Graduate Computational Physics — Problem Set #5

Prithivi Rana. (09/26)

Problem 1. *Exercise 5.17 of Newman: The Gamma function*

Solution. This problem tasks us with writing a Python function that can calculate the gamma function accurately. We are presented with multiple subsections of this problem whose solutions ultimately lead to the solution of the gamma function.

As stated in the problem, the gamma function $\Gamma(a)$ is defined by the integral:

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} dx \tag{1}$$

a) The first part of the problem asks us to make a graph of the value of the integrand $x^{a-1}e^{-x}$ as a function of x for discrete values of a. For this, I first wrote an `integrand(x,a)` function that takes values of x and a as input parameters. This function returns the value of the integrand when evaluated at the user-defined values for x and a.

Next, I made three curves by setting a = 2,3, and 4 and evaluating the integrand over the range x = [0,5]. This graph is plotted in Figure 1. All three curves start at zero, rise to a maximum, and then decay.

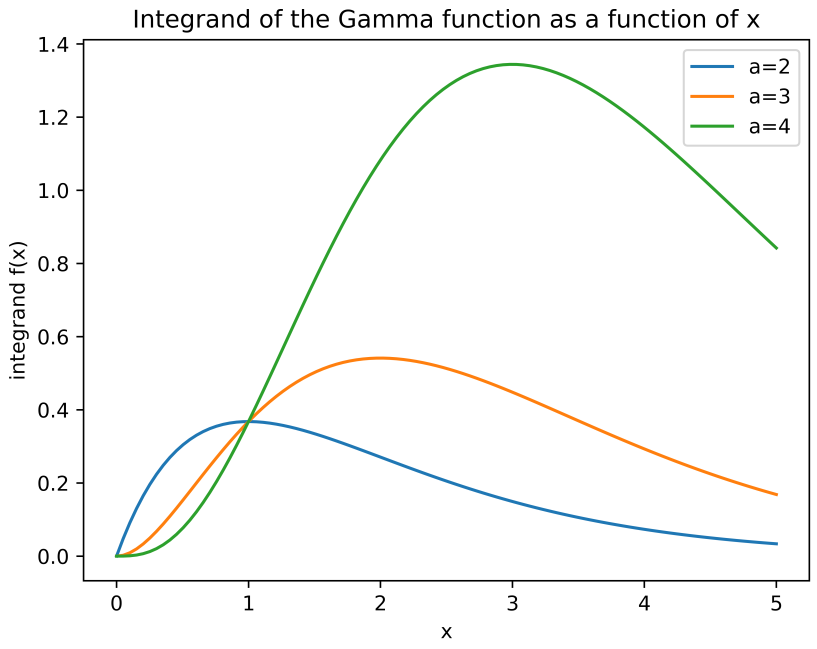


Figure 1: Integrand of Equation 1 evaluated for different a values

b) Part b asks us to prove analytically that the maximum falls at $x = a - 1$.

To find the stationary points, we set the first derivative to 0:

$$\begin{aligned}
0 &= \frac{d}{dx}(x^{a-1}e^{-x}) \\
&= \big((a-1)\cdot x^{a-2}\cdot e^{-x} - x^{a-1}\cdot e^{-x}\big) \\
&= e^{-x}((a-1)\cdot x^{a-2} - x^{a-2}\cdot x) \\
&= (e^{-x}\cdot x^{a-2})\cdot((a-1)-x)
\end{aligned}$$

We have stationary points at:

$$\begin{aligned}
& x = 0 \\
\textit{and,}\qquad & x = (a-1)
\end{aligned}$$

Looking at Figure 1, we can analyze that x=(a-1) is a maximum and x=0 seems to be a saddle point.

c) In part c, we consider how to rescale the integral from 0 to inf to a finite range while still evaluating the integral correctly. We saw in Figure 1 that most of the area under the integrand falls under the maximum. So for an accurate evaluation of the integral, we need to capture this part of the integral properly. For this we use the following change of variable given in Eq. (5.69) of Mark Newman's *Computational Physics*:,

$$z = \frac{x}{c+x}$$

Setting $x = c$ gives $z = \frac{1}{2}$. So, the appropriate choice for the parameter c would be $c = (a-1)$. This puts the peak of the integrand for the gamma function at $z = \frac{1}{2}$

$$\begin{aligned}
z &= \frac{x}{c+x} \\
x &= z\cdot c + z\cdot x \\
x\cdot(1-z) &= z\cdot c \\
x &= c\cdot\left(\frac{z}{1-z}\right) \\
x &= (a-1)\cdot\left(\frac{z}{1-z}\right)
\end{aligned}$$

I implemented this change of variables in my program as a function called `func_rescale(z,a)`. The code is listed below:

```
def func_rescale(z=None, a=2):

    x = ((a-1)*z)/(1-z)
    dx = (a-1)/((1-z)**2)
    return (dx * integrandest(x,a))
```

Where `integrandest(x,a)` is the function with the reworked integrand I will derive for part d.

d) Part d asks us to derive an alternate expression for the integrand to avoid numerical underflow and overflow in the x^{a-1} and e^{-x} terms.

Setting $x^{a-1} = e^{(a-1)\ln x}$ gives us the following expression for the gamma function:

$$\Gamma(a) = \int_0^\infty e^{(a-1)\ln x - x} dx$$

.

Our original expression was susceptible to underflow since x^{a-1} could grow very small for large values of a and small values of x, and e^{-x} could lead to problems with extreme values of x. Taking a product of these two expressions where we are multiplying a very large number by a very small one could present inaccurate results.

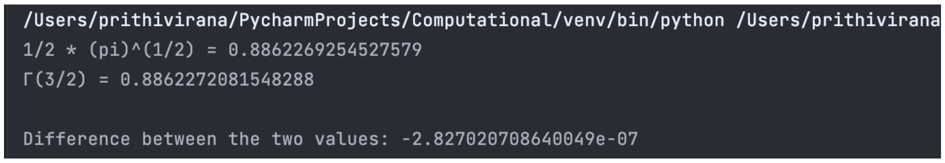
The new expression is better since the entire expression is combined into one exponential term. So, this reduces the problem of taking products of very large and very small values. Since $\ln x$ grows slower than x, it also helps avoid extreme values that would otherwise be encountered in our old expression.

e) Part e tasks us with writing a new function `gamma(a)` with our change of variables, value of c, and alternate integrand we got in the previous subsections.

I implemented this in my program as follows:

```
def gamma(a,N=50):
    (gauss_integral, _)=integrate.fixed_quad(func_rescale,0,1,args=(a,),n=N)
    return gauss_integral
```

This function calculates the Gamma function using Gaussian quadrature with the number of sample points set to 50 by default. I then tested the function for $\Gamma(\frac{3}{2}$ and compared it with $\frac{1}{2}\sqrt{\pi} \simeq 0.886$. The output is presented in Figure 2.



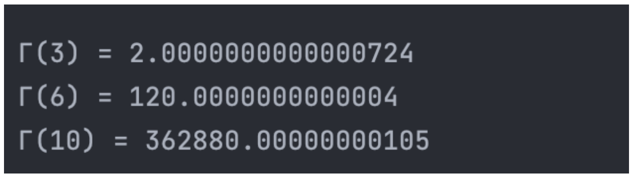
```
/Users/prithivirana/PycharmProjects/Computational/venv/bin/python /Users/prithivirana
1/2 * (pi)^(1/2) = 0.8862269254527579
Γ(3/2) = 0.8862272081548288

Difference between the two values: -2.827020708640049e-07
```

Figure 2: Testing Gamma function

f) Finally, we are tasked with using our Python function to calculate $\Gamma(3), \Gamma(6)$, and $\Gamma(10)$. We should get answers closely equal to $2! = 2$, $5! = 120$, and $9! = 362880$ since $\Gamma(n)$ is equal to the factorial of (n - 1) for integer values of n.

The output of these calculations is in Figure 3. As we can see, our values for Γ are in close agreement with the (n-1)! values.



```
Γ(3) = 2.0000000000000724
Γ(6) = 120.0000000000004
Γ(10) = 362880.00000000105
```

Figure 3: Gamma function evaluations

Problem 2. *This problem demonstrates an application of linear algebra to signal analysis. Download a "signal" as a function of time from this file. Assume that all the measurements have the same uncertainty, with a standard deviation of 2.0 in the signal units.*

Solution.

a) First, I read the data from *signal.dat* file into a NumPy array using NumPy's `genfromtxt()` function. I then sorted the data based on the time column to help our calculations. This data was then plotted using matplotlib's `plot` function which can be seen in Figure 4.

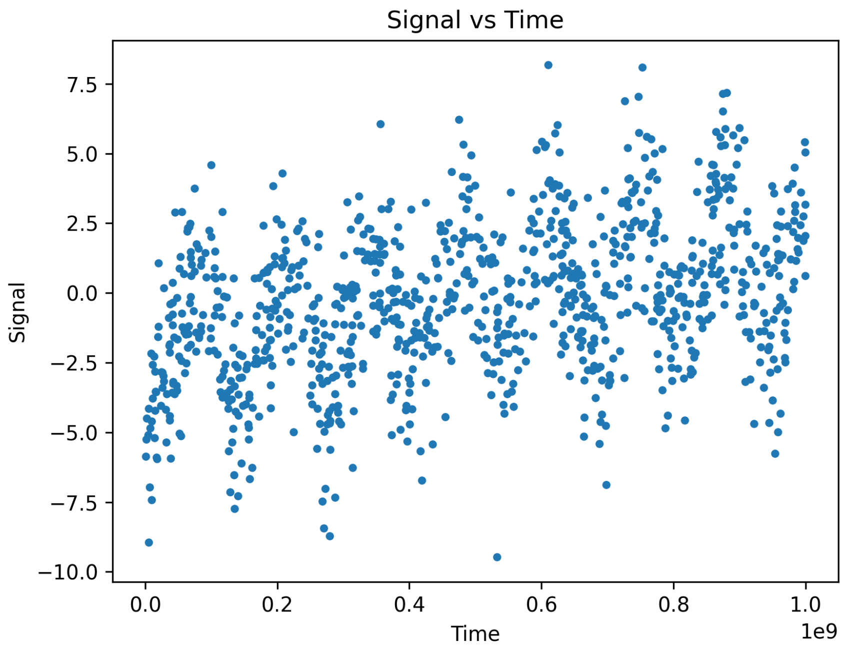


Figure 4: Scatter plot from *signal.dat* data

b) Use the SVD technique to find the best third-order polynomial fit in time to the signal. Pay attention to the scaling of the independent variable (time).

Before fitting a curve to our data, I scaled the independent variable (time) using the mean and standard deviation as follows:

$$time' = \frac{time - \bar{time}}{\sigma_{time}}$$

Before rescaling, the difference between the highest time value and the lowest time value was 9.991445108432575e+08, whereas after rescaling, it was 3.4288541344535126e+00. The large difference before rescaling would have made the condition number of our design matrix very large and make our SVD solution unstable. Rescaling helps avoid this and reduces round-off errors.

After rescaling, I wrote a `svd(y,x,n)` function that takes in y (dependant variable), x (independent variable), and n (order of the fitting polynomial) values. This function was implemented in the standard way for getting an SVD solution as presented in the class notes.

I used this function to find the best third-order polynomial fit in time to the signal (Figure 5). As evident in the plot, a third-order polynomial doesn't do a great job of fitting the curve and completely misses the periodicity and several trends of the data.

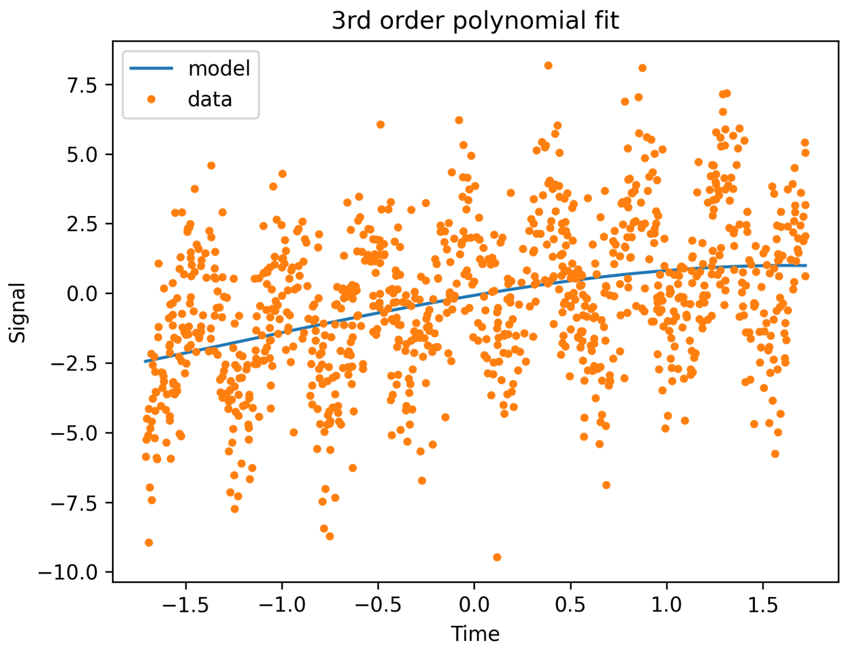


Figure 5: 3^{rd} order polynomial fit for signal vs time

c) Calculate the residuals of the data with respect to your model. Argue that this is not a good explanation of the data given what you know about the measurement uncertainties.

I then calculated the residuals of the data with respect to the model which can be seen in Figure 7.

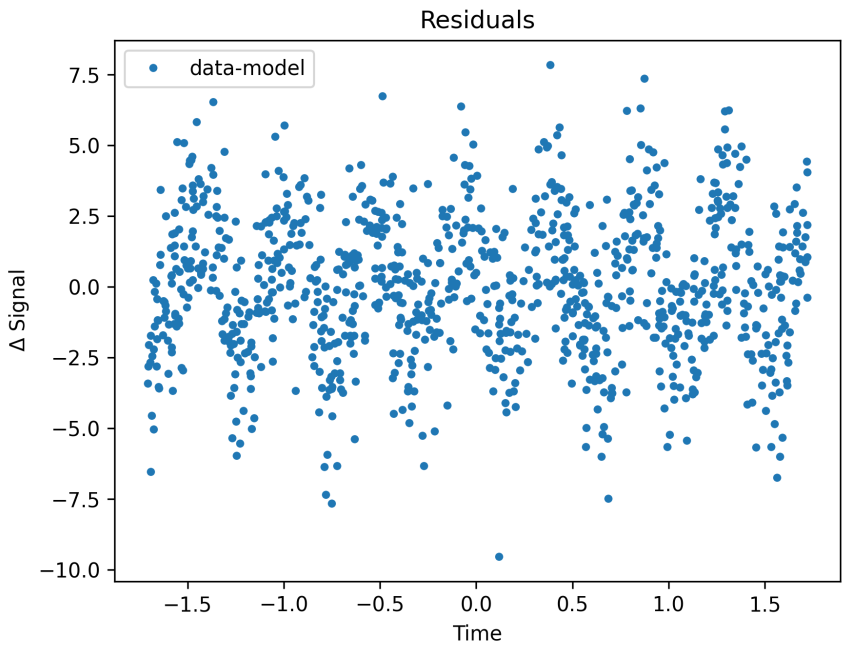
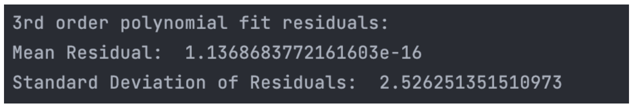


Figure 6: Residuals from the 3^{rd} order polynomial fit

Our fit does a very bad job of describing the data and our residual values have a standard deviation of 2.526251351510973 which is greater than the standard deviation in the data itself! This tells us that we need to consider a higher polynomial or a different approach.



```
3rd order polynomial fit residuals:
Mean Residual:  1.1368683772161603e-16
Standard Deviation of Residuals:  2.526251351510973
```

Figure 7: Mean and standard deviation of the residuals

d) Try a much higher-order polynomial. Is there any reasonable polynomial you can fit that is a good explanation of the data? Define "reasonable polynomial" as whether the design matrix has a viable condition number.

Although increasing the order of the polynomial can give us increasingly better descriptions of the data, it can start to fail after a certain order. Using too high of an order can make our design matrix ill-conditioned, leading to instability in our solution. So I decided to check the condition numbers for different orders of polynomials and compare it against machine precision for `np.float` values.

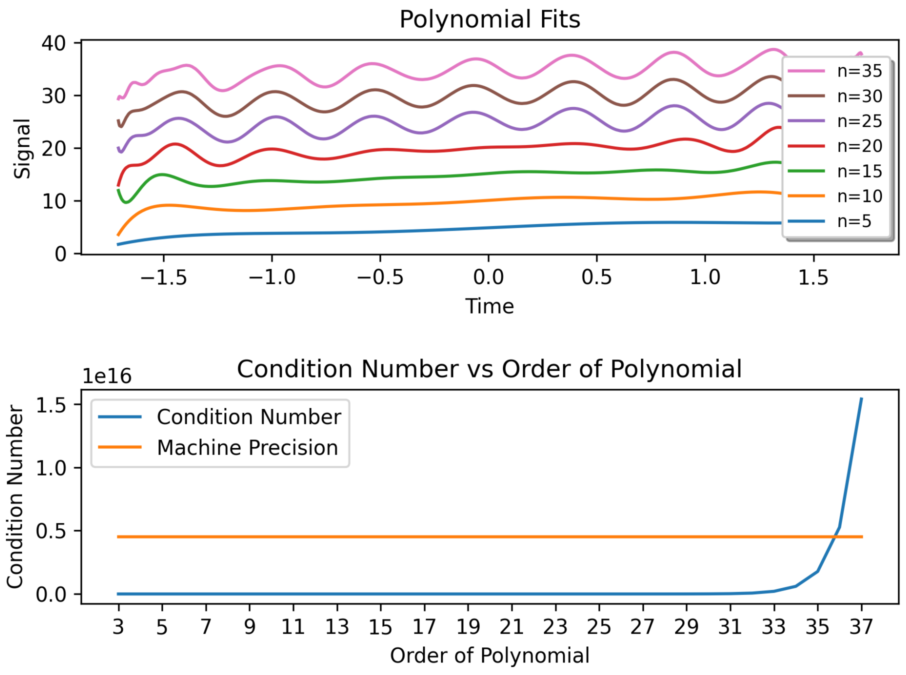
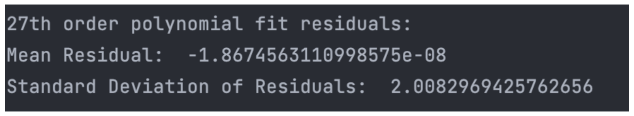


Figure 8: Trying higher order polynomials

Looking at the first plot in Figure 8, higher polynomial fits do a better job of capturing the intricacies in the data. However, the second plot shows us that the condition number shoots past machine precision above a 35^{th} degree polynomial. Although the condition numbers for polynomials around 35^{th} order are less than machine precision, that doesn't necessarily mean that they are good choices for fitting our data. Higher polynomials can also cause overfitting and can be susceptible to noise and random fluctuations in the data.

I decided to use a 27^{th} order polynomial (Figure 10) since it seemed to represent the periodicity of the data and had a condition number below machine precision. Checking the residuals showed that the standard deviation of our residuals was 2.0082969425762656 which is close to the standard deviation of our source data.



```
27th order polynomial fit residuals:
Mean Residual:  -1.8674563110998575e-08
Standard Deviation of Residuals:  2.0082969425762656
```

Figure 9: Mean and Standard Deviation for 27^{th} order polynomial fit

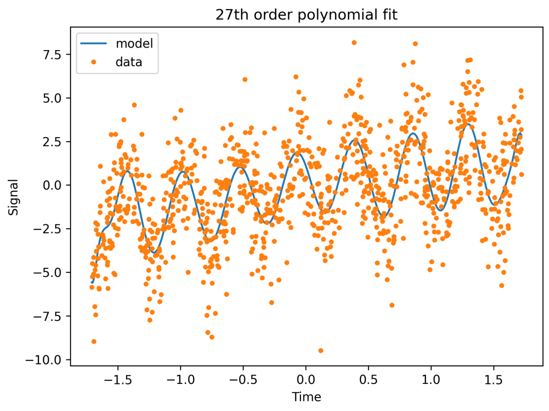


Figure 10: 27th order polynomial fit

e) Try fitting a set of sin and cos functions plus a zero-point offset. As a Fourier series does, use a harmonic sequence with increasing frequency, starting with a period equal to half of the time span covered. Does this model do a "good job" explaining the data? Are you able to determine a typical periodicity in the data? You may have noticed a periodicity from the first plot.

In order to fit a set of sin and cos functions similar to a Fourier series, I wrote a new function `svd_harmonics(signal,time,n)` implemented as follows:

```
def svd_harmonics(signal,time,n):

    period = (np.max(time) - np.min(time)) / 2

    # Design matrix with sin and cos functions (2n columns)
    # and a zero offset point (1 column)
    A = np.zeros((len(time), 2 * n + 1))

    #zero offset point in column 0
    A[:, 0] = 1/2

    # Insert alternating sin and cos functions in columns 1->2n+1
    for i in range(1, n + 1):
        A[:, 2 * i - 1] = np.sin(2 * np.pi * i * time / period)
        A[:, 2 * i] = np.cos(2 * np.pi * i * time / period)

    (u, w, vt) = np.linalg.svd(A, full_matrices=False)

    ainv = vt.transpose().dot(np.diag(1. / w)).dot(u.transpose())

    coeffs = ainv.dot(signal)
    return (A.dot(coeffs))
```

Here, the function has the form:

$$f(t) = \frac{1}{2}A_0 + \sum_{i=1,iodd}^{2n-1} A_i \sin\frac{2\pi it}{T} + \sum_{j=2,jeven}^{2n} B_j \cos\frac{2\pi jt}{T}$$

where, `T` is the period which is equal to half the time covered.

Checking the condition numbers to see where it passes machine precision gives the following plot:

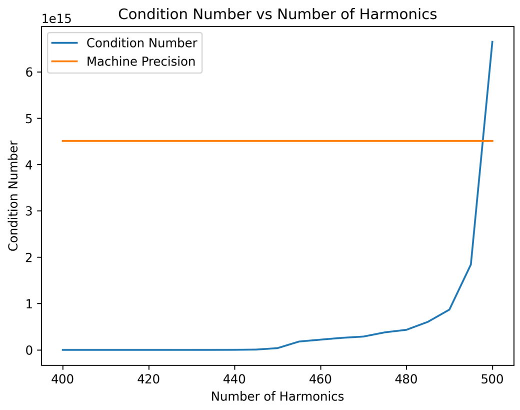


Figure 11: Condition numbers for different numbers of harmonics

We can see that anything below 490 would give a valid condition number. However, we run the risk of overfitting again when using too high of a number. Starting off with n=20 gave a fit similar to the 27th order polynomial and had a standard deviation of 2.435340722830597.

Setting n=300 gives us Figure 12 and smoothing our curve using NumPy's `convolve()` gives us Figure 13.

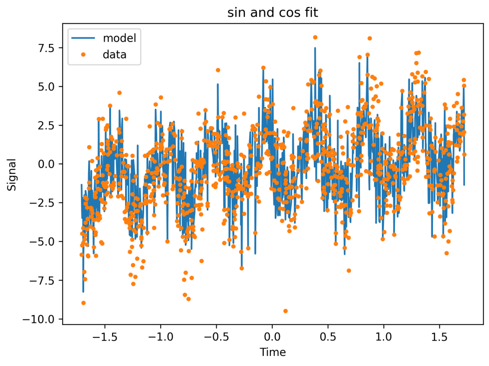


Figure 12: Fitting a set of Sin and Cos functions

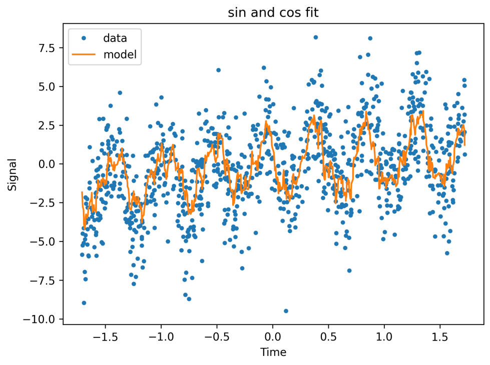
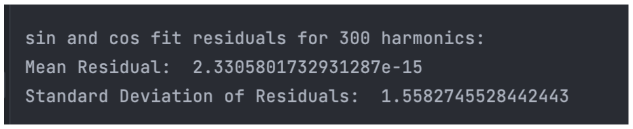


Figure 13: Smoothing the curve



```
sin and cos fit residuals for 300 harmonics:
Mean Residual:  2.3305801732931287e-15
Standard Deviation of Residuals:  1.5582745528442443
```

Figure 14: Mean and standard deviation of residuals

This method does a better job of explaining the data and we can better observe the periodicity in the data. The standard deviation of the residuals of this method was 1.5582745528442443 which is less than that of the original data.