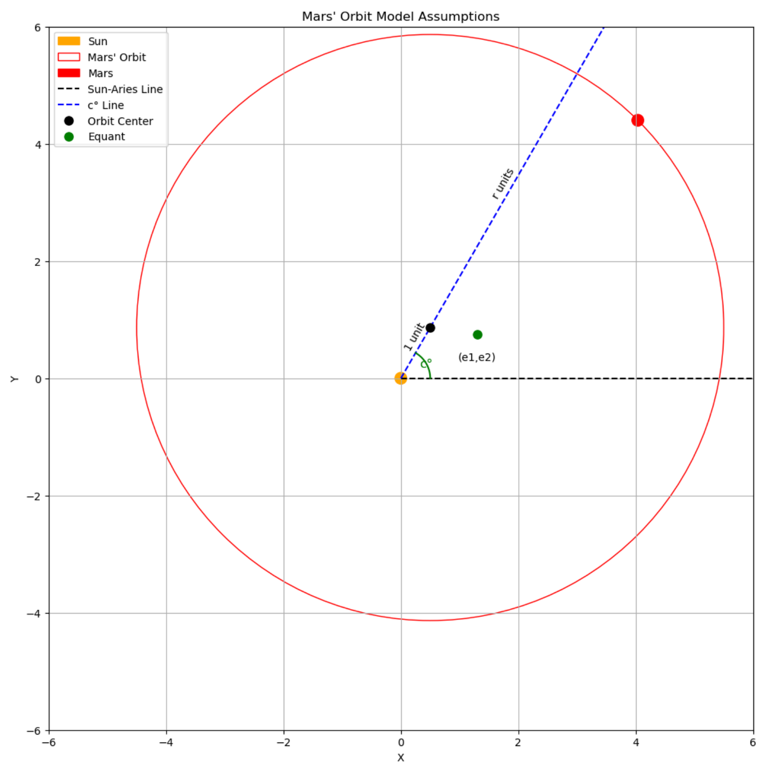
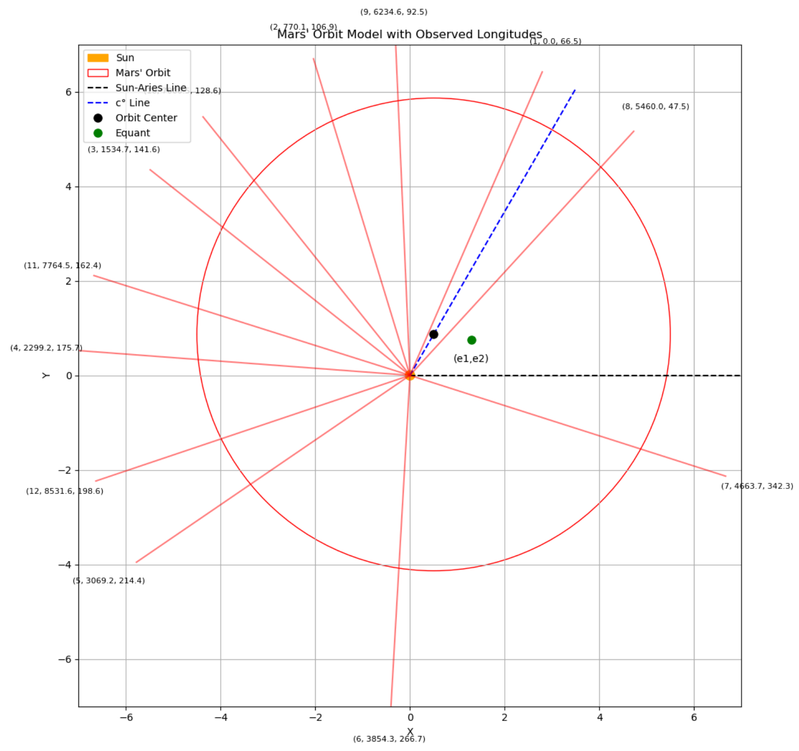
Report

- ## Plot for Mars orbit with assumptions

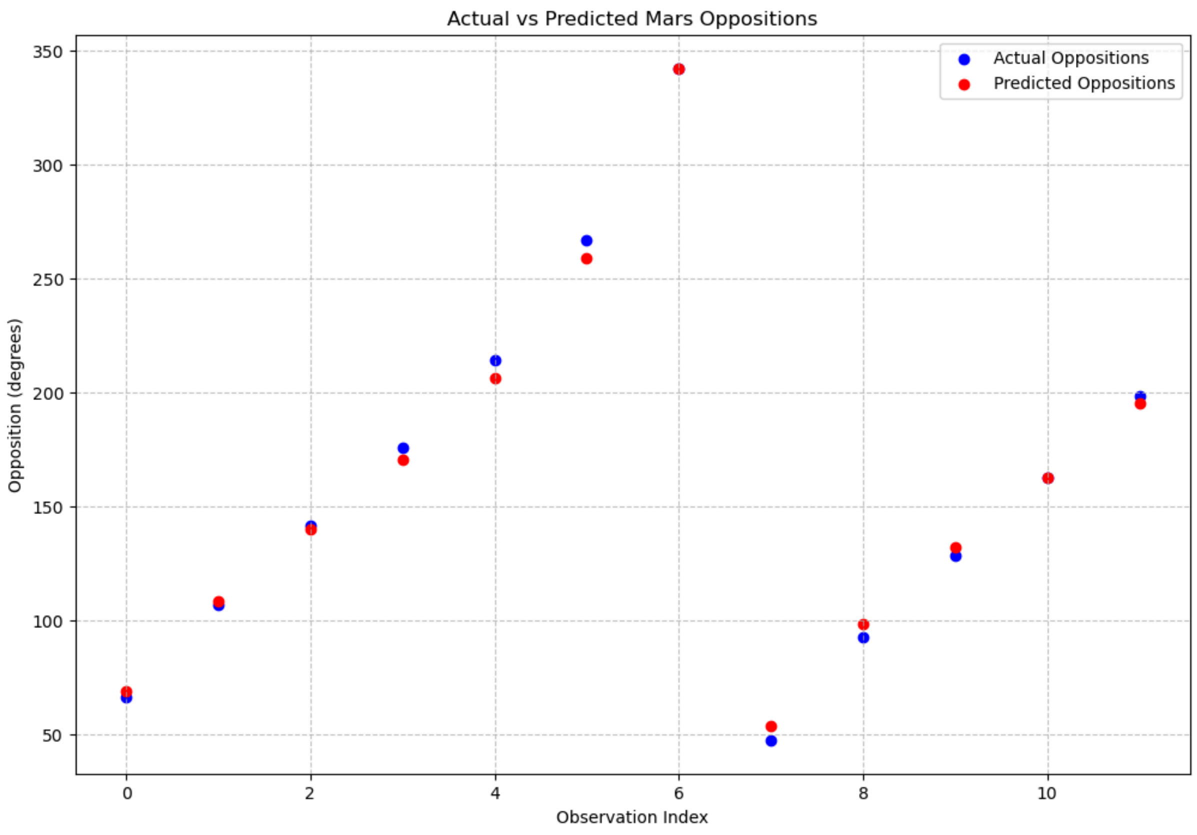


- ## Plot for Mars orbit with observed oppositions



- **The best parameters i got was:**
  - `Best parameters:`
  - `r = 1.444000`
  - `s = 0.523377`
  - `c = 0.000000`
  - `e1 = 0.100000`
  - `e2 = 140.000000`
  - `z = 60.000000`
  - `Maximum error = 8.030438`

- **The actual vs predicted oppositions:**



- **Errors array:**

```
[-2.62107888 -1.61262341  1.43363737  5.0624142
8.03043753  7.67691323
  0.09742122 -5.98070585 -6.22295592 -3.69086836
-0.17838611  3.13592717]
```

Some assumptions

Assume the following orbit model and parameters:

- The Sun is at the origin.
- Mars's orbit is circular, with the centre at a distance 1 unit from the Sun and at an angle c (degrees) from the Sun-Aries reference line.
- Mars's orbit has radius r (in units of the Sun-centre distance).
- The equant is located at (e1, e2) in polar coordinates with centre taken to be the Sun, where e1 is the distance from the Sun and e2 is the angle in degrees with respect to the Sun-Aries reference line.
- The 'equant 0' angle z (degrees) which is taken as the earliest opposition, also taken as the reference time zero, with respect to the equant-Aries line (a line parallel to the Sun-Aries line since Aries is at infinity).
- The angular velocity of Mars around the equant is s degrees per day.

> In this model, the center and the equant are different.
>
> - The center of Mars's orbit is at a distance of 1 unit from the Sun. It is the center of the circular path Mars follows around the Sun.
> - The equant, on the other hand, is a separate point located at coordinates (e1, e2) relative to the Sun in polar coordinates. The equant controls the angular speed of Mars as seen from its location. Mars moves at a uniform angular velocity s degrees per day around this equant.
>
> In simple terms: The center is where the orbit of Mars is centered, while the equant is a point from where the angular motion of Mars appears uniform. These two are not the same because they represent different parts of the model — one is related to the geometry of the orbit, and the other to how Mars's speed appears to change.

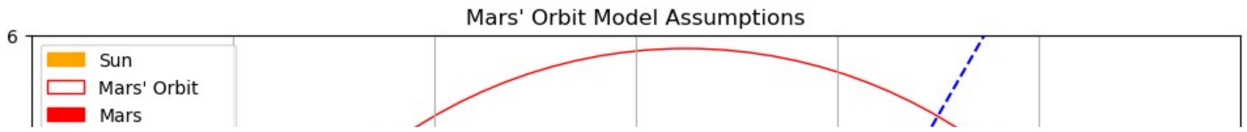
> The statement "***The 'equant 0' angle z (degrees) which is taken as the earliest opposition, also taken as the reference time zero, with respect to the equant-Aries line (a line parallel to the Sun-Aries line since Aries is at infinity)***"describes a reference point in the model, specifically when Mars is in "opposition" as seen from Earth, meaning Mars is directly opposite the Sun in the sky.
>
> Equant 0 angle (z degrees): This is an angle measured from the equant to Mars's

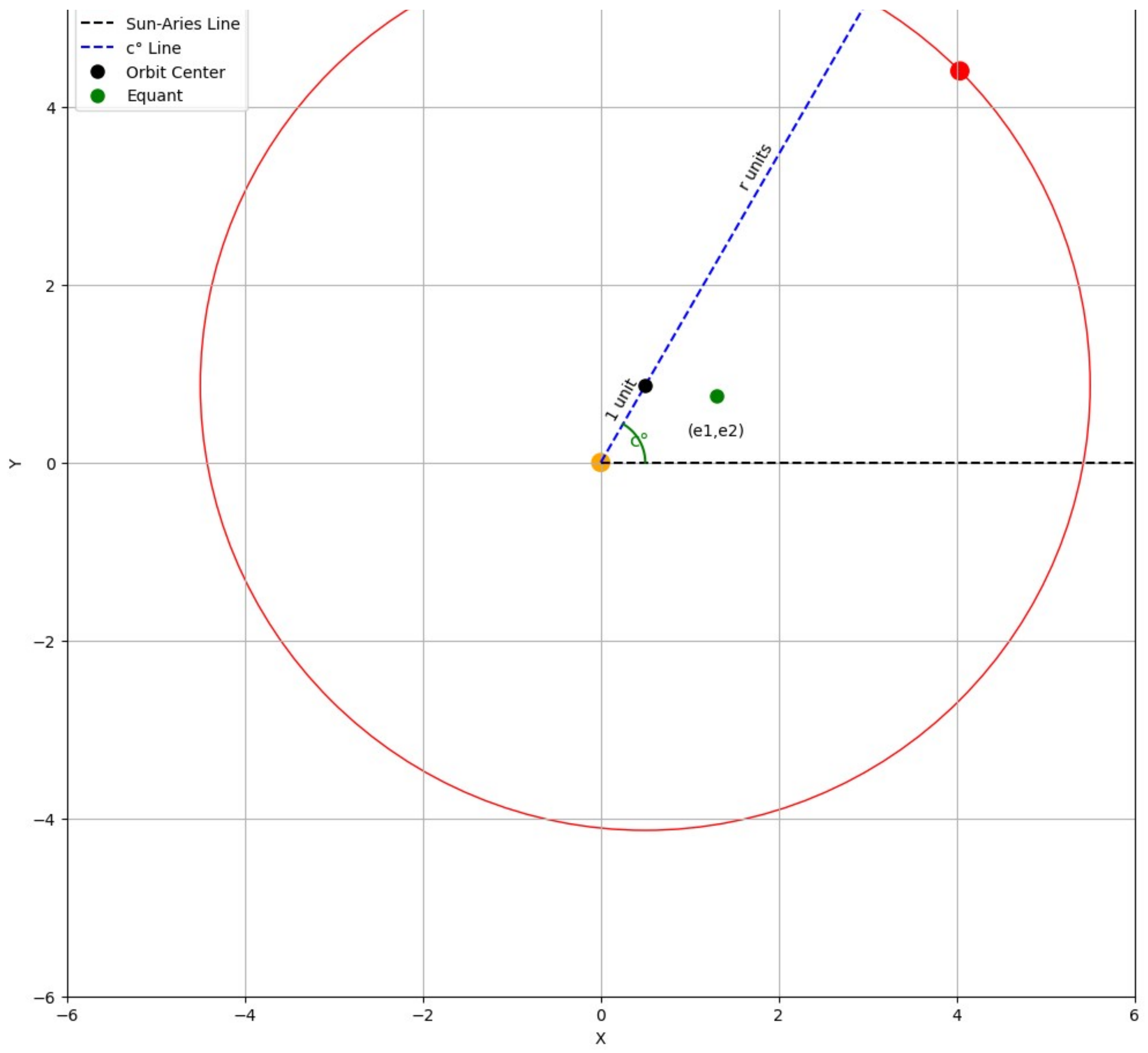
> position when it is in opposition. It defines the position of Mars at the earliest opposition (i.e., the first time Mars is directly opposite the Sun). This angle z is taken as zero degrees at this specific time, establishing a starting point for measuring time in the orbit.
>
> Reference time zero: This means that when Mars is at this position (earliest opposition), the clock starts counting from day zero.
>
> Equant-Aries line: This line is parallel to the Sun-Aries line. In ancient astronomy, Aries (a zodiac constellation) was used as a fixed reference point. The equant-Aries line is essentially a line extending from the equant, parallel to a line that would extend from the Sun to Aries. Since Aries is considered infinitely far away, the Sun-Aries and equant-Aries lines are treated as parallel.
>
> Simplified: The angle z is the starting point when Mars is first in opposition, and time is measured from that moment. The line used to define this angle is parallel to the Sun-Aries reference line.

Lets draw a picture to visualize things

```
import matplotlib.pyplot as plt  # Import matplotlib for plotting
import numpy as np  # Import numpy for numerical operations

plt.figure(figsize=(12, 12))  # Create a new figure with size 12x12
ax = plt.gca()  # Get the current axes
ax.set_aspect('equal')  # Set aspect ratio to equal

# Parameters
c = 60  # Angle in degrees
r = 5.0  # Radius of Mars' orbit (5 times bigger than 1 unit)
e1 = 1.5  # Distance of equant from Sun
e2 = 30  # Angle of equant with respect to Sun-Aries line

# Calculate positions
c_rad = np.radians(c)  # Convert angle to radians
center_x, center_y = np.cos(c_rad), np.sin(c_rad)  # Calculate orbit center coordinates
e2_rad = np.radians(e2)  # Convert equant angle to radians
equant_x, equant_y = e1 * np.cos(e2_rad), e1 * np.sin(e2_rad)  # Calculate equant coordin

# Draw the Sun
sun = plt.Circle((0, 0), 0.1, color='orange', label='Sun')  # Create Sun circle
ax.add_artist(sun)  # Add Sun to the plot

# Draw Mars' orbit
orbit = plt.Circle((center_x, center_y), r, fill=False, color='red', label="Mars' Orbit")
```

```
ax.add_artist(orbit)  # Add Mars orbit to the plot

# Draw Mars (45 degrees from orbit center)
mars_angle = np.radians(45)  # Convert Mars angle to radians
mars_x = center_x + r * np.cos(mars_angle)  # Calculate Mars x-coordinate
mars_y = center_y + r * np.sin(mars_angle)  # Calculate Mars y-coordinate
mars = plt.Circle((mars_x, mars_y), 0.1, color='red', label='Mars')  # Create Mars circle
ax.add_artist(mars)  # Add Mars to the plot

# Draw Sun-Aries reference line (0 degrees)
plt.plot([0, 7], [0, 0], 'k--', label='Sun-Aries Line')  # Plot Sun-Aries line

# Draw line at c degrees
plt.plot([0, 7*np.cos(c_rad)], [0, 7*np.sin(c_rad)], 'b--', label='c° Line')  # Plot c° l

# Mark orbit center
plt.plot(center_x, center_y, 'ko', markersize=8, label='Orbit Center')  # Plot orbit cent

# Mark equant
plt.plot(equant_x, equant_y, 'go', markersize=8, label='Equant')  # Plot equant
plt.text(equant_x, equant_y-0.3, f'(e1,e2)', ha='center', va='top')  # Add (e1,e2) text u

# Add text to show 1 unit distance
plt.text(center_x/2, center_y/2, '1 unit', rotation=c, ha='center', va='bottom')  # Add '

# Add r units along the line
plt.text(center_x + r*np.cos(c_rad)/2, center_y + r*np.sin(c_rad)/2, 'r units',
         rotation=c, ha='center', va='bottom')  # Add 'r units' text along the line

# Add angle label with arc for c
angle = np.linspace(0, c_rad, 100)  # Create angle array
arc_radius = 0.5  # Set arc radius
plt.plot(arc_radius * np.cos(angle), arc_radius * np.sin(angle), 'g-')  # Plot arc
plt.text(arc_radius * np.cos(c_rad/2), arc_radius * np.sin(c_rad/2), 'c°',
         ha='center', va='center', fontsize=12, color='green')  # Add 'c°' text

plt.xlabel('X')  # Set x-axis label
plt.ylabel('Y')  # Set y-axis label
plt.title("Mars' Orbit Model Assumptions")  # Set plot title
plt.legend(loc='upper left')  # Add legend
plt.xlim(-6, 6)  # Set x-axis limits
plt.ylim(-6, 6)  # Set y-axis limits
plt.grid(True)  # Add grid
plt.show()  # Display the plot

print("Plot generated successfully.")  # Debug statement
```





```
Plot generated successfully.
```

The dataset

```
import pandas as pd  # Import pandas library for data manipulation
import matplotlib.pyplot as plt  # Import matplotlib for plotting
from IPython.display import display  # Import display function for better output

# Load the CSV file
df = pd.read_csv('01_data_mars_opposition_updated.csv')  # Load data from CSV file
print("Data loaded from CSV file")  # Debug statement

# Display the data as a table
display(df)  # Display the DataFrame
```

```
Data loaded from CSV file
```

	Year	Month	Day	Hour	Minute	ZodiacIndex	Degree	Minute.1	Second	LatDegree
0	1580	11	18	1	31	2	6	28	35	1
1	1582	12	28	3	58	3	16	55	30	4
2	1585	1	30	19	14	4	21	36	10	4
3	1587	3	6	7	23	5	25	43	0	3
4	1589	4	14	6	23	7	4	23	0	1
5	1591	6	8	7	43	8	26	43	0	-4
6	1593	8	25	17	27	11	12	16	0	-6
7	1595	10	31	0	39	1	17	31	40	0
8	1597	12	13	15	44	3	2	28	0	3
9	1600	1	18	14	2	4	8	38	0	4
10	1602	2	20	14	13	5	12	27	0	4
11	1604	3	28	16	23	6	18	37	10	2

Columns J and K refer to the degree and minute of the geocentric latitudinal position of Mars in the ecliptic coordinate system. For our analysis, we will neglect these values and assume that Mars, Earth, the equant center, and the Sun all lie on the same plane. This simplification allows us to focus on the longitudinal aspects of Mars' orbit.

Columns L,M,N,O refer to Mars's mean longitude, with reference to Kepler's approximated equant. Instead of using this, we will find these based on your own equant. So we can ignore these columns.

Columns F, G, H, I denote the ZodiacIndex, Degree, Minute, Second, respectively, of Mars's (heliocentric) longitude in the ecliptic coordinate system. ZodiacIndex refers to the zodiac (Aries 0, Taurus 1, ..., Pisces 11). The longitude can be calculated using the formula: Longitude = ZodiacIndex*30 + Degree + Minute/60 + Second/3600 (degrees). This calculation provides the precise position of Mars in the ecliptic coordinate system.

Hence we can represnt the data as:

```
# Function to calculate longitude from zodiac data
def calculate_longitude(row):
    return row['ZodiacIndex'] * 30 + row['Degree'] + row['Minute.1'] / 60 + row['Second']

# Calculate longitude and add it as a new column
df['Longitude'] = df.apply(calculate_longitude, axis=1)  # Apply calculation to each row
print("Longitude calculated for each observation")  # Debug statement

# Select only the required columns
df_modified = df[['Year', 'Month', 'Day', 'Hour', 'Minute', 'Longitude']]  # Select relev
print("Columns selected: Year, Month, Day, Hour, Minute, Longitude")  # Debug statement

# Display the modified dataframe
display(df_modified)  # Show the modified DataFrame
print("Modified dataframe displayed")  # Debug statement
```

```
Longitude calculated for each observation
Columns selected: Year, Month, Day, Hour, Minute, Longitude
```

	Year	Month	Day	Hour	Minute	Longitude
0	1580	11	18	1	31	66.476389
1	1582	12	28	3	58	106.925000
2	1585	1	30	19	14	141.602778
3	1587	3	6	7	23	175.716667
4	1589	4	14	6	23	214.383333
5	1591	6	8	7	43	266.716667
6	1593	8	25	17	27	342.266667
7	1595	10	31	0	39	47.527778
8	1597	12	13	15	44	92.466667

9	1600	1	18	14	2	128.633333
10	1602	2	20	14	13	162.450000
11	1604	3	28	16	23	198.619444

```
Modified dataframe displayed
```

```
import pandas as pd
from datetime import datetime, timedelta

# Function to convert date and time to days elapsed
def days_elapsed(row, reference_date):
    date = datetime(int(row['Year']), int(row['Month']), int(row['Day']),
                    int(row['Hour']), int(row['Minute']))  # Create datetime object, conv
    return (date - reference_date).total_seconds() / (24 * 3600)  # Convert to days

# Get the reference date (first observation)
reference_date = datetime(int(df_modified['Year'].iloc[0]), int(df_modified['Month'].iloc
                          int(df_modified['Day'].iloc[0]), int(df_modified['Hour'].iloc[0
                          int(df_modified['Minute'].iloc[0]))  # Set reference date, conv

# Calculate days elapsed for each observation
df_modified['Days_Elapsed'] = df_modified.apply(lambda row: days_elapsed(row, reference_d

# Select only the required columns
df_final = df_modified[['Days_Elapsed', 'Longitude']]  # Select relevant columns

# Display the final dataframe
display(df_final)  # Show the final DataFrame
print("Final dataframe displayed")  # Debug statement
```

```
C:\Users\VICTOR\AppData\Local\Temp\ipykernel_11152\2642879292.py:16: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df_modified['Days_Elapsed'] = df_modified.apply(lambda row: days_elapsed(row, refer
```

	Days_Elapsed	Longitude
0	0.000000	66.476389
1	770.102083	106.925000
2	1534.738194	141.602778
3	2299.244444	175.716667
4	3069.202778	214.383333
5	3854.258333	266.716667

```
 6   4663.663889  342.266667
 7   5459.963889   47.527778
 8   6234.592361   92.466667
 9   7000.521528  128.633333
10   7764.529167  162.450000
11   8531.619444  198.619444
Final dataframe displayed
```

```
import numpy as np  # Import numpy library

# Create 'times' numpy array from 'Days_Elapsed' column
times = np.array(df_final['Days_Elapsed'])  # Convert Days_Elapsed to numpy array
print("Created 'times' numpy array:", times)  # Print debug statement

# Verify the length of 'times' array
print("Length of 'times' array:", len(times))  # Print debug statement

assert len(times) == 12, "Error: 'times' array is not of length 12"  # Assert correct len
```

```
Created 'times' numpy array: [   0.          770.10208333 1534.73819444 2299.24444444
 3854.25833333 4663.66388889 5459.96388889 6234.59236111 7000.52152778
 7764.52916667 8531.61944444]
Length of 'times' array: 12
```

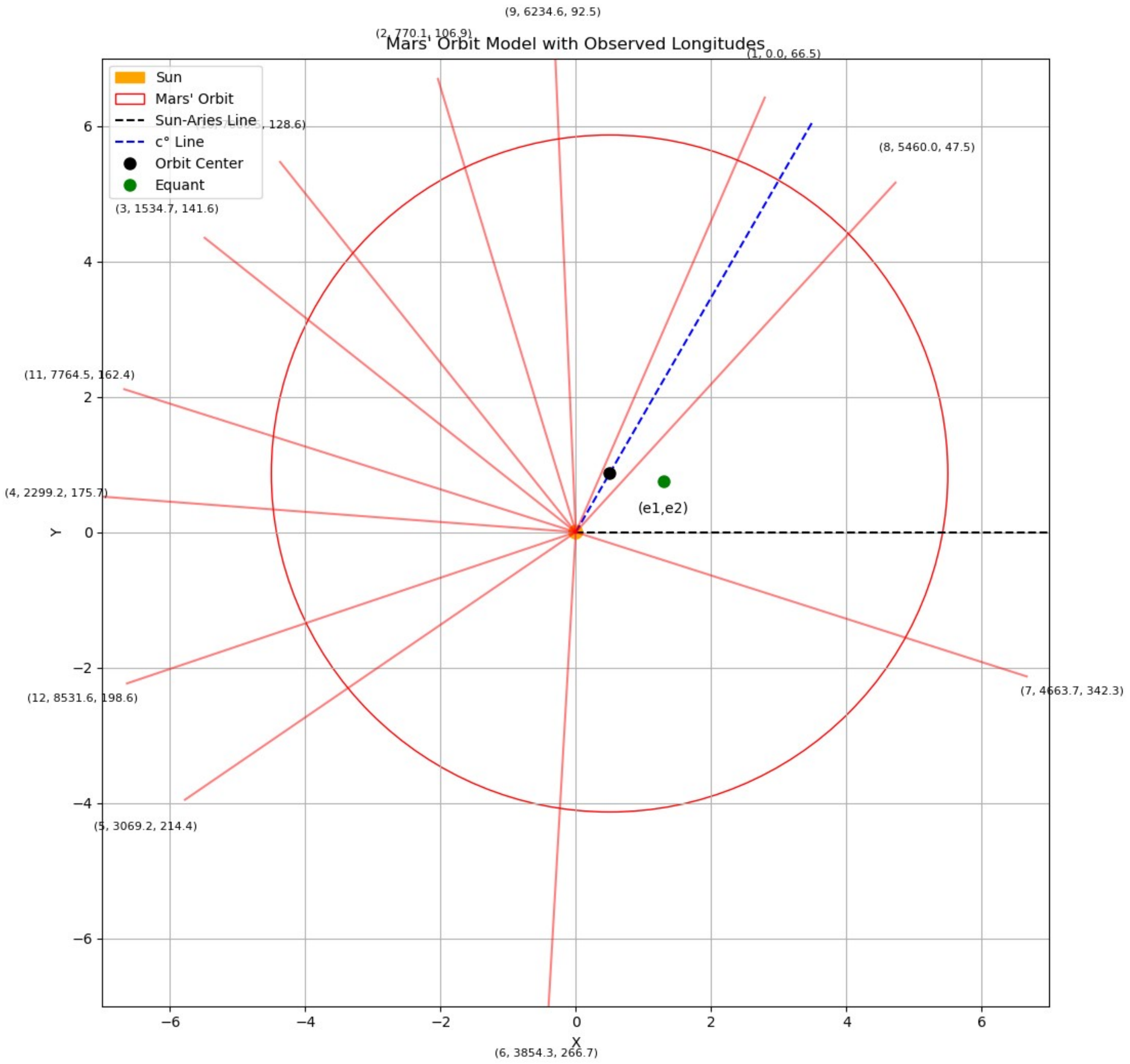
```
# Create 'oppositions' numpy array from 'Longitude' column
oppositions = np.array(df_final['Longitude'])  # Convert Longitude to numpy array
print("Created 'oppositions' numpy array:", oppositions)  # Print debug statement

# Verify the length of 'oppositions' array
print("Length of 'oppositions' array:", len(oppositions))  # Print debug statement

assert len(oppositions) == 12, "Error: 'oppositions' array is not of length 12"  # Assert
```

```
Created 'oppositions' numpy array: [ 66.47638889 106.925      141.60277778 175.71666
 266.71666667 342.26666667  47.52777778  92.46666667 128.63333333
 162.45       198.61944444]
Length of 'oppositions' array: 12
```

Now lets plot these opposition data on plot

Note that each datapoint is represented by a ray from sun along with (Ser_no, time, longtitude)

```
import matplotlib.pyplot as plt  # Import matplotlib for plotting
import numpy as np  # Import numpy for numerical operations

plt.figure(figsize=(12, 12))  # Create a new figure with size 12x12
ax = plt.gca()  # Get the current axes
ax.set_aspect('equal')  # Set aspect ratio to equal

# Parameters
c = 60  # Angle in degrees
r = 5.0  # Radius of Mars' orbit (5 times bigger than 1 unit)
e1 = 1.5  # Distance of equant from Sun
e2 = 30  # Angle of equant with respect to Sun-Aries line

# Calculate positions
c_rad = np.radians(c)  # Convert angle to radians
center_x, center_y = np.cos(c_rad), np.sin(c_rad)  # Calculate orbit center coordinates
e2_rad = np.radians(e2)  # Convert equant angle to radians
equant_x, equant_y = e1 * np.cos(e2_rad), e1 * np.sin(e2_rad)  # Calculate equant coordin

# Draw the Sun
sun = plt.Circle((0, 0), 0.1, color='orange', label='Sun')  # Create Sun circle
ax.add_artist(sun)  # Add Sun to the plot

# Draw Mars' orbit
orbit = plt.Circle((center_x, center_y), r, fill=False, color='red', label="Mars' Orbit")
ax.add_artist(orbit)  # Add Mars orbit to the plot

# Draw Sun-Aries reference line (0 degrees)
plt.plot([0, 7], [0, 0], 'k--', label='Sun-Aries Line')  # Plot Sun-Aries line

# Draw line at c degrees
plt.plot([0, 7*np.cos(c_rad)], [0, 7*np.sin(c_rad)], 'b--', label='c° Line')  # Plot c° l

# Mark orbit center
plt.plot(center_x, center_y, 'ko', markersize=8, label='Orbit Center')  # Plot orbit cent

# Mark equant
plt.plot(equant_x, equant_y, 'go', markersize=8, label='Equant')  # Plot equant
plt.text(equant_x, equant_y-0.3, f'(e1,e2)', ha='center', va='top')  # Add (e1,e2) text u

# Plot longitudes as rays from the sun
for i, (days, longitude) in enumerate(zip(times, oppositions)):  # Iterate through opposi
    angle = np.radians(longitude)  # Convert longitude to radians
    x = 7 * np.cos(angle)  # Calculate x-coordinate
    y = 7 * np.sin(angle)  # Calculate y-coordinate
    plt.plot([0, x], [0, y], 'r-', alpha=0.5)  # Plot ray
    plt.text(x*1.1, y*1.1, f'({i+1}, {days:.1f}, {longitude:.1f})', fontsize=8, ha='cente

plt.xlabel('X')  # Set x-axis label
plt.ylabel('Y')  # Set y-axis label
plt.title("Mars' Orbit Model with Observed Longitudes")  # Set plot title
plt.legend(loc='upper left')  # Add legend
plt.xlim(-7, 7)  # Set x-axis limits
plt.ylim(-7, 7)  # Set y-axis limits
plt.grid(True)  # Add grid
plt.show()  # Display the plot
```



Now lets write a function to calculate error for specific values of c,r,e1,e2,z,s,times,oppositions

```
import numpy as np

def MarsEquantModel(c, r, e1, e2, z, s, times, oppositions):
    errors = np.zeros(12)  # Initialize errors array with zeros

    # Convert angles to radians
    c_rad = np.radians(c)  # Convert c to radians
    e2_rad = np.radians(e2)  # Convert e2 to radians
    z_rad = np.radians(z)  # Convert z to radians

    # Calculate orbit center and equant positions
    center_x, center_y = np.cos(c_rad), np.sin(c_rad)  # Calculate orbit center coordinat
    equant_x, equant_y = e1 * np.cos(e2_rad), e1 * np.sin(e2_rad)  # Calculate equant coo

    for i in range(12):  # Loop through all 12 observations
        t = times[i]  # Get current time
        angle = z_rad + np.radians(s * t)  # Calculate angle of Mars around equant

        # Calculate Mars position relative to equant (assuming Mars is r distance from eq
        mars_x_eq = r * np.cos(angle)  # X-coordinate of Mars relative to equant
        mars_y_eq = r * np.sin(angle)  # Y-coordinate of Mars relative to equant

        # Calculate Mars position relative to Sun
        mars_x = equant_x + mars_x_eq
        mars_y = equant_y + mars_y_eq

        # Calculate predicted longitude
        predicted_long = np.degrees(np.arctan2(mars_y, mars_x)) % 360  # Convert to degre

        # Calculate error
        error = (predicted_long - oppositions[i] + 180) % 360 - 180  # Calculate smallest
        error = (predicted_long - oppositions[i] + 180) % 360 - 180  # Calculate smallest
        errors[i] = error  # Store error in array

    max_error = np.max(np.abs(errors))  # Calculate maximum absolute error

    return errors, max_error  # Return errors array and max error
```

Now lets fix r and s. Do a discretised exhaustive search over c, over e = (e1,e2), and over z to minimise the maximum angular error for the given r and s.

```
import numpy as np

def bestOrbitInnerParams(r, s, times, oppositions):
    # Define search ranges and step sizes
    c_range = np.arange(0, 360, 20)  # Search c from 0 to 360 in steps of 10 degrees
    e1_range = np.arange(0, 0.2, 0.1)  # Search e1 from 0 to 0.2 in steps of 0.01
    e2_range = np.arange(0, 360, 20)  # Search e2 from 0 to 360 in steps of 10 degrees
    z_range = np.arange(0, 360, 20)  # Search z from 0 to 360 in steps of 10 degrees
    total_iterations = len(c_range) * len(e1_range) * len(e2_range) * len(z_range)  # Cal
    current_iteration = 0  # Initialize current iteration counter

    best_error = float('inf')  # Initialize best error to infinity
    best_params = None  # Initialize best parameters

    # Nested loops for exhaustive search
    for c in c_range:
        for e1 in e1_range:
            for e2 in e2_range:
                for z in z_range:
                    # Calculate errors for current parameter set
                    errors, max_error = MarsEquantModel(c, r, e1, e2, z, s, times, opposi
                    # Update current iteration
                    current_iteration += 1  # Increment the iteration counter
                    # Print debug statement for current parameter set and error
                    print(f"Debug: Iteration {current_iteration}/{total_iterations}: c={c
                    # Update best parameters if current error is lower
                    if max_error < best_error:
                        best_error = max_error
                        best_params = (c, e1, e2, z, errors, max_error)


    # Unpack best parameters
    c, e1, e2, z, errors, max_error = best_params

    return c, e1, e2, z, errors, max_error
```

Fix r. Do a discretised search for s

```
def bestS(r, times, oppositions):
    # Define search range for s
    s_range = np.linspace(360/687 * 0.9, 360/687 * 1.1, 10)  # Search around 360/687, ±10

    best_error = float('inf')
    best_params = None

    for s in s_range:
        c, e1, e2, z, errors, max_error = bestOrbitInnerParams(r, s, times, oppositions)

        if max_error < best_error:
            best_error = max_error
            best_params = (s, errors, max_error)

        print(f"Debug: s={s:.6f}, max_error={max_error:.4f}, best_overall_max_error={best

    return best_params


# Set a fixed value for r (you may need to adjust this based on your previous findings)
r_fixed = 1.52  # This is an approximate value for Mars' orbit radius in AU

# Use the bestS function to find the optimal s and other parameters
best_s, best_errors, best_max_error = bestS(r_fixed, times, oppositions)

# Now use the best s to find the other optimal parameters
best_c, best_e1, best_e2, best_z, _, _ = bestOrbitInnerParams(r_fixed, best_s, times, opp
```

```
 Debug: Iteration 1/11664: c=0.00, e1=0.0000, e2=0.00, z=0.00, max_error=171.3568, bes
 Debug: Iteration 2/11664: c=0.00, e1=0.0000, e2=0.00, z=20.00, max_error=173.1009, be
 Debug: Iteration 3/11664: c=0.00, e1=0.0000, e2=0.00, z=40.00, max_error=166.8991, be
 Debug: Iteration 4/11664: c=0.00, e1=0.0000, e2=0.00, z=60.00, max_error=171.0122, be
 Debug: Iteration 5/11664: c=0.00, e1=0.0000, e2=0.00, z=80.00, max_error=168.9878, be
 Debug: Iteration 6/11664: c=0.00, e1=0.0000, e2=0.00, z=100.00, max_error=157.1905, b
 Debug: Iteration 7/11664: c=0.00, e1=0.0000, e2=0.00, z=120.00, max_error=177.1905, b
 Debug: Iteration 8/11664: c=0.00, e1=0.0000, e2=0.00, z=140.00, max_error=162.8095, b
 Debug: Iteration 9/11664: c=0.00, e1=0.0000, e2=0.00, z=160.00, max_error=167.4770, b
 Debug: Iteration 10/11664: c=0.00, e1=0.0000, e2=0.00, z=180.00, max_error=172.5230,
 Debug: Iteration 11/11664: c=0.00, e1=0.0000, e2=0.00, z=200.00, max_error=167.8651,
 Debug: Iteration 12/11664: c=0.00, e1=0.0000, e2=0.00, z=220.00, max_error=172.1349,
 Debug: Iteration 13/11664: c=0.00, e1=0.0000, e2=0.00, z=240.00, max_error=173.5236,
 Debug: Iteration 14/11664: c=0.00, e1=0.0000, e2=0.00, z=260.00, max_error=167.0773,
 Debug: Iteration 15/11664: c=0.00, e1=0.0000, e2=0.00, z=280.00, max_error=179.4240,
 Debug: Iteration 16/11664: c=0.00, e1=0.0000, e2=0.00, z=300.00, max_error=165.0264,
 Debug: Iteration 17/11664: c=0.00, e1=0.0000, e2=0.00, z=320.00, max_error=177.7961,
 Debug: Iteration 18/11664: c=0.00, e1=0.0000, e2=0.00, z=340.00, max_error=168.6432,
 Debug: Iteration 19/11664: c=0.00, e1=0.0000, e2=20.00, z=0.00, max_error=171.3568, b
 Debug: Iteration 20/11664: c=0.00, e1=0.0000, e2=20.00, z=20.00, max_error=173.1009,
 Debug: Iteration 21/11664: c=0.00, e1=0.0000, e2=20.00, z=40.00, max_error=166.8991,
 Debug: Iteration 22/11664: c=0.00, e1=0.0000, e2=20.00, z=60.00, max_error=171.0122,
 Debug: Iteration 23/11664: c=0.00, e1=0.0000, e2=20.00, z=80.00, max_error=168.9878,
 Debug: Iteration 24/11664: c=0.00, e1=0.0000, e2=20.00, z=100.00, max_error=157.1905,
 Debug: Iteration 25/11664: c=0.00, e1=0.0000, e2=20.00, z=120.00, max_error=177.1905,
 Debug: Iteration 26/11664: c=0.00, e1=0.0000, e2=20.00, z=140.00, max_error=162.8095,
 Debug: Iteration 27/11664: c=0.00, e1=0.0000, e2=20.00, z=160.00, max_error=167.4770,
 Debug: Iteration 28/11664: c=0.00, e1=0.0000, e2=20.00, z=180.00, max_error=172.5230,
 Debug: Iteration 29/11664: c=0.00, e1=0.0000, e2=20.00, z=200.00, max_error=167.8651,
 Debug: Iteration 30/11664: c=0.00, e1=0.0000, e2=20.00, z=220.00, max_error=172.1349,
 Debug: Iteration 31/11664: c=0.00, e1=0.0000, e2=20.00, z=240.00, max_error=173.5236,
 Debug: Iteration 32/11664: c=0.00, e1=0.0000, e2=20.00, z=260.00, max_error=167.0773,
 Debug: Iteration 33/11664: c=0.00, e1=0.0000, e2=20.00, z=280.00, max_error=179.4240,
 Debug: Iteration 34/11664: c=0.00, e1=0.0000, e2=20.00, z=300.00, max_error=165.0264,
 Debug: Iteration 35/11664: c=0.00, e1=0.0000, e2=20.00, z=320.00, max_error=177.7961,
 Debug: Iteration 36/11664: c=0.00, e1=0.0000, e2=20.00, z=340.00, max_error=168.6432,
 Debug: Iteration 37/11664: c=0.00, e1=0.0000, e2=40.00, z=0.00, max_error=171.3568, b
 Debug: Iteration 38/11664: c=0.00, e1=0.0000, e2=40.00, z=20.00, max_error=173.1009,
 Debug: Iteration 39/11664: c=0.00, e1=0.0000, e2=40.00, z=40.00, max_error=166.8991,
 Debug: Iteration 40/11664: c=0.00, e1=0.0000, e2=40.00, z=60.00, max_error=171.0122,
 Debug: Iteration 41/11664: c=0.00, e1=0.0000, e2=40.00, z=80.00, max_error=168.9878,
 Debug: Iteration 42/11664: c=0.00, e1=0.0000, e2=40.00, z=100.00, max_error=157.1905,
 Debug: Iteration 43/11664: c=0.00, e1=0.0000, e2=40.00, z=120.00, max_error=177.1905,
 Debug: Iteration 44/11664: c=0.00, e1=0.0000, e2=40.00, z=140.00, max_error=162.8095,
 Debug: Iteration 45/11664: c=0.00, e1=0.0000, e2=40.00, z=160.00, max_error=167.4770,
 Debug: Iteration 46/11664: c=0.00, e1=0.0000, e2=40.00, z=180.00, max_error=172.5230,
 Debug: Iteration 47/11664: c=0.00, e1=0.0000, e2=40.00, z=200.00, max_error=167.8651,
 Debug: Iteration 48/11664: c=0.00, e1=0.0000, e2=40.00, z=220.00, max_error=172.1349,
 Debug: Iteration 49/11664: c=0.00, e1=0.0000, e2=40.00, z=240.00, max_error=173.5236,
 Debug: Iteration 50/11664: c=0.00, e1=0.0000, e2=40.00, z=260.00, max_error=167.0773,
 Debug: Iteration 51/11664: c=0.00, e1=0.0000, e2=40.00, z=280.00, max_error=179.4240,
 Debug: Iteration 52/11664: c=0.00, e1=0.0000, e2=40.00, z=300.00, max_error=165.0264,
 Debug: Iteration 53/11664: c=0.00, e1=0.0000, e2=40.00, z=320.00, max_error=177.7961,
 Debug: Iteration 54/11664: c=0.00, e1=0.0000, e2=40.00, z=340.00, max_error=168.6432,
 Debug: Iteration 55/11664: c=0.00, e1=0.0000, e2=60.00, z=0.00, max_error=171.3568, b
 Debug: Iteration 56/11664: c=0.00, e1=0.0000, e2=60.00, z=20.00, max_error=173.1009,
 Debug: Iteration 57/11664: c=0.00, e1=0.0000, e2=60.00, z=40.00, max_error=166.8991,
 Debug: Iteration 58/11664: c=0.00, e1=0.0000, e2=60.00, z=60.00, max_error=171.0122,
```

```
# Print the best values
print(f"Best parameters:")
print(f"r = {r_fixed:.6f}")
print(f"s = {best_s:.6f}")
print(f"c = {best_c:.6f}")
print(f"e1 = {best_e1:.6f}")
print(f"e2 = {best_e2:.6f}")
print(f"z = {best_z:.6f}")
print(f"Maximum error = {best_max_error:.6f}")
```

```
    Best parameters:
    r = 1.520000
    s = 0.518195
    c = 0.000000
    e1 = 0.100000
    e1 = 0.100000
    e2 = 180.000000
    z = 80.000000
    Maximum error = 18.099304
```

Fix s Do discrete search for r

```
def bestR(s, times, oppositions):
    s_fixed = 0.518195  # Fixed value for s
    r_range = np.linspace(1.52 * 0.9, 1.52 * 1.1, 10)  # Search around 1.52, ±10%, with 2

    best_error = float('inf')
    best_params = None

    for r in r_range:
        c, e1, e2, z, errors, max_error = bestOrbitInnerParams(r, s_fixed, times, opposit

        if max_error < best_error:
            best_error = max_error
            best_params = (r, errors, max_error)

        print(f"Debug: r={r:.6f}, max_error={max_error:.4f}, best_overall_max_error={best

    return best_params


best_r, best_errors, best_max_error = bestR(0.518195, times, oppositions)

# Now use the best r to find the other optimal parameters
best_c, best_e1, best_e2, best_z, _, _ = bestOrbitInnerParams(best_r, 0.518195, times, op

# Print the best values
print(f"\nBest parameters:")
print(f"r = {best_r:.6f}")
print(f"s = 0.518195")  # Fixed value
print(f"c = {best_c:.6f}")
print(f"e1 = {best_e1:.6f}")
print(f"e2 = {best_e2:.6f}")
print(f"z = {best_z:.6f}")
print(f"Maximum error = {best_max_error:.6f}")
```

```
    Debug: Iteration 1/11664: c=0.00, e1=0.0000, e2=0.00, z=0.00, max_error=101.7321, bes
    Debug: Iteration 2/11664: c=0.00, e1=0.0000, e2=0.00, z=20.00, max_error=81.7321, bes
    Debug: Iteration 3/11664: c=0.00, e1=0.0000, e2=0.00, z=40.00, max_error=61.7321, bes
    Debug: Iteration 4/11664: c=0.00, e1=0.0000, e2=0.00, z=60.00, max_error=41.7321, bes
    Debug: Iteration 5/11664: c=0.00, e1=0.0000, e2=0.00, z=80.00, max_error=21.7321, bes
    Debug: Iteration 6/11664: c=0.00, e1=0.0000, e2=0.00, z=100.00, max_error=36.0622, be
    Debug: Iteration 7/11664: c=0.00, e1=0.0000, e2=0.00, z=120.00, max_error=56.0622, be
    Debug: Iteration 8/11664: c=0.00, e1=0.0000, e2=0.00, z=140.00, max_error=76.0622, be
    Debug: Iteration 9/11664: c=0.00, e1=0.0000, e2=0.00, z=160.00, max_error=96.0622, be
```

```
Debug: Iteration 10/11664: c=0.00, e1=0.0000, e2=0.00, z=180.00, max_error=116.0622,
Debug: Iteration 11/11664: c=0.00, e1=0.0000, e2=0.00, z=200.00, max_error=136.0622,
Debug: Iteration 12/11664: c=0.00, e1=0.0000, e2=0.00, z=220.00, max_error=156.0622,
Debug: Iteration 13/11664: c=0.00, e1=0.0000, e2=0.00, z=240.00, max_error=176.0622,
Debug: Iteration 14/11664: c=0.00, e1=0.0000, e2=0.00, z=260.00, max_error=174.4206,
Debug: Iteration 15/11664: c=0.00, e1=0.0000, e2=0.00, z=280.00, max_error=179.0019,
Debug: Iteration 16/11664: c=0.00, e1=0.0000, e2=0.00, z=300.00, max_error=161.7321,
Debug: Iteration 17/11664: c=0.00, e1=0.0000, e2=0.00, z=320.00, max_error=141.7321,
Debug: Iteration 18/11664: c=0.00, e1=0.0000, e2=0.00, z=340.00, max_error=121.7321,
Debug: Iteration 19/11664: c=0.00, e1=0.0000, e2=20.00, z=0.00, max_error=101.7321, b
Debug: Iteration 20/11664: c=0.00, e1=0.0000, e2=20.00, z=20.00, max_error=81.7321, b
Debug: Iteration 21/11664: c=0.00, e1=0.0000, e2=20.00, z=40.00, max_error=61.7321, b
Debug: Iteration 22/11664: c=0.00, e1=0.0000, e2=20.00, z=60.00, max_error=41.7321, b
Debug: Iteration 23/11664: c=0.00, e1=0.0000, e2=20.00, z=80.00, max_error=21.7321, b
Debug: Iteration 24/11664: c=0.00, e1=0.0000, e2=20.00, z=100.00, max_error=36.0622,
Debug: Iteration 25/11664: c=0.00, e1=0.0000, e2=20.00, z=120.00, max_error=56.0622,
Debug: Iteration 26/11664: c=0.00, e1=0.0000, e2=20.00, z=140.00, max_error=76.0622,
Debug: Iteration 27/11664: c=0.00, e1=0.0000, e2=20.00, z=160.00, max_error=96.0622,
Debug: Iteration 28/11664: c=0.00, e1=0.0000, e2=20.00, z=180.00, max_error=116.0622,
Debug: Iteration 29/11664: c=0.00, e1=0.0000, e2=20.00, z=200.00, max_error=136.0622,
Debug: Iteration 30/11664: c=0.00, e1=0.0000, e2=20.00, z=220.00, max_error=156.0622,
Debug: Iteration 31/11664: c=0.00, e1=0.0000, e2=20.00, z=240.00, max_error=176.0622,
Debug: Iteration 32/11664: c=0.00, e1=0.0000, e2=20.00, z=260.00, max_error=174.4206,
Debug: Iteration 33/11664: c=0.00, e1=0.0000, e2=20.00, z=280.00, max_error=179.0019,
Debug: Iteration 34/11664: c=0.00, e1=0.0000, e2=20.00, z=300.00, max_error=161.7321,
Debug: Iteration 35/11664: c=0.00, e1=0.0000, e2=20.00, z=320.00, max_error=141.7321,
Debug: Iteration 36/11664: c=0.00, e1=0.0000, e2=20.00, z=340.00, max_error=121.7321,
Debug: Iteration 37/11664: c=0.00, e1=0.0000, e2=40.00, z=0.00, max_error=101.7321, b
Debug: Iteration 38/11664: c=0.00, e1=0.0000, e2=40.00, z=20.00, max_error=81.7321, b
Debug: Iteration 39/11664: c=0.00, e1=0.0000, e2=40.00, z=40.00, max_error=61.7321, b
Debug: Iteration 40/11664: c=0.00, e1=0.0000, e2=40.00, z=60.00, max_error=41.7321, b
Debug: Iteration 41/11664: c=0.00, e1=0.0000, e2=40.00, z=80.00, max_error=21.7321, b
Debug: Iteration 42/11664: c=0.00, e1=0.0000, e2=40.00, z=100.00, max_error=36.0622,
Debug: Iteration 43/11664: c=0.00, e1=0.0000, e2=40.00, z=120.00, max_error=56.0622,
Debug: Iteration 44/11664: c=0.00, e1=0.0000, e2=40.00, z=140.00, max_error=76.0622,
Debug: Iteration 45/11664: c=0.00, e1=0.0000, e2=40.00, z=160.00, max_error=96.0622,
Debug: Iteration 46/11664: c=0.00, e1=0.0000, e2=40.00, z=180.00, max_error=116.0622,
Debug: Iteration 47/11664: c=0.00, e1=0.0000, e2=40.00, z=200.00, max_error=136.0622,
Debug: Iteration 48/11664: c=0.00, e1=0.0000, e2=40.00, z=220.00, max_error=156.0622,
Debug: Iteration 49/11664: c=0.00, e1=0.0000, e2=40.00, z=240.00, max_error=176.0622,
Debug: Iteration 50/11664: c=0.00, e1=0.0000, e2=40.00, z=260.00, max_error=174.4206,
Debug: Iteration 51/11664: c=0.00, e1=0.0000, e2=40.00, z=280.00, max_error=179.0019,
Debug: Iteration 52/11664: c=0.00, e1=0.0000, e2=40.00, z=300.00, max_error=161.7321,
Debug: Iteration 53/11664: c=0.00, e1=0.0000, e2=40.00, z=320.00, max_error=141.7321,
Debug: Iteration 54/11664: c=0.00, e1=0.0000, e2=40.00, z=340.00, max_error=121.7321,
Debug: Iteration 55/11664: c=0.00, e1=0.0000, e2=60.00, z=0.00, max_error=101.7321, b
Debug: Iteration 56/11664: c=0.00, e1=0.0000, e2=60.00, z=20.00, max_error=81.7321, b
Debug: Iteration 57/11664: c=0.00, e1=0.0000, e2=60.00, z=40.00, max_error=61.7321, b
Debug: Iteration 58/11664: c=0.00, e1=0.0000, e2=60.00, z=60.00, max_error=41.7321, b
```

Search iteratively over r and s

```
def bestMarsOrbitParams(times, oppositions):
    r_initial = 1.520000
    s_initial = 0.518195

    r_range = np.linspace(r_initial * 0.95, r_initial * 1.05, 6)
    s_range = np.linspace(s_initial * 0.95, s_initial * 1.05, 6)

    best_error = float('inf')
    best_params = None

    for r in r_range:
        for s in s_range:
            c, e1, e2, z, errors, max_error = bestOrbitInnerParams(r, s, times, oppositio

            if max_error < best_error:
                best_error = max_error
                best_params = (r, s, c, e1, e2, z, errors, max_error)

    return best_params


# Use the function
r, s, c, e1, e2, z, errors, maxError = bestMarsOrbitParams(times, oppositions)

print(f"\nBest parameters:")
print(f"r = {r:.6f}")
print(f"s = {s:.6f}")
print(f"c = {c:.6f}")
print(f"e1 = {e1:.6f}")
print(f"e2 = {e2:.6f}")
print(f"z = {z:.6f}")
print(f"Maximum error = {maxError:.6f}")
```

```
 Debug: Iteration 1/11664: c=0.00, e1=0.0000, e2=0.00, z=0.00, max_error=169.3221, bes
 Debug: Iteration 2/11664: c=0.00, e1=0.0000, e2=0.00, z=20.00, max_error=173.5863, be
 Debug: Iteration 3/11664: c=0.00, e1=0.0000, e2=0.00, z=40.00, max_error=166.4137, be
 Debug: Iteration 4/11664: c=0.00, e1=0.0000, e2=0.00, z=60.00, max_error=179.6681, be
 Debug: Iteration 5/11664: c=0.00, e1=0.0000, e2=0.00, z=80.00, max_error=176.7312, be
 Debug: Iteration 6/11664: c=0.00, e1=0.0000, e2=0.00, z=100.00, max_error=177.6202, b
 Debug: Iteration 7/11664: c=0.00, e1=0.0000, e2=0.00, z=120.00, max_error=179.9132, b
 Debug: Iteration 8/11664: c=0.00, e1=0.0000, e2=0.00, z=140.00, max_error=178.6290, b
 Debug: Iteration 9/11664: c=0.00, e1=0.0000, e2=0.00, z=160.00, max_error=158.6290, b
 Debug: Iteration 10/11664: c=0.00, e1=0.0000, e2=0.00, z=180.00, max_error=138.6290,
 Debug: Iteration 11/11664: c=0.00, e1=0.0000, e2=0.00, z=200.00, max_error=133.5236,
 Debug: Iteration 12/11664: c=0.00, e1=0.0000, e2=0.00, z=220.00, max_error=153.5236,
 Debug: Iteration 13/11664: c=0.00, e1=0.0000, e2=0.00, z=240.00, max_error=173.5236,
 Debug: Iteration 14/11664: c=0.00, e1=0.0000, e2=0.00, z=260.00, max_error=172.1849,
 Debug: Iteration 15/11664: c=0.00, e1=0.0000, e2=0.00, z=280.00, max_error=173.9262,
 Debug: Iteration 16/11664: c=0.00, e1=0.0000, e2=0.00, z=300.00, max_error=176.1675,
 Debug: Iteration 17/11664: c=0.00, e1=0.0000, e2=0.00, z=320.00, max_error=176.5399,
 Debug: Iteration 18/11664: c=0.00, e1=0.0000, e2=0.00, z=340.00, max_error=170.6779,
 Debug: Iteration 19/11664: c=0.00, e1=0.0000, e2=20.00, z=0.00, max_error=169.3221, b
 Debug: Iteration 20/11664: c=0.00, e1=0.0000, e2=20.00, z=20.00, max_error=173.5863,
 Debug: Iteration 21/11664: c=0.00, e1=0.0000, e2=20.00, z=40.00, max_error=166.4137,
 Debug: Iteration 22/11664: c=0.00, e1=0.0000, e2=20.00, z=60.00, max_error=179.6681,
 Debug: Iteration 23/11664: c=0.00, e1=0.0000, e2=20.00, z=80.00, max_error=176.7312,
 Debug: Iteration 24/11664: c=0.00, e1=0.0000, e2=20.00, z=100.00, max_error=177.6202,
 Debug: Iteration 25/11664: c=0.00, e1=0.0000, e2=20.00, z=120.00, max_error=179.9132,
 Debug: Iteration 26/11664: c=0.00, e1=0.0000, e2=20.00, z=140.00, max_error=178.6290,
 Debug: Iteration 27/11664: c=0.00, e1=0.0000, e2=20.00, z=160.00, max_error=158.6290,
 Debug: Iteration 28/11664: c=0.00, e1=0.0000, e2=20.00, z=180.00, max_error=138.6290,
 Debug: Iteration 29/11664: c=0.00, e1=0.0000, e2=20.00, z=200.00, max_error=133.5236,
 Debug: Iteration 30/11664: c=0.00, e1=0.0000, e2=20.00, z=220.00, max_error=153.5236,
 Debug: Iteration 31/11664: c=0.00, e1=0.0000, e2=20.00, z=240.00, max_error=173.5236,
 Debug: Iteration 32/11664: c=0.00, e1=0.0000, e2=20.00, z=260.00, max_error=172.1849,
 Debug: Iteration 33/11664: c=0.00, e1=0.0000, e2=20.00, z=280.00, max_error=173.9262,
 Debug: Iteration 34/11664: c=0.00, e1=0.0000, e2=20.00, z=300.00, max_error=176.1675,
 Debug: Iteration 35/11664: c=0.00, e1=0.0000, e2=20.00, z=320.00, max_error=176.5399,
 Debug: Iteration 36/11664: c=0.00, e1=0.0000, e2=20.00, z=340.00, max_error=170.6779,
 Debug: Iteration 37/11664: c=0.00, e1=0.0000, e2=40.00, z=0.00, max_error=169.3221, b
 Debug: Iteration 38/11664: c=0.00, e1=0.0000, e2=40.00, z=20.00, max_error=173.5863,
 Debug: Iteration 39/11664: c=0.00, e1=0.0000, e2=40.00, z=40.00, max_error=166.4137,
 Debug: Iteration 40/11664: c=0.00, e1=0.0000, e2=40.00, z=60.00, max_error=179.6681,
 Debug: Iteration 41/11664: c=0.00, e1=0.0000, e2=40.00, z=80.00, max_error=176.7312,
 Debug: Iteration 42/11664: c=0.00, e1=0.0000, e2=40.00, z=100.00, max_error=177.6202,
 Debug: Iteration 43/11664: c=0.00, e1=0.0000, e2=40.00, z=120.00, max_error=179.9132,
 Debug: Iteration 44/11664: c=0.00, e1=0.0000, e2=40.00, z=140.00, max_error=178.6290,
 Debug: Iteration 45/11664: c=0.00, e1=0.0000, e2=40.00, z=160.00, max_error=158.6290,
 Debug: Iteration 46/11664: c=0.00, e1=0.0000, e2=40.00, z=180.00, max_error=138.6290,
 Debug: Iteration 47/11664: c=0.00, e1=0.0000, e2=40.00, z=200.00, max_error=133.5236,
 Debug: Iteration 48/11664: c=0.00, e1=0.0000, e2=40.00, z=220.00, max_error=153.5236,
 Debug: Iteration 49/11664: c=0.00, e1=0.0000, e2=40.00, z=240.00, max_error=173.5236,
 Debug: Iteration 50/11664: c=0.00, e1=0.0000, e2=40.00, z=260.00, max_error=172.1849,
 Debug: Iteration 51/11664: c=0.00, e1=0.0000, e2=40.00, z=280.00, max_error=173.9262,
 Debug: Iteration 52/11664: c=0.00, e1=0.0000, e2=40.00, z=300.00, max_error=176.1675,
 Debug: Iteration 53/11664: c=0.00, e1=0.0000, e2=40.00, z=320.00, max_error=176.5399,
 Debug: Iteration 54/11664: c=0.00, e1=0.0000, e2=40.00, z=340.00, max_error=170.6779,
 Debug: Iteration 55/11664: c=0.00, e1=0.0000, e2=60.00, z=0.00, max_error=169.3221, b
 Debug: Iteration 56/11664: c=0.00, e1=0.0000, e2=60.00, z=20.00, max_error=173.5863,
 Debug: Iteration 57/11664: c=0.00, e1=0.0000, e2=60.00, z=40.00, max_error=166.4137,
 Debug: Iteration 58/11664: c=0.00, e1=0.0000, e2=60.00, z=60.00, max_error=179.6681,
```

```python
# Print the complete errors array
print("\nErrors array:")
print(errors)
```

```
Errors array:
[-2.62107888 -1.61262341  1.43363737  5.0624142   8.03043753  7.67691323
  0.09742122 -5.98070585 -6.22295592 -3.69086836 -0.17838611  3.13592717]
```

```python
# Print the opposition array
print("\nOpposition array:")
print(oppositions)
```

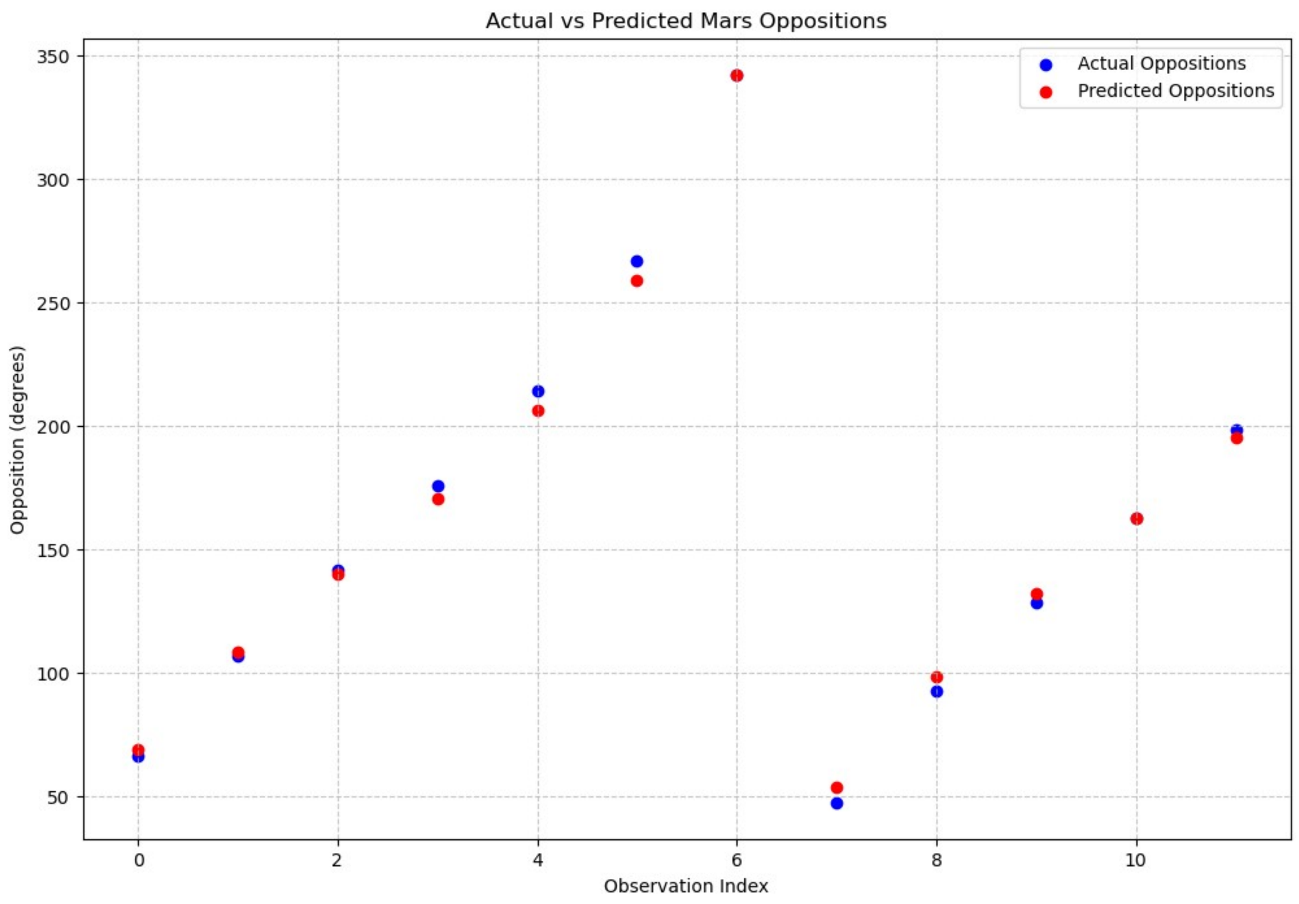
```
Opposition array:
[ 66.47638889 106.925      141.60277778 175.71666667 214.38333333
 266.71666667 342.26666667  47.52777778  92.46666667 128.63333333
 162.45       198.61944444]
```

```
# Calculate predicted oppositions by subtracting errors from actual oppositions
predicted_oppositions = oppositions - errors

# Print the predicted opposition array
print("\nPredicted Opposition array:")
print(predicted_oppositions)

# Calculate the difference between actual and predicted oppositions
opposition_difference = oppositions - predicted_oppositions

# Print the difference
print("\nDifference between actual and predicted oppositions:")
print(opposition_difference)
```

```
Predicted Opposition array:
[ 69.09746777 108.53762341 140.16914041 170.65425247 206.3528958
 259.03975344 342.16924545  53.50848363  98.68962258 132.32420169
 162.62838611 195.48351727]

Difference between actual and predicted oppositions:
[-2.62107888 -1.61262341  1.43363737  5.0624142   8.03043753  7.67691323
  0.09742122 -5.98070585 -6.22295592 -3.69086836 -0.17838611  3.13592717]
```

```
import matplotlib.pyplot as plt

# Create a scatter plot
plt.figure(figsize=(12, 8))

# Plot actual oppositions
plt.scatter(range(len(oppositions)), oppositions, color='blue', label='Actual Oppositions

# Plot predicted oppositions
plt.scatter(range(len(predicted_oppositions)), predicted_oppositions, color='red', label=

# Customize the plot
plt.xlabel('Observation Index')
plt.ylabel('Opposition (degrees)')
plt.title('Actual vs Predicted Mars Oppositions')
plt.legend()

# Add a grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)
```

```
# Show the plot
plt.show()

# Calculate and print the correlation coefficient
correlation = np.corrcoef(oppositions, predicted_oppositions)[0, 1]
print(f"\nCorrelation coefficient between actual and predicted oppositions: {correlation:
```



```
Correlation coefficient between actual and predicted oppositions: 0.9990
```