

Python Final Project for analysis of Light Curve

Lingyu Xia
ID Number: EJ7410069
(Dated: May 5, 2024)

I. CODE-MAIN.PY

Listing 1: main.py code

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.stats import norm
5  from statsmodels.graphics.tsaplots import plot_acf
6  from statsmodels.tsa.seasonal import seasonal_decompose
7  from scipy.interpolate import interp1d
8  import logging
9
10
11  ##### BASE CLASS #####
12
13  class TimeSeriesAnalyzer:
14  def __init__(self, filename, energy_range, title):
15      self.filename = filename
16      self.energy_range = energy_range
17      self.title = title
18      self.data = None
19      self.logger = logging.getLogger(__name__)
20
21  def read_data(self):
22      """Read the data from a text file into a pandas DataFrame."""
23      try:
24          self.data = pd.read_csv(self.filename, sep='\t', skiprows=4, names=['Tstart (s)', 'Counts/s'])
25          self.logger.info("Data loaded successfully.")
26      except FileNotFoundError:
27          self.logger.error("File not found.")
28      except Exception as e:
29          self.logger.error(f"Error loading data: {e}")
30
31  def clean_data(self):
32      """Clean the data by handling missing values, outliers, and adjusting data types.
33      """
34      pass # Placeholder for data cleaning
35
36  def display_data(self):
37      """Display the data using various types of plots."""
38      if self.data is not None:
39          # Time series plot
40          plt.figure(figsize=(10, 6))
41          plt.plot(self.data['Tstart (s)'], self.data['Counts/s'], label='Time Series')
42          plt.xlabel('Time (s)')
43          plt.ylabel('Counts/s')
44          plt.title('Time Series Plot')
45          plt.legend()
46          plt.grid(True)
47          plt.show()
```

```

48         # Scatter plot
49         plt.figure(figsize=(8, 6))
50         plt.scatter(self.data['Tstart (s)'], self.data['Counts/s'], marker='.', color=
           'blue', label='Scatter Plot')
51         plt.xlabel('Time (s)')
52         plt.ylabel('Counts/s')
53         plt.title('Scatter Plot')
54         plt.legend()
55         plt.grid(True)
56         plt.show()
57
58         # Histogram
59
60         # Histogram with fitted normal distribution line
61         plt.figure(figsize=(8, 6))
62         plt.hist(self.data['Counts/s'], bins=20, color='green', alpha=0.7, density=
           True, label='Histogram')
63
64         # Fit a normal distribution to the data
65         mu, sigma = self.data['Counts/s'].mean(), self.data['Counts/s'].std()
66         xmin, xmax = plt.xlim()
67         x = np.linspace(xmin, xmax, 100)
68         p = norm.pdf(x, mu, sigma)
69         plt.plot(x, p, 'k', linewidth=2, label='Fitted Normal Distribution')
70
71         plt.xlabel('Counts/s')
72         plt.ylabel('Frequency')
73         plt.title('Histogram with Fitted Normal Distribution')
74         plt.legend()
75         plt.grid(True)
76         plt.show()
77
78         self.logger.info("Data displayed.")
79     else:
80         self.logger.warning("No data to display.")
81
82     def apply_operations(self):
83         """Apply mathematical operations or perform statistical analyses."""
84         pass # Placeholder for operations or analyses
85
86
87
88     ##### SUB CLASS #####
89
90     class DetrendingAnalyzer(TimeSeriesAnalyzer):
91         def __init__(self, filename, energy_range, title):
92             super().__init__(filename, energy_range, title)
93
94         def detrend_rescale(self):
95             """Detrend the data by subtracting the mean and rescale it to fit within the
           interval [-1, 1]."""
96             if self.data is not None:
97                 try:
98                     self.data['Counts/s'] -= self.data['Counts/s'].mean()
99                     self.data['Counts/s'] /= max(abs(self.data['Counts/s']))
100                     self.logger.info("Data detrended and rescaled.")
101                 except Exception as e:
102                     self.logger.error(f"Error detrending data: {e}")
103             else:
104                 self.logger.warning("No data to detrend.")
105
106         def plot_detrended_time_series(self):

```

```

107     """Plot the detrended time series to visually inspect the removal of trends."""
108     if self.data is not None:
109         plt.figure(figsize=(10, 6))
110         plt.plot(self.data['Tstart (s)'], self.data['Counts/s'], label='Detrended
111                  Counts/s')
112         plt.xlabel('Time (s)')
113         plt.ylabel('Detrended Counts/s')
114         plt.title('Detrended Time Series')
115         plt.legend()
116         plt.grid(True)
117         plt.show()
118         self.logger.info("Detrended time series plotted.")
119     else:
120         self.logger.warning("No data to plot.")
121
122     def plot_acf_detrended(self):
123         """Calculate and plot the autocorrelation function (ACF) to assess stationarity.
124         """
125         if self.data is not None:
126             try:
127                 plot_acf(self.data['Counts/s'], lags=50)
128                 plt.title('Autocorrelation Function (ACF) of Detrended Series')
129                 plt.xlabel('Lag')
130                 plt.ylabel('ACF')
131                 plt.show()
132                 self.logger.info("Autocorrelation function of detrended series plotted.")
133             except Exception as e:
134                 self.logger.error(f"Error plotting autocorrelation function: {e}")
135         else:
136             self.logger.warning("No data to plot.")
137
138     def time_series_decomposition(self):
139         """Perform time series decomposition to separate trend, seasonal, and residual
140         components."""
141         if self.data is not None:
142             try:
143                 result = seasonal_decompose(self.data['Counts/s'], model='additive',
144                                             period=100)
145                 result.plot()
146                 plt.suptitle('Time Series Decomposition')
147                 plt.show()
148                 self.logger.info("Time series decomposition performed.")
149             except Exception as e:
150                 self.logger.error(f"Error performing time series decomposition: {e}")
151         else:
152             self.logger.warning("No data to decompose.")
153
154     class InterpolationAnalyzer(TimeSeriesAnalyzer):
155         def __init__(self, filename, energy_range, title):
156             super().__init__(filename, energy_range, title)
157
158         def interpolate_data(self, method='linear'):
159             """Perform interpolation on the time series data."""
160             if self.data is not None:
161                 try:
162                     # Define interpolation function
163                     interp_func = interp1d(self.data['Tstart (s)'], self.data['Counts/s'],
164                                             kind=method)
165                     # Generate interpolated data
166                     interpolated_counts = interp_func(self.data['Tstart (s)'])
167                     # Update DataFrame with interpolated values
168                     self.data['Interpolated Counts/s'] = interpolated_counts

```

```

164         self.logger.info(f"Data interpolated using {method} method.")
165     except Exception as e:
166         self.logger.error(f"Error interpolating data: {e}")
167     else:
168         self.logger.warning("No data to interpolate.")
169
170 def plot_interpolated_time_series(self):
171     """Plot the original and interpolated time series."""
172     if self.data is not None:
173         plt.figure(figsize=(10, 6))
174         plt.plot(self.data['Tstart (s)'], self.data['Counts/s'], label='Original Time
175                 Series', color='blue')
176         plt.plot(self.data['Tstart (s)'], self.data['Interpolated Counts/s'], label='
177                 Interpolated Time Series', color='red', linestyle='--')
178         plt.xlabel('Time (s)')
179         plt.ylabel('Counts/s')
180         plt.title('Original vs Interpolated Time Series')
181         plt.legend()
182         plt.grid(True)
183         plt.show()
184         self.logger.info("Interpolated time series plotted.")
185     else:
186         self.logger.warning("No data to plot.")
187
188 class CumulativeSummationAnalyzer(TimeSeriesAnalyzer):
189     def __init__(self, filename, energy_range, title):
190         super().__init__(filename, energy_range, title)
191
192     def calculate_cumulative_sum(self):
193         """Calculate the cumulative sum of the counts."""
194         if self.data is not None:
195             try:
196                 self.data['Cumulative Sum'] = self.data['Counts/s'].cumsum()
197                 self.logger.info("Cumulative sum calculated.")
198             except Exception as e:
199                 self.logger.error(f"Error calculating cumulative sum: {e}")
200         else:
201             self.logger.warning("No data to calculate cumulative sum.")
202
203     def plot_cumulative_sum(self):
204         """Plot the cumulative sum."""
205         if self.data is not None:
206             plt.figure(figsize=(10, 6))
207             plt.plot(self.data['Tstart (s)'], self.data['Cumulative Sum'], label='
208                     Cumulative Sum', color='green')
209             plt.xlabel('Time (s)')
210             plt.ylabel('Cumulative Sum')
211             plt.title('Cumulative Sum Plot')
212             plt.legend()
213             plt.grid(True)
214             plt.show()
215             self.logger.info("Cumulative sum plot generated.")
216         else:
217             self.logger.warning("No data to plot.")
218
219     def find_t90(self):
220         """Find the T90 duration for the GRB event."""
221         if self.data is not None:
222             try:
223                 # Sort the data by time
224                 sorted_data = self.data.sort_values(by='Tstart (s)')

```

```

223         # Calculate cumulative sum
224         sorted_data['Cumulative Sum'] = sorted_data['Counts/s'].cumsum()
225
226         # Calculate total counts
227         total_counts = sorted_data['Counts/s'].sum()
228
229         # Find the index when cumulative sum reaches 5% and 95% of total counts
230         start_index = (sorted_data['Cumulative Sum'].cumsum() >= 0.05 *
231                        total_counts).idxmax()
232         end_index = (sorted_data['Cumulative Sum'].cumsum() >= 0.95 * total_counts
233                     ).idxmax()
234
235         # Get the times corresponding to the initial and final indices
236         initial_time = sorted_data.loc[start_index, 'Tstart (s)']
237         final_time = sorted_data.loc[end_index, 'Tstart (s)']
238
239         self.logger.info(f"T_90: {initial_time} to {final_time} seconds")
240         return initial_time, final_time
241     except Exception as e:
242         self.logger.error(f"Error finding T_90: {e}")
243     else:
244         self.logger.warning("No data to find T_90.")

```

Listing 2: presentation.py code

```

1
2     import main
3 import logging
4
5     # System MacOS Python Vesion: 3.9.6
6     # matplotlib          3.7.0
7     # matplotlib-inline   0.1.6
8     # numpy                1.23.4
9     # pandas               2.2.2
10    # scipy                 1.10.0
11    # statsmodels           0.14.2
12
13    # Example usage:
14    filename = "data.txt"
15    energy_range = "50-300 KeV"
16    title = "Light Curve for Fermi Event BN081224887"
17
18    # Set up logging
19    logging.basicConfig(level=logging.INFO)
20
21    # Create instances of subclasses
22    detrending_analyzer = main.DetrendingAnalyzer(filename, energy_range, title)
23    detrending_analyzer.read_data()
24
25    detrending_analyzer.detreng_rescale()
26
27    detrending_analyzer.plot_detrended_time_series()
28
29    detrending_analyzer.plot_acf_detrended()
30
31    detrending_analyzer.time_series_decomposition()
32
33    detrending_analyzer.display_data()
34

```

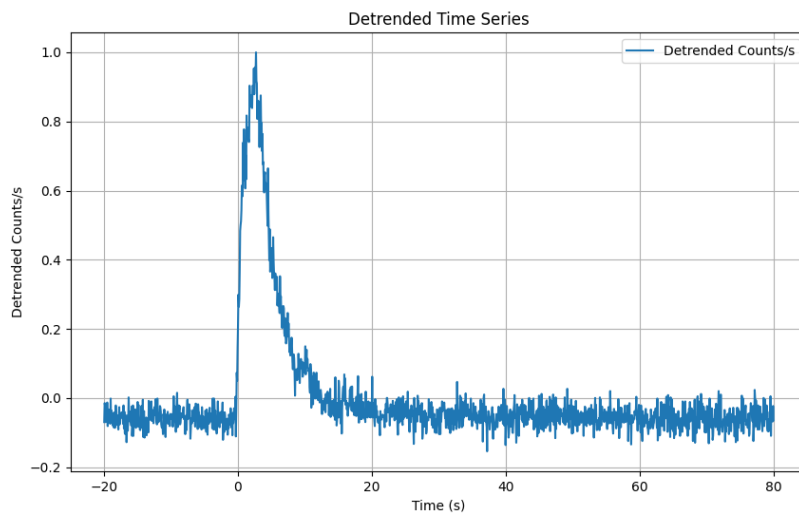
```

35     # Create instances of the InterpolationAnalyzer and CumulativeSummationAnalyzer
        subclasses
36     interpolation_analyzer = main.InterpolationAnalyzer(filename, energy_range, title)
37     cumulative_summation_analyzer = main.CumulativeSummationAnalyzer(filename,
        energy_range, title)
38
39     # Read data
40     interpolation_analyzer.read_data()
41     cumulative_summation_analyzer.read_data()
42
43     # Interpolate data using linear interpolation
44     interpolation_analyzer.interpolate_data(method='linear')
45
46     # Calculate cumulative sum
47     cumulative_summation_analyzer.calculate_cumulative_sum()
48
49     # Plot interpolated time series
50     interpolation_analyzer.plot_interpolated_time_series()
51
52     # Plot cumulative sum
53     cumulative_summation_analyzer.plot_cumulative_sum()

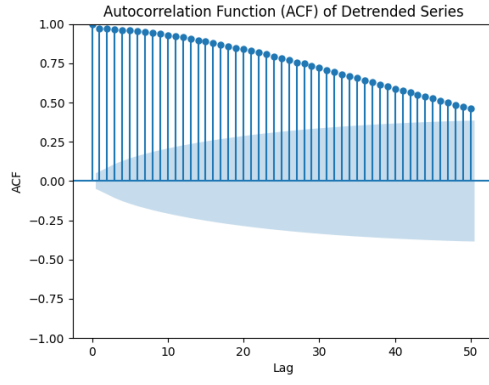
```

II. RESULT

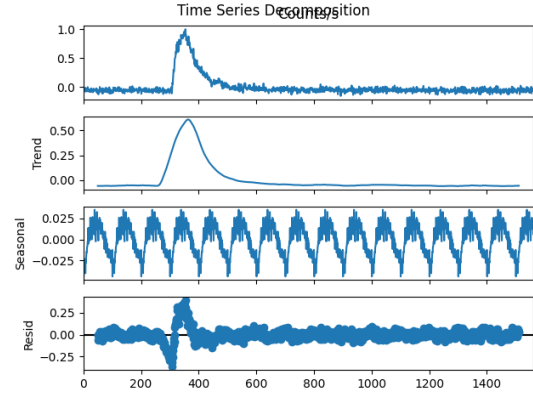
We firstly plot the intensity with time series for light curve.



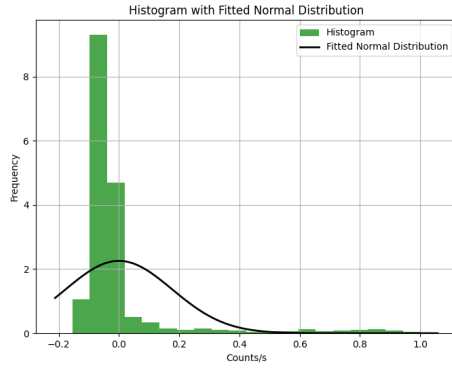
Then we can do the following analysis for this light curve.



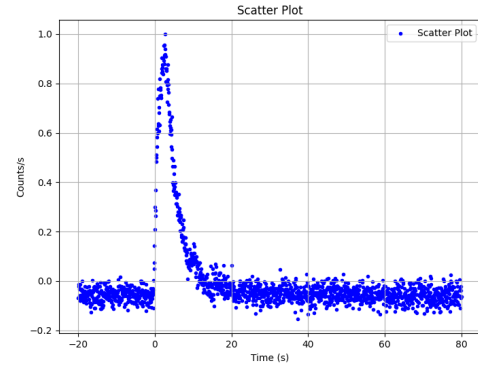
(a) Diagram for ACF



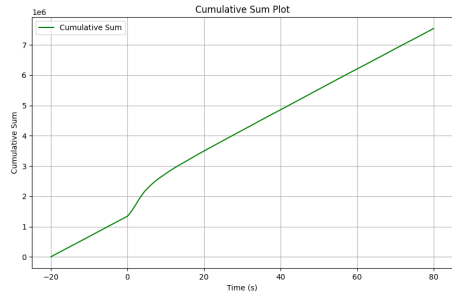
(b) Diagram for decomposition



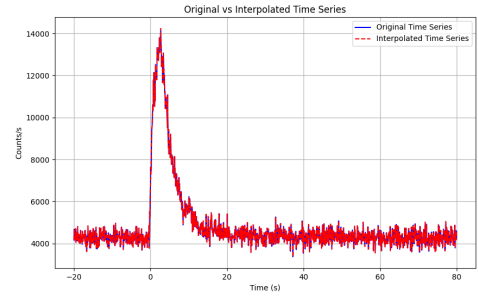
(c) Diagram for light curve's histogram



(d) Diagram for light curve's scatter plot



(e) Cumulative plot



(f) Interpolated Plot

III. APPENDIX FILE

A. Log.txt

Listing 3: log.txt

```
2024-05-05 20:02:52,699 - INFO - Data loaded successfully.
2024-05-05 20:02:52,701 - INFO - Data detrended and rescaled.
2024-05-05 20:02:54,417 - INFO - Detrended time series plotted.
2024-05-05 20:02:55,597 - INFO - Autocorrelation function of detrended series plotted.
2024-05-05 20:02:56,664 - INFO - Time series decomposition performed.
2024-05-05 20:02:59,331 - INFO - Data displayed.
2024-05-05 20:02:59,335 - INFO - Data loaded successfully.
```

2024-05-05 20:02:59,336 — INFO — Data loaded successfully.
2024-05-05 20:02:59,338 — INFO — Data interpolated using linear method.
2024-05-05 20:02:59,338 — INFO — Cumulative sum calculated.
2024-05-05 20:03:00,497 — INFO — Interpolated time series plotted.
2024-05-05 20:03:01,481 — INFO — Cumulative sum plot generated.