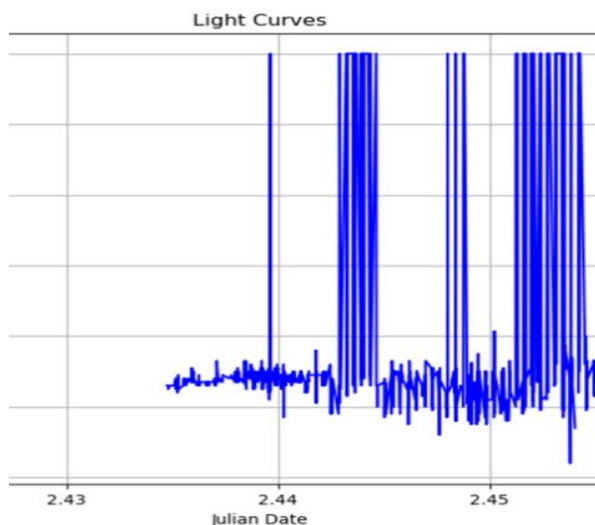


Fig. 2. Extreme outliers in Nasa dataset

Hence, I must eliminate the dataset from NASA, and use the one provided by the AAVSO, as extreme values and missing values would be detrimental to machine learning models such as the widely used Fourier transform, shown by the peaks in data within figure 2. Specifically, the outliers of these extreme values would essentially disable predictions that utilize regression to analyse trends, as these outlier values would cause regression lines to be skewed up and not accurately predict future data.

3. Data Preprocessing

Since I have finalized the dataset for analysis, a stage of preprocessing must be added between the raw data collection and the analysis—this is because there are likely factors that must be hindered or strengthened within the data to strengthen the accuracy of the data analysis—as demonstrated in literature 2 [2]. And as predicted, there is a major issue within the dataset that would hinder the analysis. Because the dataset consists of over one million instances, the high volume would still be virtually impossible to plot via matplotlib, as the values would be barely readable. In other words, the large volume of x data points (magnitude) versus the relatively lower volume of y data points (time in Julian date) would cause the data to be “horizontally compressed”, which essentially makes it so that all the data points seem to be vertically clustered as if they were a line. Moreover, the nature of the data would also mean the natural patterns within the data would be unobservable. The following can be observed explicitly in the raw data below:

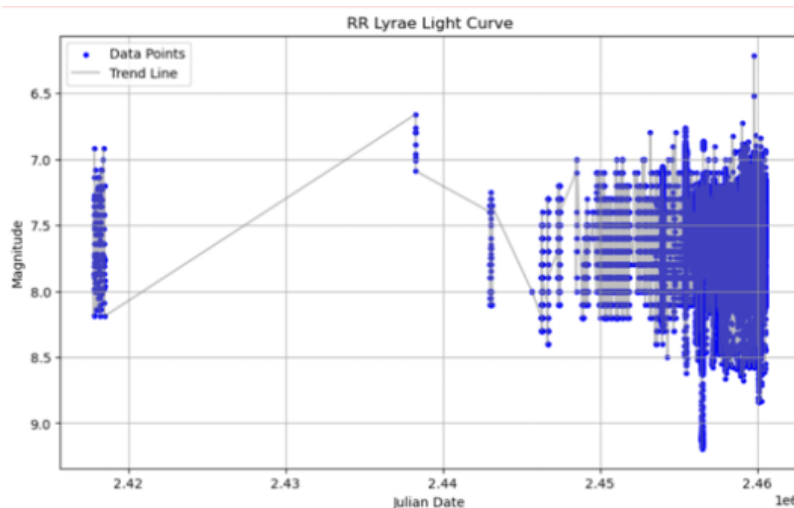


Fig. 3. Raw data graphed

As observed in figure 3 above, I can see that from around Julian Date 2.43 to 2.44, the entirety of the data is missing, which is likely due to missing instances within the dataset itself. However, this was not detrimental to the research, since RR Lyrae is a self-pulsating star, meaning that the time frame in which I analyzed its light curves would not affect the results of the research. Hence, to combat these issues, I selected a random subset of the data analysis. In other words, a smaller, more visible portion of the data would be selected for analysis. This would result in a more visible curvature in the nature of the plotted points, and a much better experience for observers. Not only would this result in a less condensed dataset, but it would also make sure that there was ample room between datasets.

4. Prediction methods

Next, with the processed data, ready for analysis, the following three categories of analysis was decided, with reference to literature 3's suggestions [3]:

1. Mathematical regression
 - a. Linear
 - b. Sine wave
2. Fourier transform machine learning
3. Pytorch neural network [3]

Specifically, the methods of analysis in which are utilized are inspired by previous literature in this area of research, specifically literature 3, and by implementing these contrasting methods, I hope to draw a secondary conclusion that highlights which methods draw the best prediction. To begin, I utilize mathematically consistent models in the form of simplistic functions—linear and sine—. From these models, I can aim to draw a fixed or concrete relation between the time (Julian) and magnitude of RR Lyrae's stellar light curves. Specifically, linear regression models will map the relationship in a straight-line, allowing us to see the general growth or decay (if there is) of the stellar light curves. Moreover, sine wave regression will allow us to model the oscillatory behavior of stellar light curves (if one does further research upon self-pulsating stars such as RR Lyrae, one will find that the stellar light waves generated will mimic oscillatory behavior within mathematics), allowing us to see the mathematical fluctuation between the independent and dependent variables. Next, I move on to a more complex method, using machine learning to better fit the cluster of data. Specifically, by utilizing a machine learning model, it can account for mathematical regression models' failure to "adjust" to a dataset, allowing us to see the frequency within a time series model. Last, I also implement flexible analysis in the form of neural networks, in which I aim to perfect the analysis by purely predicting future values from learning and training from given values, meaning that rather than predicting from correlation, I am predicting based on underlying patterns within the data.

5. Implementation

To perform analysis, the medium of python and its libraries (specifically matplotlib, scikit learn, and Pytorch) were used to visualize the data and analyze based off it. From this four codes were developed for each method of analysis [4], as shown by figures 4, 5, 6, and 7:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Data
jd = [
    2417815.4650, 2417815.5900, 2417816.4960, 2417823.3360, 2417823.4040,
    2417823.5030, 2417827.3790, 2417827.4110, 2417828.3700, 2417828.5530,
    2417829.3240, 2417829.4430, 2417830.4210, 2417830.4930, 2417831.3800,
    2417831.4860, 2417833.3990, 2417837.3380, 2417837.4850, 2417838.4650,
    2417839.3110, 2417839.5240, 2417842.3450, 2417842.4780, 2417843.3310,
    2417843.4890, 2417845.4000, 2417846.4320, 2417847.4710, 2417853.3220,
    2417853.4760, 2417859.4650, 2417861.3160, 2417863.4300, 2417874.3780,
    2417877.3080, 2417881.3140, 2417882.2940, 2417882.4250, 2417884.2940,
    2417888.2230, 2417888.3500, 2417891.2180, 2417902.2220, 2417902.3070,
    2417910.2270, 2417916.2300, 2417919.2300, 2417920.2240, 2417924.3090,
    2417926.2290, 2417927.2260, 2417934.2190, 2417937.2140, 2417938.1960,
    2417940.3070, 2417941.2950, 2417942.2920, 2417943.2220, 2417944.2330,
    2417945.2910, 2417946.2210, 2417951.2290, 2417952.2920, 2417953.2320,
    2417954.2260, 2417955.2230, 2417955.3290, 2417973.2980, 2418000.5280,
    2418006.5230, 2418023.4960, 2418026.5230, 2418028.4850, 2418032.4940,
    2418033.5110, 2418039.4860, 2418044.4620, 2418045.4280, 2418046.4430,
    2418053.4140, 2418053.5560, 2418054.4760, 2418056.4730, 2418056.5640,
    2418059.4670, 2418061.3900, 2418061.4830, 2418061.5670, 2418069.4080,
    2418071.4370, 2418071.5390, 2418072.3940, 2418075.3660, 2418075.4500,
    2418078.4810, 2418079.4190, 2418080.4110, 2418080.4730, 2418082.4670,
    2418082.6440, 2418084.5030, 2418089.4170, 2418089.4890, 2418090.3920,
    2418090.4720, 2418090.5590, 2418094.4760, 2418095.4860, 2418095.5620,
    2418096.4850, 2418096.5410, 2418098.4660, 2418103.4400, 2418104.4620,
    2418108.4650, 2418117.4470, 2418118.4680, 2418120.4620, 2418122.4620,
    2418123.4430, 2418124.4750, 2418125.4820, 2418134.4260, 2418138.3990,
    2418140.4730, 2418141.5050
]

magnitude = [
    7.89, 8.18, 7.36, 7.74, 7.53, 7.87, 7.72, 7.81, 7.42, 7.87,
    7.78, 7.36, 7.90, 8.19, 7.81, 7.87, 7.29, 7.50, 7.31, 7.64,
    7.87, 7.98, 7.87, 6.92, 7.98, 8.00, 7.36, 7.31, 7.87, 7.42,
    7.87, 7.14, 7.50, 7.27, 7.64, 7.57, 7.99, 7.48, 7.87, 7.93,
    7.93, 7.48, 7.08, 7.64, 7.87, 7.94, 7.42, 8.00, 7.41, 7.64,
    8.00, 7.76, 7.94, 7.36, 8.00, 7.81, 7.57, 7.42, 8.14, 7.94,
    7.76, 7.28, 8.14, 7.80, 7.72, 7.57, 7.87, 7.42, 7.87, 8.00,
    7.87, 7.42, 7.48, 7.28, 7.81, 7.74, 7.23, 7.72, 7.57, 7.64,
    7.42, 7.94, 7.98, 8.00, 7.31, 7.53, 7.76, 7.42, 7.57, 7.42,
    7.98, 7.98, 7.14, 7.64, 7.21, 8.14, 7.98, 7.76, 8.05, 7.14,
    7.98, 8.14, 7.64, 7.81, 7.98, 8.14, 7.87, 7.28, 7.64, 7.76,
    7.87, 7.87, 7.64, 7.80, 7.57, 7.87, 7.87, 7.31, 7.98, 7.76,
    7.64, 7.42, 7.35, 7.14, 7.87
]

# Ensure both lists are of equal length
jd = jd[:len(magnitude)]

# Create DataFrame
df = pd.DataFrame({"JD": jd, "Magnitude": magnitude})

# Feature Engineering
X = df[['JD']].values
y = df['Magnitude'].values

# Split the Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

# Make Predictions
y_pred = model.predict(X_test)

# Calculate Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Predict Future Values
future_jd = np.array([[2418150], [2418200], [2418250]]) # Example future dates
future_magnitudes = model.predict(future_jd)
print("Future Magnitudes:", future_magnitudes)

# Plot the Results
plt.figure(figsize=(10, 6))
plt.scatter(df['JD'], df['Magnitude'], color='blue', s=10, label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.scatter(future_jd, future_magnitudes, color='green', s=50, label='Future Predictions')
plt.gca().invert_yaxis()
plt.xlabel('Julian Date')
plt.ylabel('Magnitude')
plt.title('RR Lyrae Light Curve Prediction')
plt.legend()
plt.grid(True)
plt.show()
```

Fig. 4. Linear regression code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Data
jd = [
    2417815.4650, 2417815.5900, 2417816.4960, 2417823.3360, 2417823.4040,
    2417823.5030, 2417827.3790, 2417827.4110, 2417828.3700, 2417828.5530,
    2417829.3240, 2417829.4430, 2417830.4210, 2417830.4930, 2417831.3800,
    2417831.4860, 2417833.3990, 2417837.3380, 2417837.4850, 2417838.4650,
    2417839.3110, 2417839.5240, 2417842.3450, 2417842.4780, 2417843.3310,
    2417843.4890, 2417845.4000, 2417846.4320, 2417847.4710, 2417853.3220,
    2417853.4760, 2417859.4650, 2417861.3160, 2417863.4300, 2417874.3780,
    2417877.3880, 2417881.3140, 2417882.2940, 2417882.4250, 2417884.2940,
    2417888.2230, 2417888.3500, 2417891.2180, 2417902.2220, 2417902.3070,
    2417910.2270, 2417916.2300, 2417919.2300, 2417920.2240, 2417924.3090,
    2417926.2290, 2417927.2260, 2417934.2190, 2417937.2140, 2417938.1960,
    2417940.3870, 2417941.2950, 2417942.2920, 2417943.2220, 2417944.2330,
    2417945.2910, 2417946.2210, 2417951.2290, 2417952.2920, 2417953.2320,
    2417954.2260, 2417955.2230, 2417955.3290, 2417973.2980, 2418000.5280,
    2418006.5230, 2418023.4960, 2418026.5230, 2418028.4850, 2418032.4940,
    2418033.5110, 2418039.4860, 2418044.4620, 2418045.4280, 2418046.4430,
    2418053.4140, 2418053.5560, 2418054.4760, 2418056.4730, 2418056.5640,
    2418059.4670, 2418061.3900, 2418061.4830, 2418061.5670, 2418069.4080,
    2418071.4370, 2418071.5390, 2418072.3940, 2418075.3660, 2418075.4500,
    2418078.4810, 2418079.4190, 2418080.4110, 2418080.4730, 2418082.4670,
    2418082.6440, 2418084.5030, 2418089.4170, 2418089.4890, 2418090.3920,
    2418090.4720, 2418090.5590, 2418094.4760, 2418095.4860, 2418095.5620,
    2418096.4850, 2418096.5410, 2418098.4660, 2418103.4400, 2418104.4620,
    2418108.4650, 2418117.4470, 2418118.4680, 2418120.4620, 2418122.4620,
    2418123.4430, 2418124.4750, 2418125.4820, 2418134.4260, 2418138.3990,
    2418140.4730, 2418141.5050
]

magnitude = [
    7.89, 8.18, 7.36, 7.74, 7.53, 7.87, 7.72, 7.81, 7.42, 7.87,
    7.78, 7.36, 7.90, 8.19, 7.81, 7.87, 7.29, 7.50, 7.31, 7.64,
    7.87, 7.98, 7.87, 6.92, 7.98, 8.00, 7.36, 7.31, 7.87, 7.42,
    7.87, 7.14, 7.50, 7.27, 7.64, 7.57, 7.99, 7.48, 7.87, 7.93,
    7.93, 7.48, 7.08, 7.64, 7.87, 7.94, 7.42, 8.00, 7.41, 7.64,
    8.00, 7.76, 7.94, 7.36, 8.00, 7.81, 7.57, 7.42, 8.14, 7.94,
    7.76, 7.28, 8.14, 7.80, 7.72, 7.57, 7.87, 7.42, 7.87, 8.00,
    7.87, 7.42, 7.48, 7.28, 7.81, 7.74, 7.23, 7.72, 7.57, 7.64,
    7.42, 7.94, 7.98, 8.00, 7.31, 7.53, 7.76, 7.42, 7.57, 7.42,
    7.98, 7.98, 7.14, 7.64, 7.21, 8.14, 7.98, 7.76, 8.05, 7.14,
    7.98, 8.14, 7.64, 7.81, 7.98, 8.14, 7.87, 7.28, 7.64, 7.76,
    7.87, 7.87, 7.64, 7.80, 7.57, 7.87, 7.87, 7.31, 7.98, 7.76,
    7.64, 7.42, 7.35, 7.14, 7.87
]

# Ensure both lists are of equal length
jd = jd[:len(magnitude)]

# Define a sine wave function
def sine_wave(x, A, omega, phi, offset):
    return A * np.sin(omega * x + phi) + offset

# Fit the model
params, _ = curve_fit(sine_wave, jd, magnitude, p0=[0.5, 0.01, 0, 8])

# Predict future values
future_jd = np.array([2418150, 2418200, 2418250])
future_magnitudes = sine_wave(future_jd, *params)

# Print future predictions
print("Future Magnitudes:", future_magnitudes)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(jd, magnitude, color='blue', s=10, label='Actual')
jd_fine = np.linspace(min(jd), max(jd) + 100, 1000)
plt.plot(jd_fine, sine_wave(jd_fine, *params), color='red', linewidth=2, label='Sine Fit')
plt.scatter(future_jd, future_magnitudes, color='green', s=50, label='Future Predictions')
plt.gca().invert_yaxis()
plt.xlabel('Julian Date')
plt.ylabel('Magnitude')
plt.title('RR Lyrae Light Curve Prediction with Sine Wave')
plt.legend()
plt.grid(True)
plt.show()
```

Fig. 5. Sine wave regression code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq, ifft
from scipy.interpolate import interp1d
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Data
jd = [
    2417815.4650, 2417815.5900, 2417816.4960, 2417823.3360, 2417823.4040,
    2417823.5030, 2417827.3790, 2417827.4110, 2417828.3700, 2417828.5530,
    2417829.3240, 2417829.4430, 2417830.4210, 2417830.4930, 2417831.3800,
    2417831.4860, 2417833.3990, 2417837.3380, 2417837.4850, 2417838.4650,
    2417839.3110, 2417839.5240, 2417842.3450, 2417842.4780, 2417843.3310,
    2417843.4890, 2417845.4800, 2417846.4320, 2417847.4710, 2417853.3220,
    2417853.4760, 2417859.4650, 2417861.3160, 2417863.4300, 2417874.3780,
    2417877.3880, 2417881.3140, 2417882.2940, 2417882.4250, 2417884.2940,
    2417888.2230, 2417888.3500, 2417891.2180, 2417902.2220, 2417902.3070,
    2417910.2270, 2417916.2300, 2417919.2300, 2417920.2240, 2417924.3090,
    2417926.2290, 2417927.2260, 2417934.2190, 2417937.2140, 2417938.1960,
    2417940.3070, 2417941.2950, 2417942.2920, 2417943.2220, 2417944.2330,
    2417945.2910, 2417946.2210, 2417951.2290, 2417952.2920, 2417953.2320,
    2417954.2260, 2417955.2230, 2417955.3290, 2417973.2980, 2418000.5280,
    2418006.5230, 2418023.4960, 2418026.5230, 2418028.4850, 2418032.4940,
    2418033.5110, 2418039.4860, 2418044.4620, 2418045.4280, 2418046.4430,
    2418053.4340, 2418053.5560, 2418054.4760, 2418056.4730, 2418056.5640,
    2418059.4670, 2418061.3900, 2418061.4830, 2418061.5670, 2418069.4080,
    2418071.4370, 2418071.5390, 2418072.3940, 2418075.3660, 2418075.4590,
    2418078.4810, 2418079.4190, 2418080.4110, 2418080.4730, 2418082.4670,
    2418082.6440, 2418084.5030, 2418089.4170, 2418089.4890, 2418090.3920,
    2418090.4720, 2418090.5590, 2418094.4760, 2418095.4860, 2418095.5620,
    2418096.4850, 2418096.5410, 2418098.4660, 2418103.4400, 2418104.4620,
    2418108.4650, 2418117.4470, 2418118.4680, 2418120.4620, 2418122.4620,
    2418123.4430, 2418124.4750, 2418125.4820, 2418134.4260, 2418138.3990,
    2418140.4730, 2418141.5050
]

magnitude = [
    7.89, 8.18, 7.36, 7.74, 7.53, 7.87, 7.72, 7.81, 7.42, 7.87,
    7.78, 7.36, 7.90, 8.19, 7.81, 7.87, 7.29, 7.50, 7.31, 7.64,
    7.87, 7.98, 7.87, 6.92, 7.98, 8.00, 7.36, 7.31, 7.87, 7.42,
    7.87, 7.14, 7.50, 7.27, 7.64, 7.57, 7.99, 7.48, 7.87, 7.93,
    7.93, 7.48, 7.08, 7.64, 7.87, 7.94, 7.42, 8.00, 7.41, 7.64,
    8.00, 7.76, 7.94, 7.36, 8.00, 7.81, 7.57, 7.42, 8.14, 7.94,
    7.76, 7.28, 8.14, 7.80, 7.72, 7.57, 7.87, 7.42, 7.87, 8.00,
    7.87, 7.42, 7.48, 7.28, 7.81, 7.74, 7.23, 7.72, 7.57, 7.64,
    7.42, 7.94, 7.98, 8.00, 7.31, 7.53, 7.76, 7.42, 7.57, 7.42,
    7.98, 7.98, 7.14, 7.64, 7.21, 8.14, 7.98, 7.76, 8.05, 7.14,
    7.98, 8.14, 7.64, 7.81, 7.98, 8.14, 7.87, 7.28, 7.64, 7.76,
    7.87, 7.87, 7.64, 7.80, 7.57, 7.87, 7.87, 7.31, 7.98, 7.76,
    7.64, 7.42, 7.35, 7.14, 7.87
]

# Ensure both lists are of equal length
jd = jd[:len(magnitude)]

# Split data into training (80%) and test (20%) sets
split_index = int(len(jd) * 0.8)
jd_train = jd[:split_index]
magnitude_train = magnitude[:split_index]
jd_test = jd[split_index:]
magnitude_test = magnitude[split_index:]

# Normalize JD for better numerical stability
jd_offset = np.min(jd_train)
jd_train_normalized = jd_train - jd_offset
jd_test_normalized = jd_test - jd_offset

# Interpolate training data to create a continuous line
f_interp = interp1d(jd_train_normalized, magnitude_train, kind='linear')
jd_fine = np.linspace(min(jd_train_normalized), max(jd_train_normalized), 1000)
magnitude_fine = f_interp(jd_fine)

# Perform Fourier Transform on interpolated data
magnitude_fft = fft(magnitude_fine)
frequencies = fftfreq(len(magnitude_fine), d=np.mean(np.diff(jd_fine)))

# Adjust frequency selection for better alignment
peak_indices = np.argsort(np.abs(magnitude_fft))[-10:]
filtered_fft = np.zeros_like(magnitude_fft)
filtered_fft[peak_indices] = magnitude_fft[peak_indices]

# Inverse Fourier Transform to get the smoothed function
magnitude_smoothed_fine = ifft(filtered_fft).real

# Interpolate smoothed data back to original training data size
f_smooth = interp1d(jd_fine, magnitude_smoothed_fine, kind='linear')
magnitude_smoothed = f_smooth(jd_train_normalized)

# Predict future values by extending the timeline
jd_future = np.linspace(max(jd_train_normalized), max(jd_test_normalized) + len(jd_test_normalized))
future_magnitude = np.zeros_like(jd_future)

amp = np.abs(filtered_fft[peak_indices])
phase = np.angle(filtered_fft[peak_indices])

for a, p, freq in zip(amp, phase, frequencies[peak_indices]):
    future_magnitude += a * np.cos(2 * np.pi * freq * jd_future + p)

# Scale future_magnitude to match the original data range
min_mag, max_mag = min(magnitude), max(magnitude)
future_magnitude = (future_magnitude - np.min(future_magnitude)) / (np.max(future_magnitude) - np.min(future_magnitude))
future_magnitude = future_magnitude * (max_mag - min_mag) + min_mag

# Calculate errors
mse = mean_squared_error(magnitude_test, future_magnitude)
mae = mean_absolute_error(magnitude_test, future_magnitude)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(jd_train, magnitude_train, color='blue', linestyle='-', linewidth=1, label='Interpolated (Train)')
plt.plot(jd_train, magnitude_smoothed, color='red', linestyle='-', linewidth=2, label='Fourier Fit')
plt.plot(jd_test, magnitude_test, 'o', color='orange', label='Actual (Test)')
plt.plot(jd_test, future_magnitude, color='green', linestyle='--', linewidth=2, label='Prediction')
plt.gca().invert_yaxis()
plt.xlabel('Julian Date')
plt.ylabel('Magnitude')
plt.title('RR Lyrae Light Curve with Fourier Transform Prediction')
plt.legend()
plt.grid(True)
plt.show()
```

Fig. 6. Fourier analysis code [4]

```

import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler

# Data
jd = [
    2417815.4650, 2417815.5900, 2417816.4960, 2417823.3360, 2417823.4040,
    2417829.5030, 2417829.3790, 2417827.4110, 2417828.3700, 2417828.5530,
    2417829.4430, 2417830.4210, 2417830.4930, 2417831.3800,
    2417831.4800, 2417833.3900, 2417837.3300, 2417837.4850, 2417838.4650,
    2417839.3110, 2417839.5240, 2417842.3450, 2417842.4780, 2417843.3330,
    2417843.4890, 2417845.4000, 2417846.4320, 2417847.4730, 2417853.3220,
    2417853.4760, 2417859.4050, 2417861.3160, 2417863.4340, 2417874.3780,
    2417877.3080, 2417881.3140, 2417882.2940, 2417882.4250, 2417884.2940,
    2417888.2230, 2417888.3500, 2417891.2180, 2417892.2220, 2417892.3070,
    2417930.2270, 2417916.2300, 2417919.2300, 2417928.2240, 2417924.3090,
    2417926.2290, 2417927.2260, 2417934.2190, 2417937.2140, 2417938.1960,
    2417940.3070, 2417942.2950, 2417942.2920, 2417943.2330, 2417944.2330,
    2417945.2910, 2417946.2210, 2417951.2290, 2417952.2920, 2417953.2320,
    2417954.2260, 2417955.2230, 2417955.3290, 2417973.2980, 2418000.5280,
    2418002.5230, 2418023.4060, 2418026.5230, 2418028.4850, 2418032.4940,
    2418033.5110, 2418039.4860, 2418044.4620, 2418045.4280, 2418046.4430,
    2418053.4140, 2418053.5360, 2418054.4760, 2418056.4730, 2418056.5640,
    2418059.4670, 2418061.3900, 2418061.4830, 2418061.5670, 2418069.4080,
    2418073.4370, 2418071.5390, 2418072.3940, 2418075.3660, 2418075.4500,
    2418078.4810, 2418079.4190, 2418080.4110, 2418080.4730, 2418082.4670,
    2418082.4440, 2418084.5030, 2418089.4170, 2418089.4890, 2418090.3920,
    2418090.4720, 2418090.5390, 2418094.0760, 2418094.4860, 2418095.5620,
    2418096.4850, 2418096.5410, 2418098.4660, 2418103.4400, 2418104.4620,
    2418108.4650, 2418117.4470, 2418118.4600, 2418120.4620, 2418122.4620,
    2418123.4430, 2418124.4750, 2418125.4820, 2418134.4260, 2418138.3990,
    2418140.4730, 2418141.5050
]

magnitude = [
    7.89, 8.18, 7.36, 7.74, 7.53, 7.87, 7.72, 7.81, 7.42, 7.87,
    7.78, 7.36, 7.90, 8.19, 7.81, 7.87, 7.29, 7.50, 7.31, 7.64,
    7.87, 7.98, 7.67, 8.02, 7.98, 8.00, 7.36, 7.31, 7.07, 7.42,
    7.87, 7.14, 7.50, 7.27, 7.64, 7.57, 7.99, 7.40, 7.87, 7.93,
    7.93, 7.84, 7.84, 7.87, 7.94, 7.42, 8.00, 7.41, 7.64,
    8.00, 7.76, 7.94, 7.36, 8.00, 7.81, 7.57, 7.42, 8.14, 7.94,
    7.76, 7.28, 8.14, 7.80, 7.72, 7.57, 7.87, 7.42, 7.87, 8.00,
    7.87, 7.42, 7.48, 7.20, 7.81, 7.23, 7.72, 7.51, 7.84,
    7.42, 7.94, 7.98, 8.00, 7.31, 7.53, 7.76, 7.42, 7.57, 7.42,
    7.98, 7.94, 7.98, 7.76, 7.21, 7.24, 7.98, 7.76, 8.05, 7.14,
    7.98, 8.14, 7.64, 7.81, 7.98, 8.14, 7.87, 7.20, 7.64, 7.76,
    7.87, 7.87, 7.64, 7.80, 7.57, 7.87, 7.87, 7.31, 7.98, 7.76,
    7.64, 7.42, 7.35, 7.14, 7.87
]

# Ensure both lists are of equal length
jd = jd[:len(magnitude)]

# Convert Julian Date to phase
period = 0.5668 # Typical period for RR Lyrae stars, adjust if known
def jd_to_phase(jd):
    return ((np.array(jd) - jd[0]) % period) / period

phase = jd_to_phase(jd)

# Create input features
X = np.column_stack((phase, np.sin(2 * np.pi * phase), np.cos(2 * np.pi * phase)))

# Normalize data
scaler_X = MinMaxScaler(feature_range=(-1, 1))
scaler_Y = MinMaxScaler(feature_range=(-1, 1))
X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_Y.fit_transform(np.array(magnitude).reshape(-1, 1).flatten())

# Split data into training (80%) and test (20%) sets
split_index = int(len(X_scaled) * 0.8)
X_train = X_scaled[:split_index]
y_train = y_scaled[:split_index]
X_test = X_scaled[split_index:]
y_test = y_scaled[split_index:]

# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# Define the Improved neural network model
class ImprovedNN(nn.Module):
    def __init__(self):
        super(ImprovedNN, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout(x)
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Initialize the model, loss function, and optimizer
model = ImprovedNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
epochs = 500
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 100 == 0:
        print(f"Epoch {(epoch+1)/(epochs)}, Loss: {loss.item():.4f}")

# Evaluate the model
model.eval()
with torch.no_grad():
    predictions = model(X_test_tensor).flatten().numpy()

# Inverse scale the predictions
predictions = scaler_Y.inverse_transform(predictions.reshape(-1, 1).flatten())
y_test_original = scaler_Y.inverse_transform(y_test.reshape(-1, 1).flatten())

# Calculate errors
mse = mean_squared_error(y_test_original, predictions)
mae = mean_absolute_error(y_test_original, predictions)

print(f"Mean Squared Error: (mse)")
print(f"Mean Absolute Error: (mae)")

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(jd[split_index:], magnitude[split_index:], color='blue', linestyle='-', linewidth=1, label='Train')
plt.scatter(jd[split_index:], magnitude[split_index:], color='orange', label='Actual (Test)')
plt.scatter(jd[split_index:], predictions, color='green', marker='x', s=50, label='Prediction')
plt.gca().invert_yaxis()
plt.xlabel('Julian Date')
plt.ylabel('Magnitude')
plt.title('RR Lyrae Light Curve with Improved Neural Network Prediction')
plt.legend()
plt.grid(True)
plt.show()

# Plot phase-folded light curve
plt.figure(figsize=(12, 6))
plt.scatter(phase[split_index:], magnitude[split_index:], color='blue', label='Train')
plt.scatter(phase[split_index:], magnitude[split_index:], color='orange', label='Actual (Test)')
plt.scatter(phase[split_index:], predictions, color='green', marker='x', s=50, label='Prediction')
plt.gca().invert_yaxis()
plt.xlabel('Phase')
plt.ylabel('Magnitude')
plt.title('Phase-folded RR Lyrae Light Curve with Improved Neural Network Prediction')
plt.legend()
plt.grid(True)
plt.show()

```

Fig. 7. Pytorch neural network code [3]

Essentially, for each of these codes, I selected the same subset within the data—shown by the data within brackets in figures 4, 5, 6, and 7—and plotted them through matplotlib. From this, I connected each data point so that there was a “trend” that could be analyzed—each data point would be connected to the closest data point moving forwards in time—. And last in the process, I implemented various forms of analysis to give us the results. Specifically, the code writing process was based on prior knowledge within the field of python coding and its libraries, as well as the assistance of prior literature such as literature 2. Along the way, many issues were encountered with file handling and

getting the models to perform well, but this was combated through logically changing the code based on tips and information in literature 2.

6. Evaluation metrics

To assess the results, I essentially split the dataset into two sections: a training set and a testing set [5]. Specifically, the training set would be utilized to “train” the data, in which the models would be given free access to solve for correlation and try to imitate the pattern. Next, with the data and patterns concluded from the training set, the testing set will be the raw prediction from the model, and the actual values will be plotted out to visualize whether the prediction is accurate or not. However, just by purely using the naked eye to evaluate whether the data is accurate or not does not bring upon the magnitude of accuracy nor the actual accuracy of the data. Hence, here is where I introduce evaluation metrics.

Within this experiment, I utilized the evaluation metric of mean squared error (referred to MSE from now on) to evaluate whether the prediction is accurate and applicable or not [10]. Essentially MSE calculates the average of the squares of the differences between predicted and actual values. Which in this context, is essentially the data points in which the model predicts and the actual values of the light curves. The formula is:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^2 - \hat{y}^2) \quad (1)$$

Where y are the actual values $\hat{y}$ are the predicted values, and n is the number of observations. Hence, lower MSE indicates a better fit of the model to the data, as when the difference between the actual values and predicted values are almost zero, the accuracy is naturally better. Finally, now that I have the code and two forms of evaluation, one being the human observer and two being MSE, I can produce results.

7. Results

7.1. Linear Regression

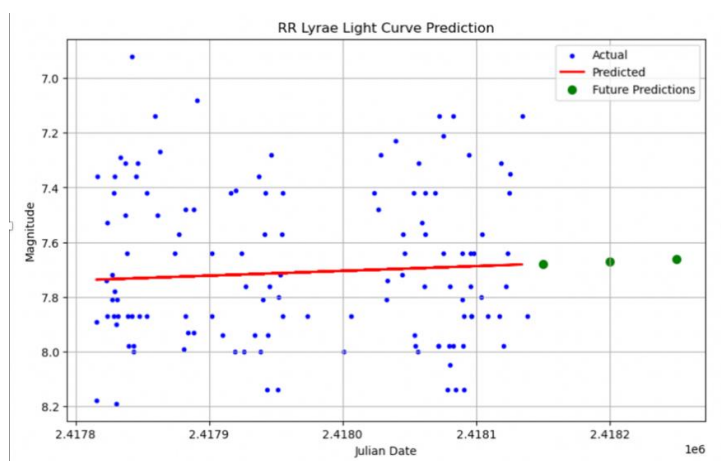


Fig. 8. Linear regression results

For the first method of mathematical regression, the linear regression model performed badly, as its prediction is solely reliant on the line of best fit generated from the model. Hence, it is unable to “fit” organically to the dataset and can only produce rigid predictions in the form of a straight line, as shown above in figure 8 in correlation to information in literature 5. This can be supported by its abysmal MSE score, which was almost 0.9. Moreover, this simplistic model has the issue of underfitting, in the sense that it is unable to capture underlying patterns and the periodicity of light magnitude (from previous context). If I use the naked eye to just view the predictions, they are off by a lot, as the predictions are just a straight line—shown in figure 8 in the green dots, which represent

future predicted values—. However, this method does provide some value in the sense that it shows the general trend within the brightness of RR Lyrae, and it shows that within this subset, the brightness of the star is generally increasing, since the line has a gradient that is greater than zero (observe figure 8).

7.2. Sine wave

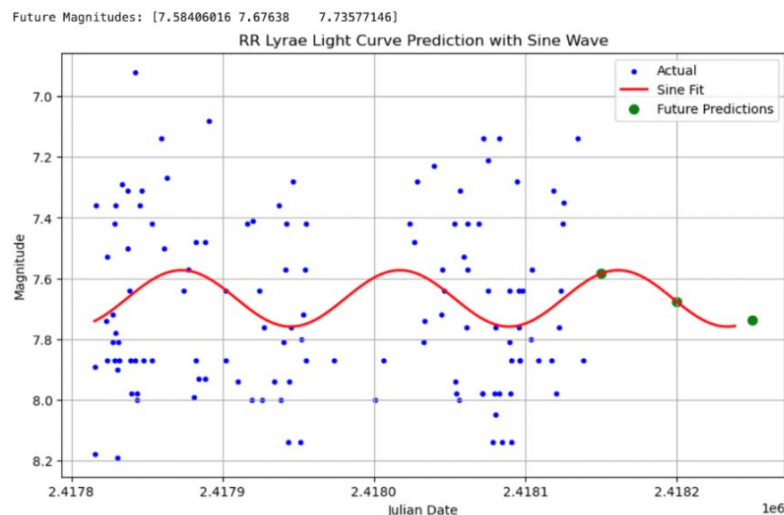


Fig. 9. Sine wave results

Since the linear regression performed poorly against this organic looking data that is in clusters, sine wave regression would likely perform better, as the sine wave matches the periodicity nature of pulsating stars, given the context within literature 2. However, I can still observe that prediction values are still based solely off the mathematical function, and not independent of the correlation, shown by figure 9, in which predicted values are plotted on the regression line. Hence, this still brought about rather poor results, yielding a MSE value of 0.7. However, I can observe a generally clearer trend within the data.

7.3. Fourier transform

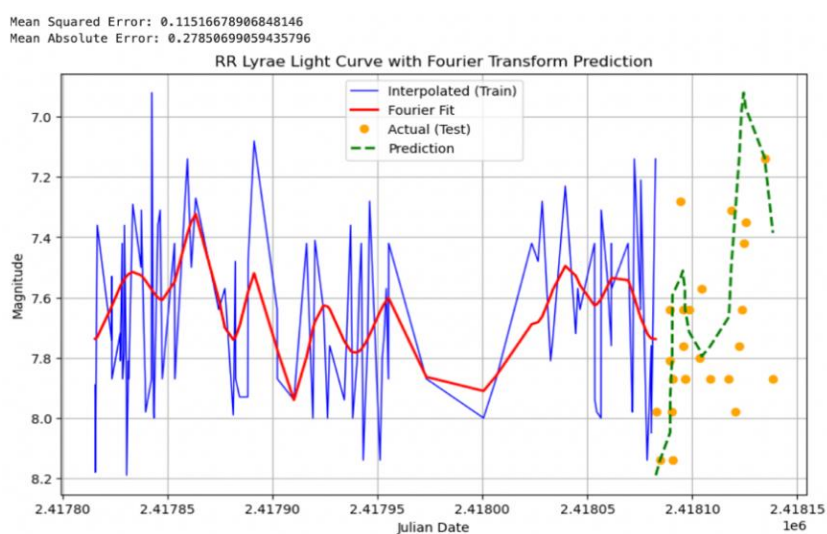


Fig. 10. Fourier transform results

Since mathematical regression models were unable to grasp the organic nature of the data points as suggested in literature 5, I can move on to a more flexible machine learning algorithm—Fourier transform. Since this model was very popular in prior literature, it is natural that it is the top candidate for a machine learning model in the research. To implement it, I first connected the datapoints to turn them into lines which are a time series—see the difference between figures 8-9 and 10—, which I

could not previously do for the mathematical regression models (regression requires data points and not lines). From this, I can train the machine learning model, in which it trains the red lines out (see figure 10). From the training, I can see that the machine learning model grasps the patterns well and is able to successfully predict future values to a quite acceptable accuracy, yielding a MSE value of 0.115 (rounded to three significant figures).

7.4. Pytorch neural network

```
Epoch [100/500], Loss: 0.1357
Epoch [200/500], Loss: 0.1255
Epoch [300/500], Loss: 0.1229
Epoch [400/500], Loss: 0.1091
Epoch [500/500], Loss: 0.1078
Mean Squared Error: 0.11913341380804368
Mean Absolute Error: 0.29510792312622086
```

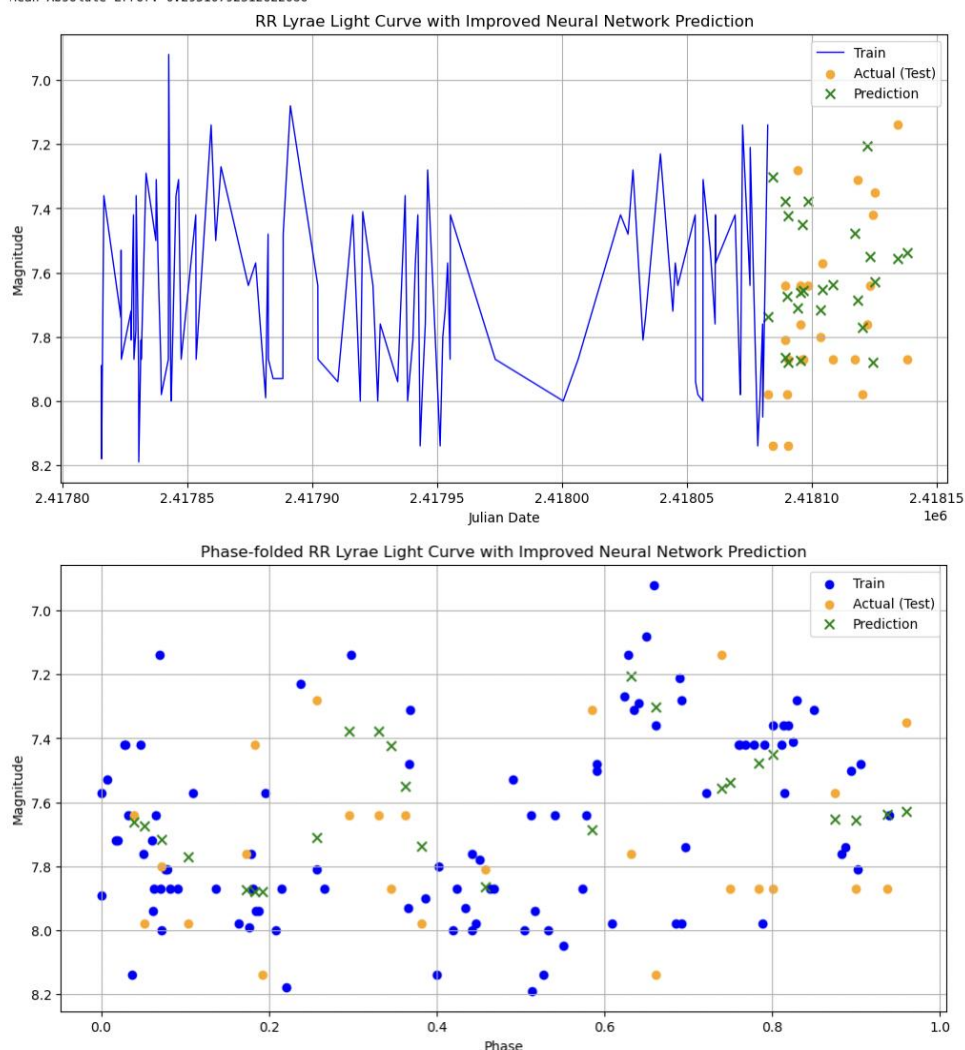


Fig. 11. Pytorch neural network results (500 epochs)

From the code, a simplistic Pytorch neural network is used to predict future data points, rather than a trendline (see figures 10, 9, and 8) that the other models provided. Since this model is a neural network, this would mean that setting parameters such as epochs is necessary to yield higher accuracy. From figure 11, the neural network prediction results with 500 epochs shows to have a MSE value of around 0.119, which is slightly worse than the Fourier analysis method before. Hence, to increase the accuracy, I can adjust the number of epochs to increase the accuracy of the method, in which information is provided in literature 5.

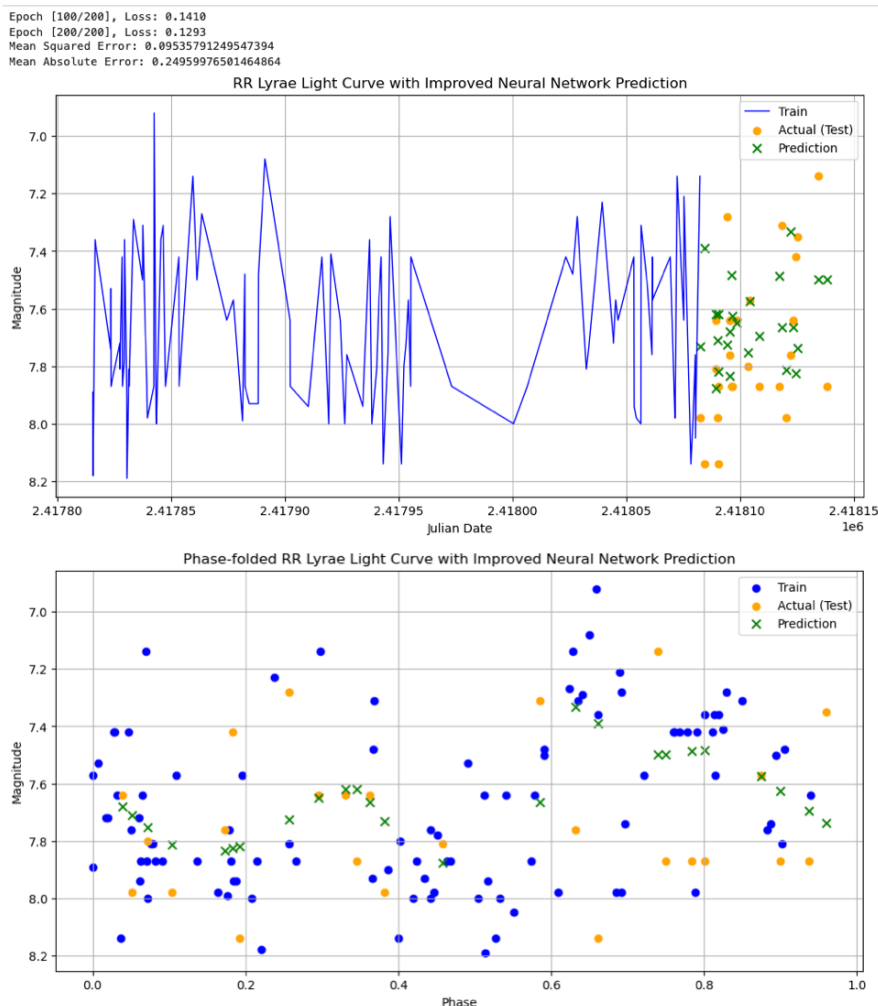


Fig. 12. Pytorch neural network results (200 epochs)

After experimenting and adjusting the epoch amount given the suggestions in literature 5, it seemed that 200 epochs yielded the best results, as shown in the MSE of 0.0953 (See figure 12). Originally, it was believed that having a higher epoch would yield the best results, as increasing the amount of testing/training sets would likely grant more data for predictions. However, increasing the epoch rate seemed to have a positive proportionality to the MSE, as an epoch of 800 yielded a MSE of almost 0.2 which correlates to data in literature 8.

8. Findings and Conclusion

Hence, after seeing all the methods used, I see that the method that brings upon the best data and prediction is the PyTorch neural network. However, unlike Fourier Transform, Pytorch does not yield time series data, in the sense that its prediction does not show trend nor a “path” that the data is to follow. Hence, when I am analyzing the “curve” of the light magnitude, methods such as Fourier transform that bring us predictions in the form of continuous “line” data, which will allow the analysis of the light curves itself. Still, in terms of accuracy, Pytorch does bring the best accuracy, yielding a MSE of 0.0953, allowing more accurate and reliable predictions. This is likely because unlike Fourier analysis, which relies on sinusoidal functions, neural networks offer greater flexibility and can adapt to various light curve types without specific assumptions, quoting literature 5. Moreover, Pytorch neural networks show better performance and robustness when it comes noise and fluctuations within data, hence explaining the better performance when it comes to data noise in stellar light curve data, as shown by the predictions in figures 11 and 12.

Interestingly, rather than being completely useless, mathematical regression models such as the linear regression model can be very useful for certain contexts in accordance to literature 5. For

example, the linear regression model can capture the overall trend within the data; showing whether the data values are growing or decreasing (figure 8). Which, in the subset of data chosen, the linear regression model suggests a general growth in the data, meaning that despite the pulsations within the star, the overall brightness of the star is still increasing. Hence, regression models that analyze the dataset can act as indicators of growth or decay. Hence, in conclusion, from the research, I can conclude that when it comes to accuracy itself:

PyTorch > Fourier > Mathematical Regression

However, from the findings discussed above, it can also be added that each method has its benefits under a given context, see more in literature 2. Thus, from this experiment, I can successfully predict stellar light curves of RR Lyrae to a certain accuracy. However, despite the successful results, there still exist limitations within this research. Some include the randomly selected subset of data, which might not accurately represent the true nature of the RR Lyrae's light curve data. These issues can be fixed by taking inspiration from prior successful and more validated literature. However, the results of this paper can shed light into the following topics, which are also discussed in literature 8:

- Pulsation Period: The regularity of the light curve's peaks and troughs indicates the star's pulsation period, which is related to its internal structure and stability.
- Star's Size and Mass: The period-luminosity relationship allows astronomers to infer the star's size and mass, as larger stars tend to have longer pulsation periods.
- Composition and Temperature: Changes in brightness can indicate shifts in temperature and composition, revealing the star's chemical makeup and evolutionary stage.
- Distance Measurement: Pulsating stars, like Cepheids, serve as standard candles for measuring cosmic distances due to their predictable brightness changes.
- Stellar Evolution: Light curves offer insights into the star's life cycle, helping astronomers understand how stars evolve over time.

References

- [1] Light Curves - Introduction. Nasa.gov. 2020. Available from: <https://imagine.gsfc.nasa.gov/science/toolbox/timing1.html>
- [2] Kalae MJ, Hasanzadeh A. A time series analysis of light curve of R Scuti star from 1970 to 2017. New Astron. 2019 Jul; 70:57-63. doi: 10.1016/j.newast.2019.02.007.
- [3] Vaughan S. Time series analysis in astronomy. [Internet]. [cited 2024 Aug 31]. Available from: <https://www.ma.imperial.ac.uk/~nsjones/TalkSlides/VaughnSlides.pdf>
- [4] Wang C, Bai Y, Han H, Yang H, Liu J. Transfer Learning Applied to Stellar Light Curve Classification. arXiv.org. 2023. Available from: <https://arxiv.org/abs/2305.13745>
- [5] Pan JS, Ting YS, Yu J. Astroconformer: The prospects of analysing stellar light curves with transformer-based deep learning models. Mon Not R Astron Soc. 2024 Jan;528(4):5890-5903. doi: 10.1093/mnras/stae068.
- [6] Automated eccentricity measurement from raw eclipsing binary light curves with intrinsic variability. Arxiv.org. 2024. Available from: <https://arxiv.org/html/2402.06084v1>
- [7] Kalae MJ, Hasanzadeh A. A time series analysis of light curve of R Scuti star from 1970 to 2017. New Astron. 2019 Jul; 70:57-63. doi: 10.1016/j.newast.2019.02.007.
- [8] Serre T. Predicting Variable Star Light Curves. In: Springer eBooks. 1993 Jan. p. 145-7. doi: 10.1007/978-94-011-1062-4_17
- [9] Hinnert TA, Tat K, Thorp R. Machine Learning Techniques for Stellar Light Curve Classification. Astron J. 2018 Jun;156(1):7. doi: 10.3847/1538-3881/aac16d.
- [10] Xavier Jr A. Using Python to analyze astronomy data: TESS exoplanet data. Medium. 2023 Oct 6. Available from: <https://xavnet2.medium.com/using-python-to-analyze-astronomy-data-tess-exoplanet-search-b858e23dfa49>
- [11] New Rotation Period Measurements for Kepler Stars Using Deep Learning: The 100K Sample. Arxiv.org. 2014. Available from: <https://arxiv.org/html/2407.06858v1>