Assignment 4: Fourier Approximations

J.Phani Jayanth - EE19B026

March 11, 2021

1 Abstract

This week's Python Assignment involves the following learning outcomes:

- Fitting the functions e^x and *cos(cos(x))* using Fourier Series
- Using the built-in integrator in Python in calculation of Fourier coefficients
- Finding the coefficients for the best fit of the above functions to Fourier series, using Least Squares approach
- Comparing the coefficients obtained from Direct Integration and Least Square approach

2 Introduction

Fourier series approximation is used to approximate a periodic function as a sum of sine and cosine functions. General form of Fourier series is as follows:

$$f(x) = a_0 + \sum_{n=1}^{\infty}(a_n cos(nx) + b_n sin(nx))$$

The coefficients are obtained as follows:

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(x)dx$$

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x)cos(nx)dx$$

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)sin(nx)dx$$

3 Plotting the Exponential and Coscos functions

We define the functions e^x and *cos(cos(x))* in our code as follows:

```
def coscos(x):
        return np.cos(np.cos(x))

x_axis = np.linspace(-2*np.pi,4*np.pi, 10000)

def func(n):
    if n==0:
        return np.exp, 'exp(x)'
    elif n==1:
        return coscos, 'cos(cos(x))'
```

Here, the function *func()* is defined such that when it takes 0 and 1 as arguments, it returns e^x and *cos(cos(x))* respectively. These functions are then plotted from [-2π,4π] by taking 10000 points in the interval of [-2π,4π].

Since exponential function increases rapidly (Figure 1), it is plotted on a semilog Y-axis i.e., *log(*e^x*)* vs x is plotted as shown in Figure 2.

We can clearly see from the plots of e^x and *cos(cos(x))* that cos(cos(x)) is periodic, but e^x isn't. Since Fourier series is periodic in nature, for cos(cos(x)) we can expect the same function to be generated through Fourier series also. But approximation of e^x using Fourier series repeats itself for every interval of 2π. Periodic extenstion of e^x is shown in Figure 4.
The code for this is as follows:

```
for n in range(0,2):
        #Plotting the functions against x
        plt.plot(x_axis,func(n)[0](x_axis),'r')
        plt.title(f'{func(n)[1]} vs x')
        plt.grid()
        plt.xlabel('x    $\longrightarrow$')
        plt.ylabel(f'{func(n)[1]}    $\longrightarrow$')
        plt.show()
        #Plotting the functions against x in semilog-Y scale(for exp(x))
        plt.semilogy(x_axis,func(n)[0](x_axis))
        plt.title(f'{func(n)[1]} vs x: Semilogy')
        plt.grid()
        plt.xlabel('x    $\longrightarrow$')
        plt.ylabel(f'{func(n)[1]}    $\longrightarrow$')
        plt.show()
```

The plots are as follows:

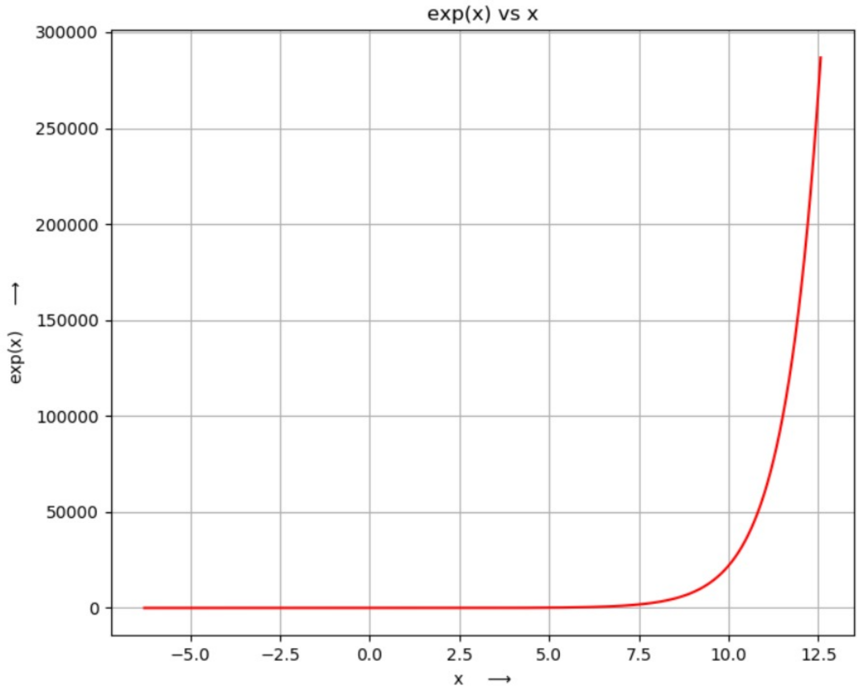

Figure 1: e^x vs x

```
# Periodic extension of exp(x)
plt.plot(x_axis,func(0)[0](x_axis%(2*np.pi)),'m')
plt.title(f'Periodic extenstion of {func(0)[1]} vs x')
plt.grid()
plt.xlabel('x    $\longrightarrow$')
plt.ylabel(f'Expected Fourier {func(0)[1]}    $\longrightarrow$')
plt.show()
```

4 Obtaining the Fourier Coefficients

For finding the Fourier coefficients, we first define two new functions u and v as follows:

```
# Defining functions f(x)coskx and f(x)sinkx for finding coefficients
def u(x,n,k):
        return np.cos(k*x)*func(n)[0](x)
def v(x,n,k):
        return np.sin(k*x)*func(n)[0](x)
```

We make use of the **integrate.quad()** function from *Scipy* library to in-

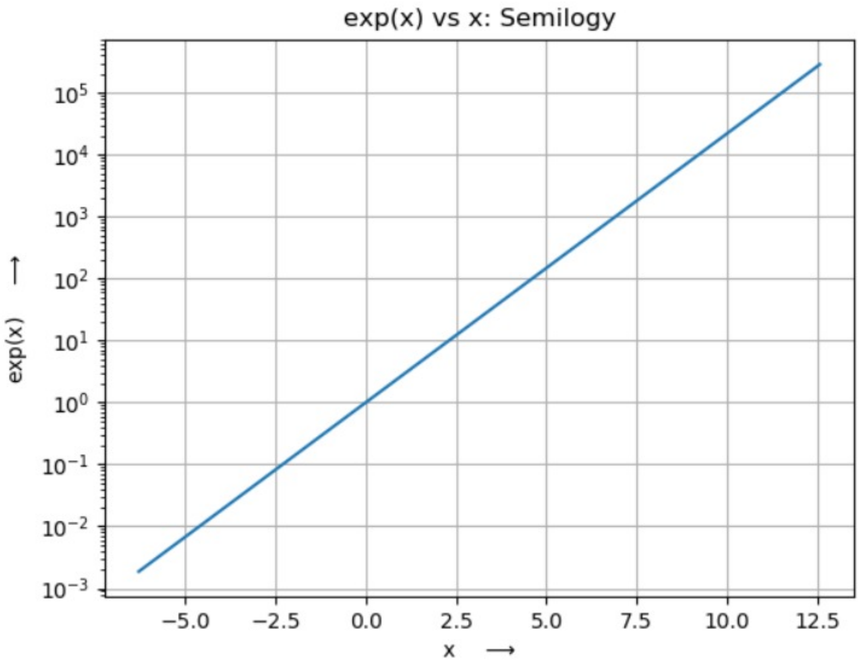

Figure 2: e^x in log-scale vs x

tegrate the above defined functions, and obtain the Fourier coefficients as follows:

```python
# Getting a0:
coeff = [integrate.quad(func(n)[0],0,2*np.pi)[0] / (2*np.pi)]
# Getting ak and bk:
for k in range(1,26):
        coeff.append(integrate.quad(u,0,2*np.pi,args=(n,k))[0] / np.pi)
        coeff.append(integrate.quad(v,0,2*np.pi,args=(n,k))[0] / np.pi)
```

We then plot the following plots:

- Magnitude of Fourier coefficients vs Index of coefficient(n)
- Fourier coefficients on a semilog-Y axis vs n
- loglog plot of Fourier coefficients vs n

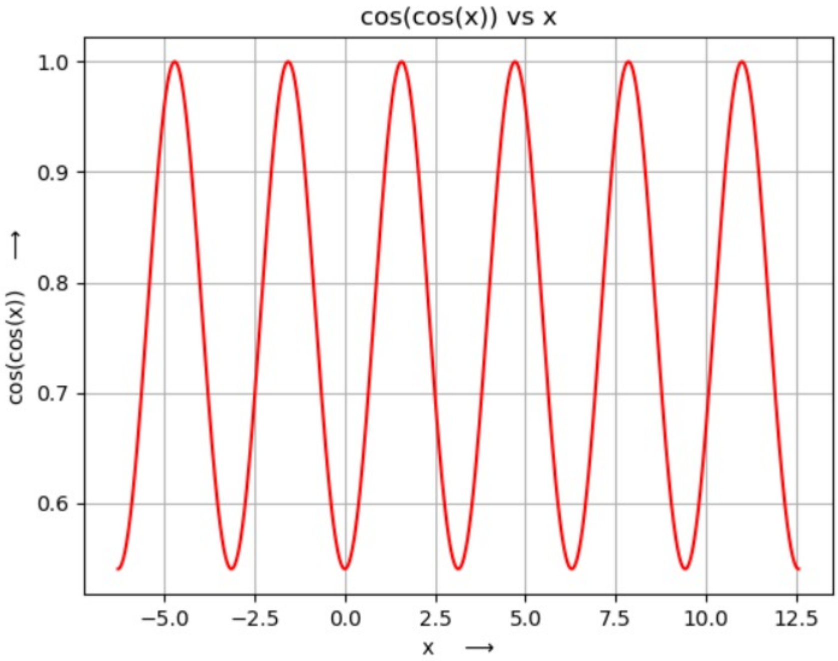

Figure 3: cos(cos(x)) vs x

```
for n in range(0,2):
    # Getting a0:
    coeff = [integrate.quad(func(n)[0],0,2*np.pi)[0] / (2*np.pi)]
    # Getting ak and bk:
    for k in range(1,26):
        coeff.append(integrate.quad(u,0,2*np.pi,args=(n,k))[0] /
        coeff.append(integrate.quad(v,0,2*np.pi,args=(n,k))[0] /
    #Plotting the magnitude of fourier coefficients vs n:
    plt.plot(0,coeff[0],'bo',label='a\N{SUBSCRIPT ZERO}')
    plt.plot(np.arange(1,26,1),np.abs(coeff[1::2]),'ro',label='a')
    plt.plot(np.arange(1,26,1),np.abs(coeff[2::2]),'yo',label='b')
    plt.title(f'{func(n)[1]}: Magnitude of Fourier coefficients vs n'
    plt.xlabel('n    $\longrightarrow$')
    plt.ylabel('Magnitude of Fourier Coefficients   $\longrightarrow$
    plt.grid()
    plt.legend()
    plt.show()
    #Plotting the coefficients on log scale against 'n'
    plt.semilogy(0,np.abs(coeff[0]),'bo')
    plt.semilogy(np.arange(1,26,1),np.abs(coeff[1::2]),'ro')
    plt.semilogy(np.arange(1,26,1),np.abs(coeff[2::2]),'yo')
```

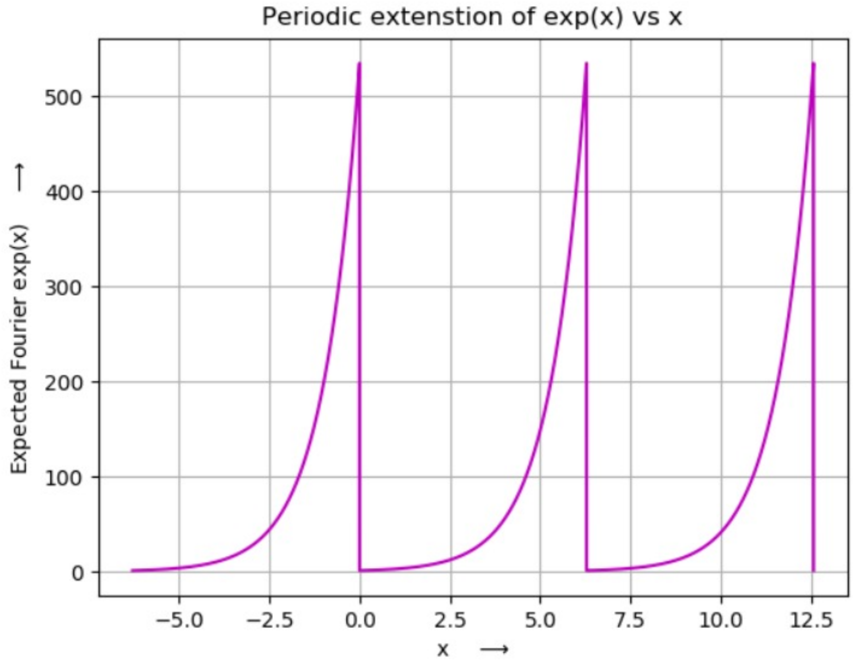

Figure 4: Periodic Extenstion of e^x

```
plt.title(f'{func(n)[1]}: Semilogy - Fourier Coefficients vs n')
plt.xlabel('n    $\longrightarrow$')
plt.ylabel('log(Fourier coefficients)   $\longrightarrow$')
plt.grid()
plt.show()
#Plotting the coefficients on log scale against 'n', also in log
plt.loglog(0,np.abs(coeff[0]),'bo')
plt.loglog(np.arange(1,26,1),np.abs(coeff[1::2]),'ro')
plt.loglog(np.arange(1,26,1),np.abs(coeff[2::2]),'yo')
plt.title(f'{func(n)[1]}: Loglog Plot')
plt.xlabel('log(n)    $\longrightarrow$')
plt.ylabel('log(Fourier coefficients)   $\longrightarrow$')
plt.grid()
plt.show()
```

Question 3

- If you did Q1 correctly, the b_n coefficients in the second case should be nearly zero. Why does this happen?
  Reason: b_n coefficients of cos(cos(x)) are nearly zero, as shown in Figure 6. This is as expected, because the function cos(cos(x)) is an

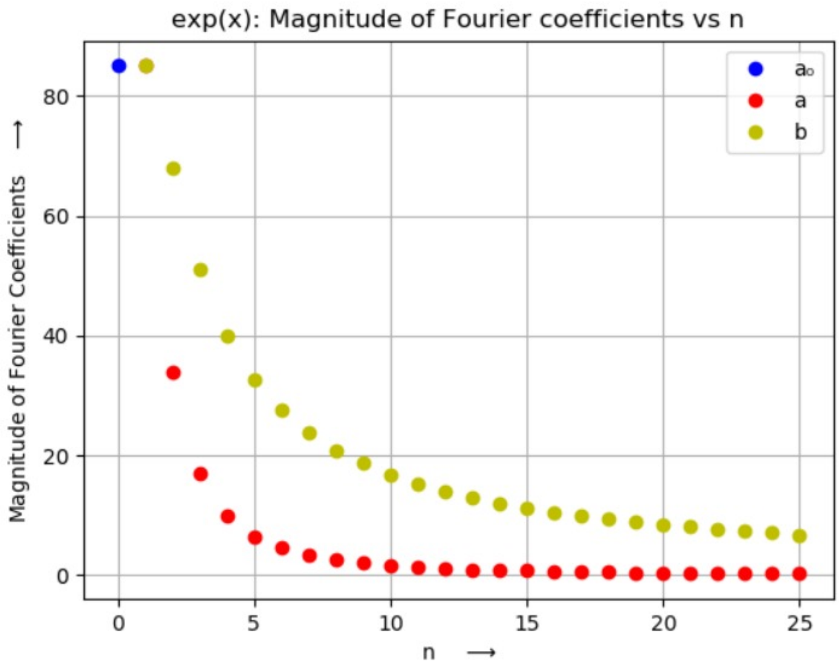

Figure 5: e^x - Fourier coefficients vs n

even function with a period of 2π, and hence doesn't contain any sine terms.

- In the first case, the coefficients do not decay as quickly as the coefficients for the second case. Why not?
  Reason: e^x has infinite harmonics, and it is also not periodic, so we expect it to have higher number of non-zero coefficients, and hence to decay slowly. In contrast to this, cos(cos(x)) is almost sinusoidal, and hence has fewer non-zero cos-harmonics. Therfore, it decays fastly.

- Why does loglog plot in Figure 9 look linear, wheras the semilog plot in Figure 10 looks linear?
  Reason: The Fourier coefficients of e^x vary as $\frac{1}{n^2}$ and therfore, the coefficients and their corresponding indices(n) are proportional in log-scale. Hence, we get a linear loglog plot. Similarly, Fourier coefficients of cos(cos(x)) vary as $\frac{1}{n}$. Therefore, we obtain a linear plot with semilog-Y axis.

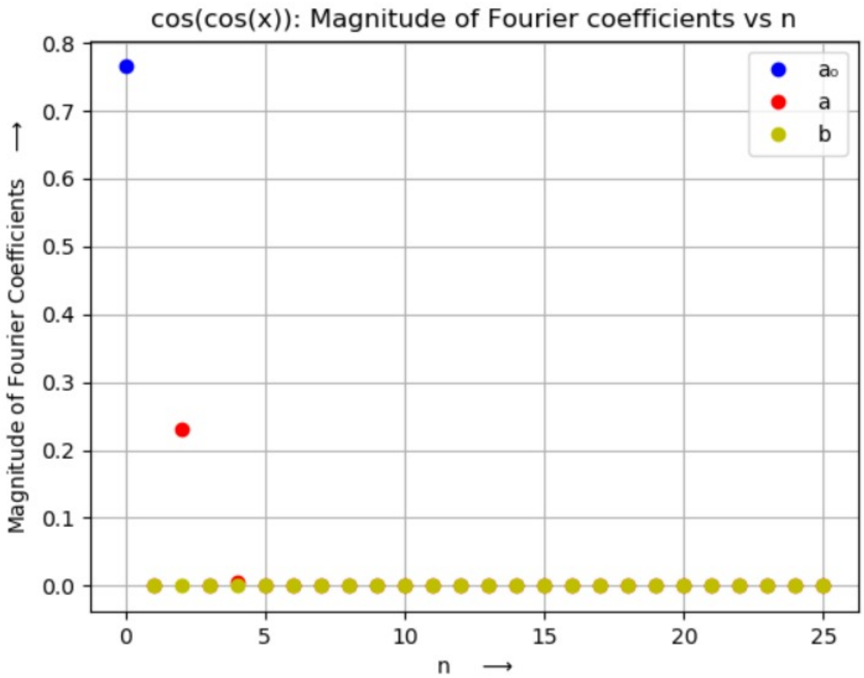

Figure 6: cos(cos(x)) - Fourier coefficients vs n

5 Least Squares method - Finding coefficients for best fit

Instead of employing direct integration to obtain the coefficients, we can use the *"Least Squares method"* to estimate the coefficients. For this, we create the following matrices:

$$\text{A} = \begin{pmatrix} 1 & cosx_1 & sinx_1 & ... & cos25x_1 & sin25x_1 \\ 1 & cosx_2 & sinx_2 & ... & cos25x_2 & sin25x_2 \\ ... & ... & ... & ... & ... & ... \\ 1 & cosx_{400} & sinx_{400} & ... & cos25x_{400} & sin25x_{400} \end{pmatrix} \text{ and}$$

$$\text{b} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ ... \\ f(x_{400}) \end{pmatrix}$$

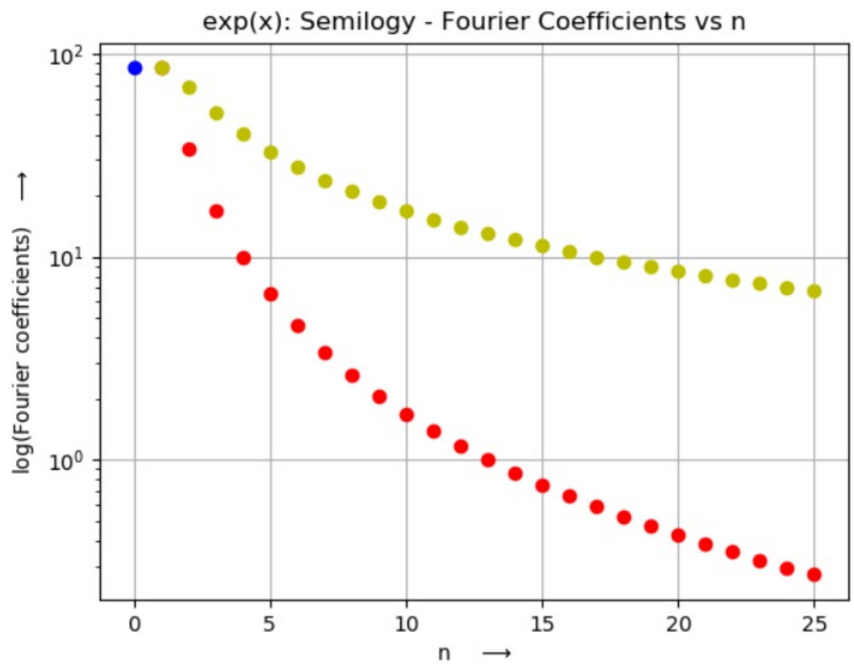

Figure 7: e^x - Fourier coefficients on semilog-Y axis vs n

Let the coefficient matrix be c = $\begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ \ldots \\ a_{25} \\ b_{25} \end{pmatrix}$

We are aware of the relation: Ac = b. We solve this equation using the *lstsq()* function to obtain the coefficient matrix c, for the best fit of the functions e^x and cos(cos(x)).

```
# To find the coefficients using Least Squares approach:
x_vector = np.linspace(0,2*np.pi,401)[:-1] # entries from 0 to 2pi
# List b contains function values at each x in x_vector
b = func(n)[0](x_vector)
#Forming the A matrix
A = np.transpose(np.array([1]*400))

for k in range(1,26):
```

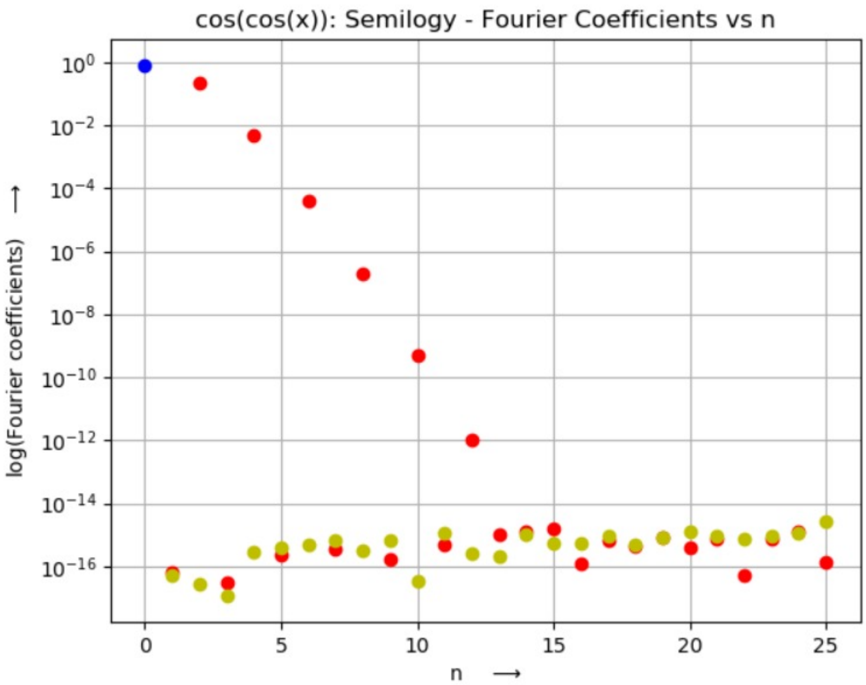

Figure 8: cos(cos(x)) - Fourier coefficients on semilog-Y axis vs n

```
        A=np.c_[A,np.cos(k*x_vector)]
        A=np.c_[A,np.sin(k*x_vector)]

#Solving for matrix c in Ac = b:
try:
        sol = np.linalg.lstsq(A,b)[0]
except Exception:
        print("The equation Ac = b couldn't be sovled!")
        exit()
```

Least square method solves the 400 equations using lstsq() function, and the coeffients obtained are plotted against 'n'(in green circles). Also, in the same plot, we plot the coefficients generated by direct integration(in red circles).
The following are plotted:

- Coefficients obtained from Least Squares approach against indices of coefficients(n), with semilog-Y axis - Figure 12,15
- loglog plot of Coefficients obtained from Least Squares approach against indices of coefficients(n) - Figure 13,16

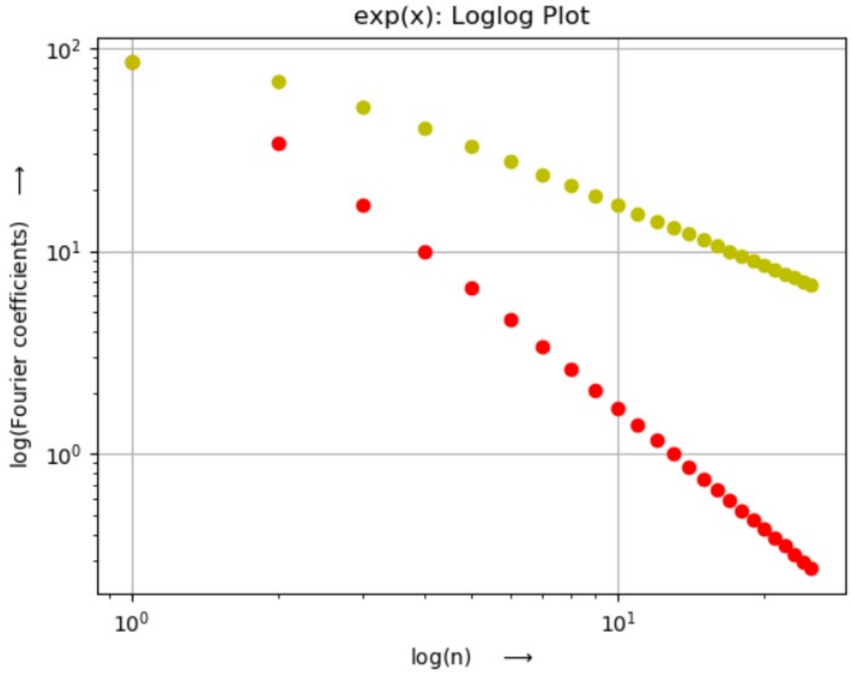

Figure 9: e^x - loglog plot of Fourier coefficients vs n

```
#Plotting the magnitude of coefficients obtained from Least s
plt.plot(0,sol[0],'bo',label='a\N{SUBSCRIPT ZERO}')
plt.plot(np.arange(1,26,1),np.abs(sol[1::2]),'ro',label='a')
plt.plot(np.arange(1,26,1),np.abs(sol[2::2]),'yo',label='b')
plt.title(f'{func(n)[1]}: Magnitude of coefficients vs n (Least Squares a
plt.xlabel('n     $\longrightarrow$')
plt.ylabel('Magnitude of coefficients    $\longrightarrow$')
plt.grid()
plt.legend()
plt.show()
#Plotting the coefficients obtained from Least squares approach in log sc
plt.semilogy(0,np.abs(coeff[0]),'ro')
plt.semilogy(np.arange(1,26,1),np.abs(coeff[1::2]),'ro')
plt.semilogy(np.arange(1,26,1),np.abs(coeff[2::2]),'ro',label='Direct Int
plt.semilogy(0,np.abs(sol[0]),'go')
plt.semilogy(np.arange(1,26,1),np.abs(sol[1::2]),'go')
plt.semilogy(np.arange(1,26,1),np.abs(sol[2::2]),'go',label='Least Square
plt.title(f'Semilogy: {func(n)[1]} - Coefficients obtained by both method
plt.xlabel('n     $\longrightarrow$')
plt.ylabel('log(Coefficients)    $\longrightarrow$')
plt.legend()
```

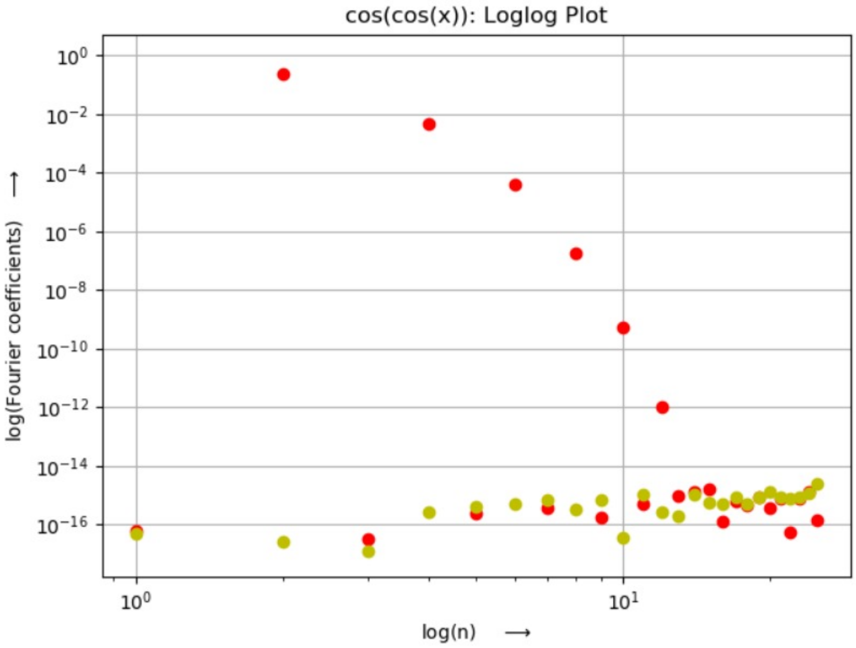

Figure 10: cos(cos(x)) - loglog plot of Fourier coefficients vs n

```
plt.grid()
plt.show()
#Plotting the loglog plot of coefficients obtained from Least squares app
plt.loglog(0,np.abs(coeff[0]),'ro')
plt.loglog(np.arange(1,26,1),np.abs(coeff[1::2]),'ro')
plt.loglog(np.arange(1,26,1),np.abs(coeff[2::2]),'ro',label='Direct Integ
plt.loglog(0,np.abs(sol[0]),'go')
plt.loglog(np.arange(1,26,1),np.abs(sol[1::2]),'go')
plt.loglog(np.arange(1,26,1),np.abs(sol[2::2]),'go',label='Least Squares 
plt.title(f'{func(n)[1]}: log(Coefficients) vs log(n) - Direct Integratio
plt.xlabel('log(n)    $\longrightarrow$')
plt.ylabel('log(Coefficients)   $\longrightarrow$')
plt.grid()
plt.legend()
plt.show()
```

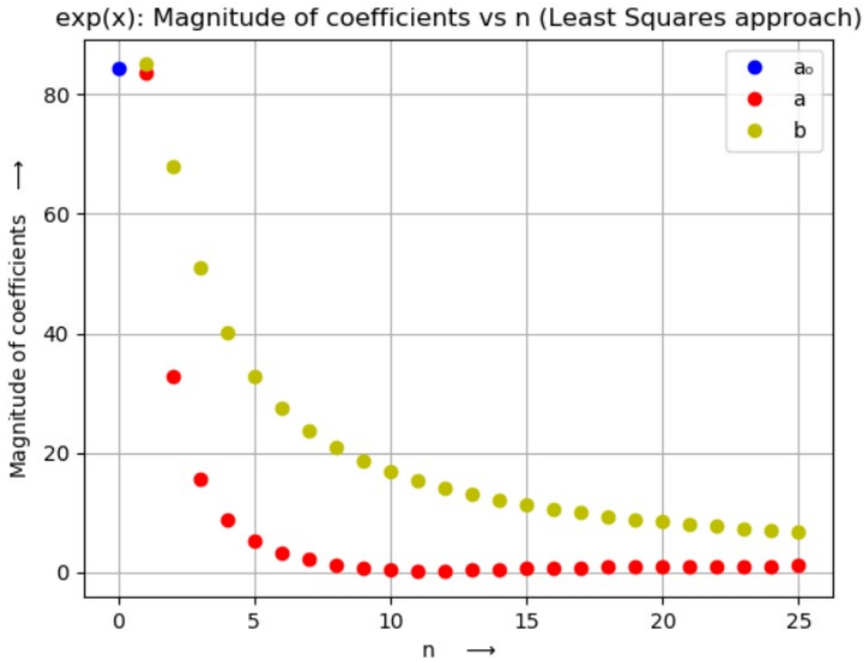

Figure 11: e^x - Magnitude of coefficients vs n (Least Squares method)

6 Comparing the coefficients obtained from both methods and finding the largest deviation

The coefficients obtained through Least Squares approach and Direct Integration are subtracted, and the absolute error/difference is plotted for both e^x and cos(cos(x)) - Figures 17 and 18. The error matrix is found to be small for both the functions (although note that the deviations in the case of cos(cos(x)) are of the order of 1e-15), and the respective largest deviations are found to be the following:

- The largest deviation for e^x = 1.33
- The largest deviation for cos(cos(x)) = 2.60e-15

This is done with the help of *amax()* function available in the *numpy* library, after calculating the absolute difference using *abs()* function of *numpy*.

```
#Calculating the maximum difference in calculating coefficients from both
print(f'For {func(n)[1]}: Largest deviation = {np.amax(np.abs(sol-coeff))
#Plotting absolute difference of coefficients obtained from both methods
plt.plot(0,np.abs(coeff[0]-sol[0]),'bo',label='a\N{SUBSCRIPT ZERO}')
plt.plot(np.arange(1,26,1),np.abs(coeff[1::2]-sol[1::2]),'ro',label='a')
```

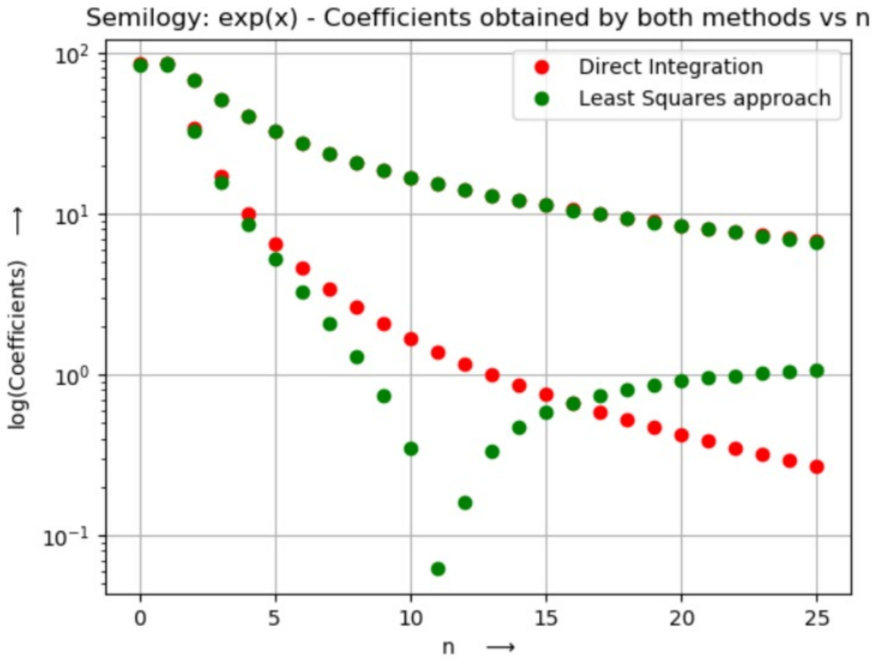

Figure 12: e^x - Magnitude of coefficients vs n (Both methods) with semilog-Y axis

```
plt.plot(np.arange(1,26,1),np.abs(coeff[2::2]-sol[2::2]),'yo',label='b')
plt.xlabel('n    $\longrightarrow$')
plt.ylabel('Deviation    $\longrightarrow$')
plt.title(f'{func(n)[1]}: Absolute Difference between coefficients obtain
plt.grid()
plt.legend(loc='best')
plt.show()
```

7 Generating Ac and plotting the function values generated against the actual functions

Multiplying the A and c matrices gives us the expected function values, in the range of 0 to 2π. These values are plotted in green circles alongside the actual functions. As cos(cos(x)) is periodic and the coefficients die down quickly, the 51- coefficient approximation yields a closer result to the plot of cos(cos(x)) that is analytically generated. But e^x is a non-periodic function, and hence varies a lot from the actual function's plot. Therefore, we will need a lot more than 51 coefficients to be able to obtain e^x, with more accuracy.

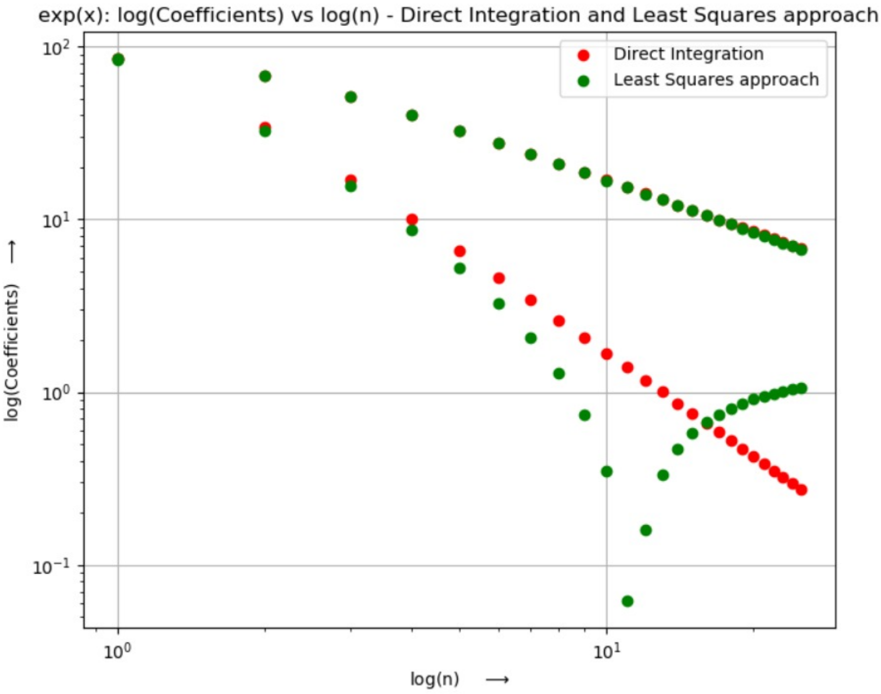

Figure 13: e^x - loglog Plot of Magnitude of coefficients vs n (Both methods)

```
#Plotting Function values generated by least-squares as a function of x
plt.plot(x_vector,Ac,'go',label='Function values from Least Squares appro
plt.plot(x_axis,func(n)[0](x_axis),'r',label=f'{func(n)[1]}')
plt.title(f'{func(n)[1]} alongside Function values obtained through Least
plt.xlabel('x   $\longrightarrow$')
plt.ylabel(f'{func(n)[1]}    $\longrightarrow$')
plt.grid()
plt.legend(loc='best')
plt.show()
#Plotting Function values generated by least-squares approach in log scal
plt.semilogy(x_vector,Ac,'go',label='Function values from Least Squares a
plt.semilogy(x_axis,func(n)[0](x_axis),'r',label=f'{func(n)[1]}')
plt.title(f'Semilogy: {func(n)[1]} alongside Function values obtained thr
plt.xlabel('x   $\longrightarrow$')
plt.ylabel(f'{func(n)[1]} - log scale    $\longrightarrow$')
plt.grid()
plt.legend(loc='best')
plt.show()
```

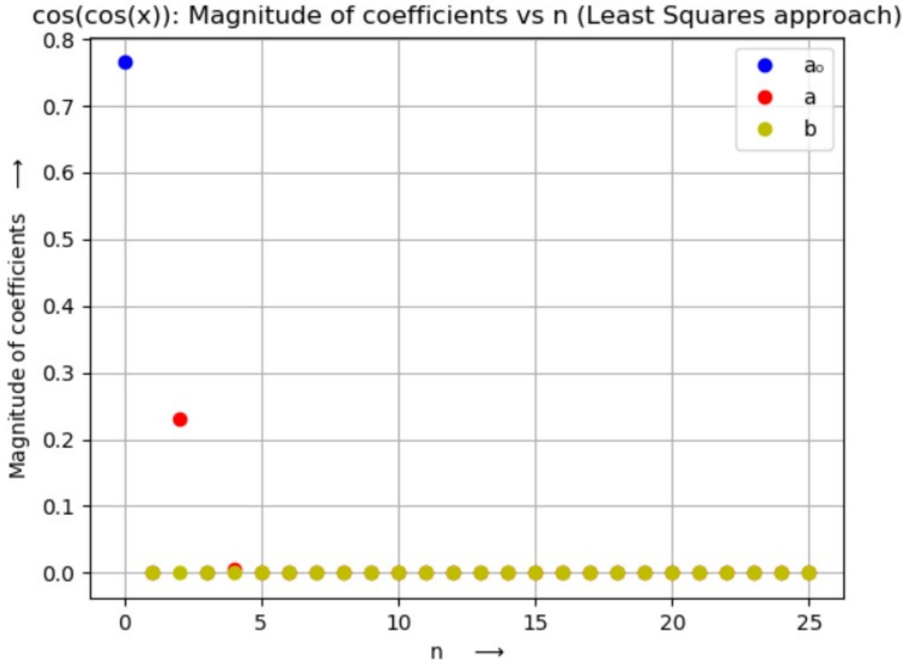

Figure 14: cos(cos(x)) - Magnitude of coefficients vs n (Least Squares method)

The inferences from the plots (Figures 19,20,21 and 22) obtained are:

- Figure 19 shows the generated function values plotted against the actual function e^x. Since the exponential function increases rapidly, the deviation cannot be noted here
- Figure 20 shows the generated function values plotted against the actual function cos(cos(x)). There is little/no deviation seen because cos(cos(x)) is periodic and hence is approximated almost accurately
- Figure 21 gives us a closer look of the deviation found in approximation of e^x, where generated function values and the actual function are plotted with semilog-Y axis
- Figure 22 is the plot of generated function values against the actual function cos(cos(x)) with semilog-Y axis. As expected, there isn't much deviation seen.

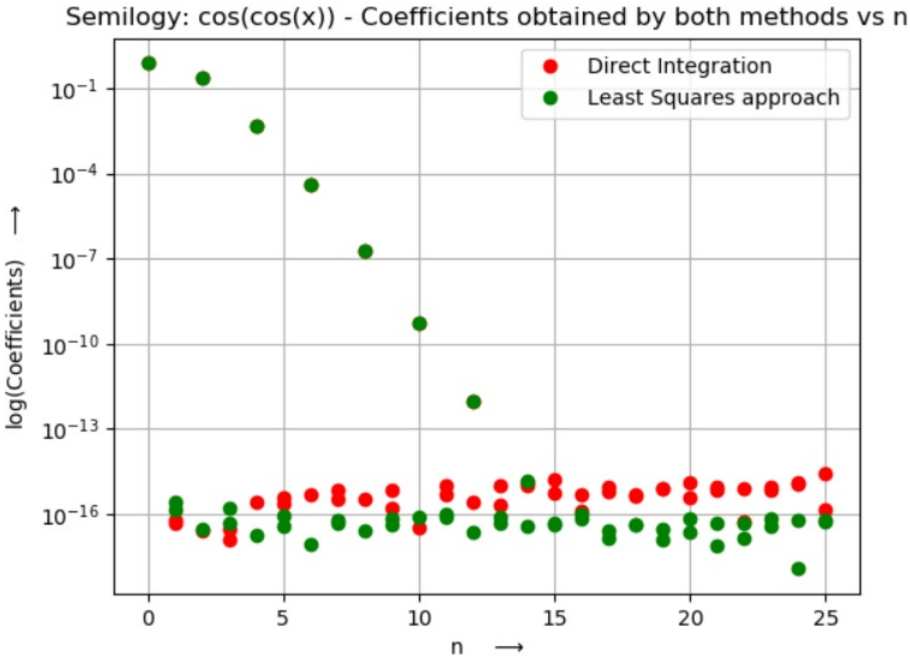

Figure 15: cos(cos(x)) - Magnitude of coefficients vs n (Both methods) with semilog-Y axis

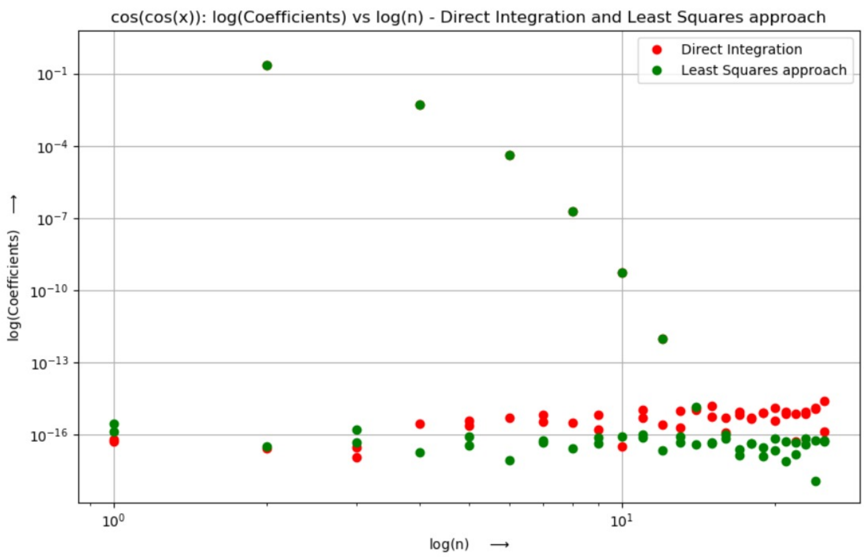

Figure 16: cos(cos(x)) - loglog Plot of Magnitude of coefficients vs n (Both methods)

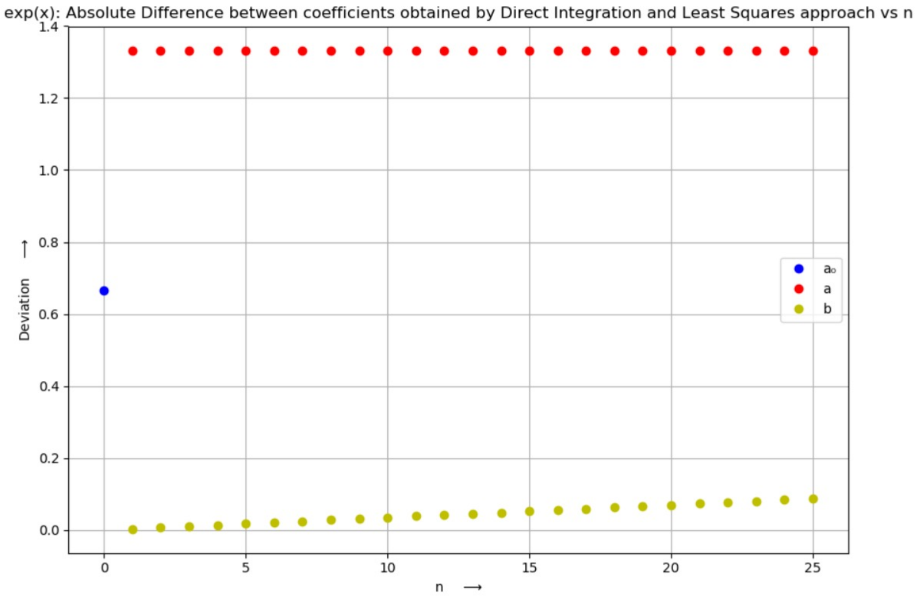

Figure 17: e^x - Absolute difference between coefficients (obtained through both methods) against n

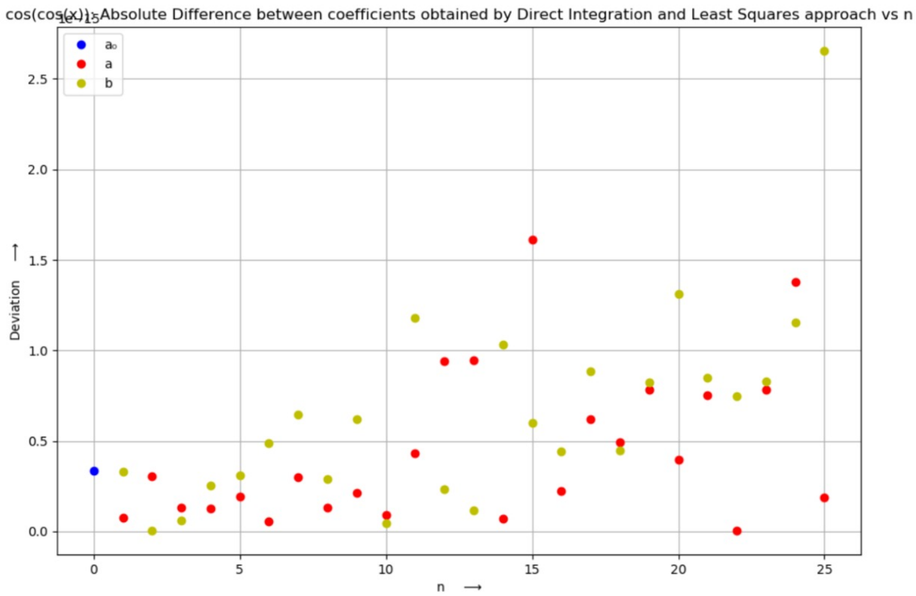

Figure 18: cos(cos(x)) - Absolute difference between coefficients (obtained through both methods) against n

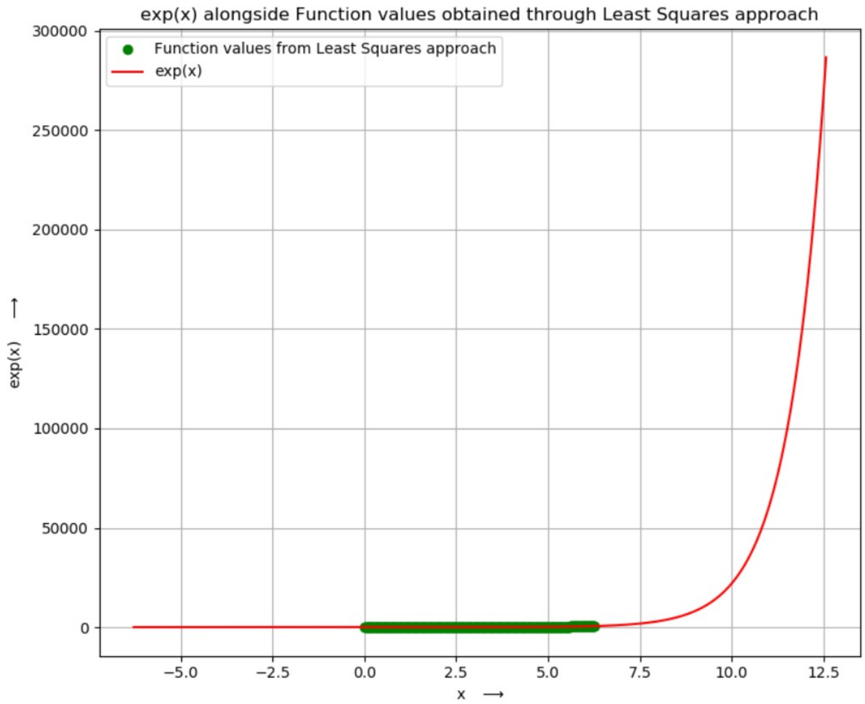

Figure 19: e^x alongside the generated function values from Least Squares approach

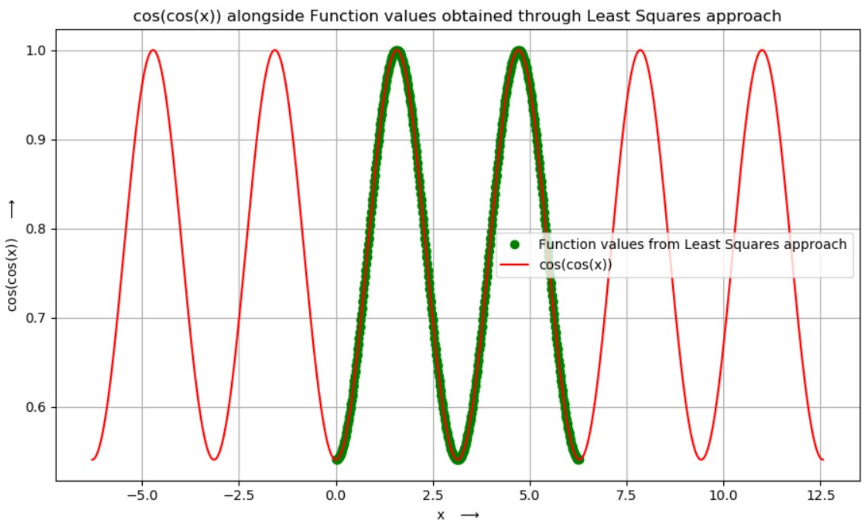

Figure 20: cos(cos(x)) alongside the generated function values from Least Squares approach

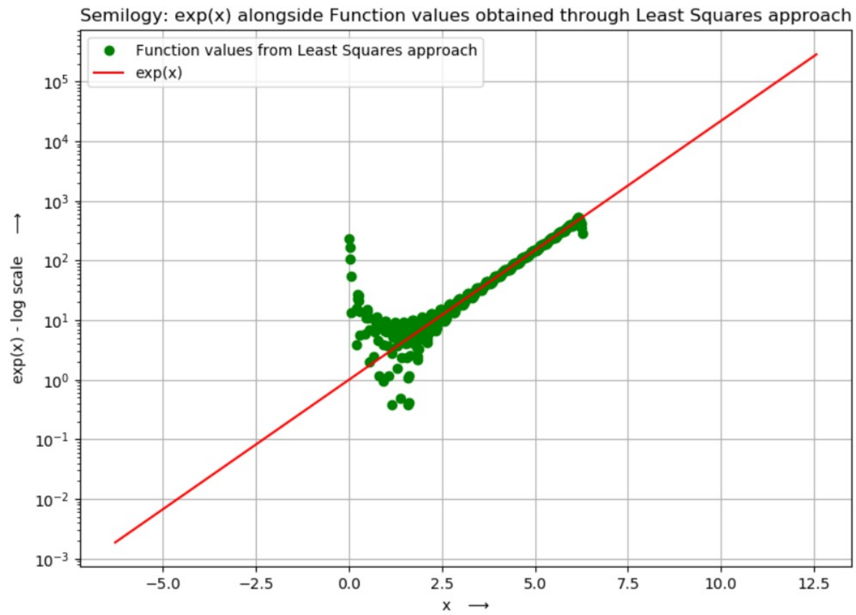

Figure 21: e^x alongside the generated function values from Least Squares approach, with semilog-Y axis

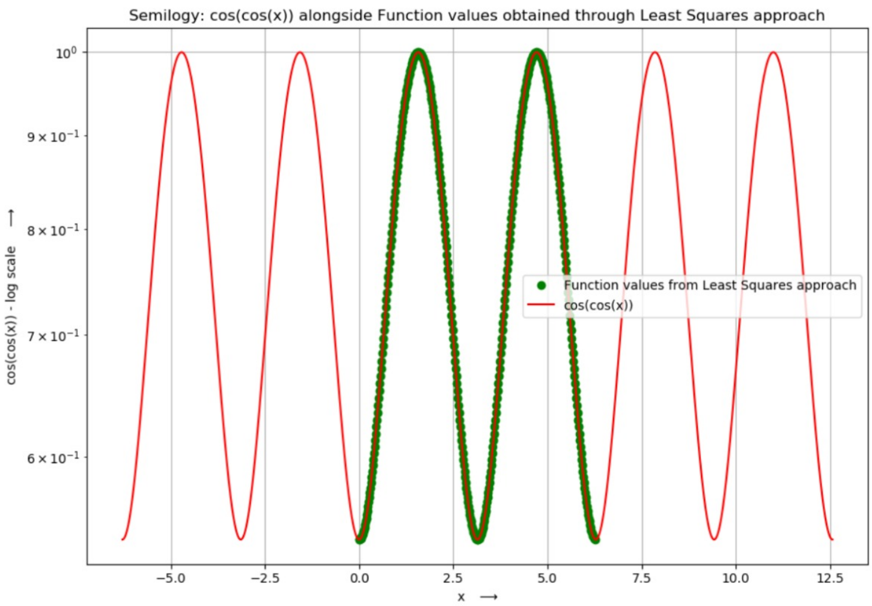

Figure 22: cos(cos(x)) alongside the generated function values from Least Squares approach, with semilog-Y axis