EE2703 : Applied Programming Lab
Assignment 4
Fourier Approximations

Debojyoti Mazumdar
EE20B030

March 10, 2022

0.1 Abstract

The aim of this assignment is to understand how any function can be represented as a sum of periodic trigonometric functions, forming *Fourier coefficients* through *Integration* as well as *Least Square Method* and the checking the accuracy of the newly formed functions.

0.2 Introduction

We are given two functions $cos(cos(x))$, e^x and we have to represent these two functions as fourier series over the interval $(0, 2\pi)$ given by

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

We need to

- Find Fourier Coefficients through Integration and plot them
- Find Fourier Coefficients through Least Square Method and plot them
- Compare the results

0.3 Assignment

0.3.1 Q1

Importing the standard libraries

```
import math
from matplotlib import pyplot as plt
import numpy as np
from scipy.integrate import quad
```

The code here generates two functions for either a scalar/vector input and returns scalar/vector output.

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

def exp_x(x):
    return np.exp(x)
```

Now we plot these both functions over the interval $[-2\pi, 4\pi)$ and see that $cos(cos(x))$ is a periodic function with the period π and e^x isn't periodic.But for this question, we are considering e^x to be periodic with period 2π

```
x = np.linspace(-2*math.pi, 4*math.pi, 1500)
y_exp = exp_x(x)
y_cos_cos = cos_cos_x(x)
plt.title("Plot of exp(x) vs x")
plt.xlabel("x")
plt.ylabel("exp(x)")
plt.plot(x, y_exp, "r")
plt.show()
plt.title("Plot of cos(cos(x)) vs x")
plt.xlabel("x")
plt.ylabel("cos(cos(x))")
plt.plot(x, y_cos_cos, "b")
plt.show()
```

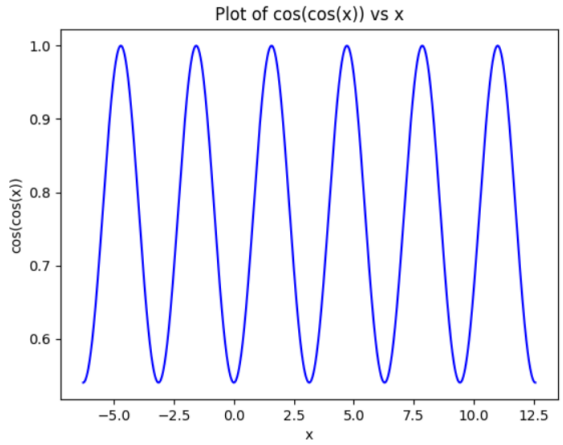

Figure 1: Double Cosine Function

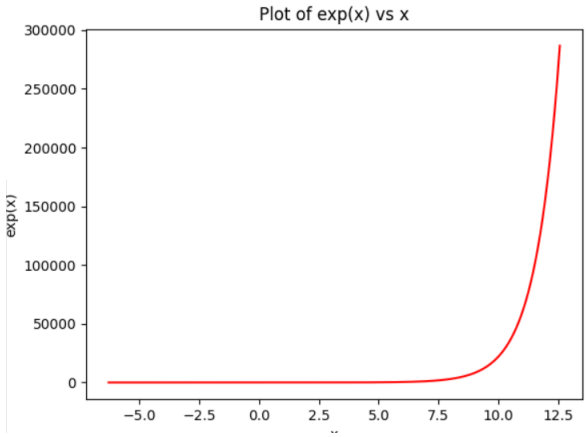

Figure 2: Exponential Function

If we look at the fourier series, we can clearly see that it is only going to represent periodic

functions.So the function here we are considering is actually a periodic function with period 2π and behaves like the actual function in $[0, 2\pi)$

0.3.2 Q2

Now we are going to obtain the coefficients through integration. The formulae for coefficients is

$$\begin{aligned} a_0 &= \frac{1}{2\pi}\int_0^{2\pi} f(x)dx \\ a_n &= \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(nx)\,dx \\ b_n &= \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(nx)\,dx \end{aligned}$$

Implementing the code for the above formulae

```
def return_coeff_of_function(func):
    return_array = []
    if func == "exp(x)":
        for i in range(0, 26):
            a = quad(exp_x_cos_kx, 0, 2*math.pi, args=(i))[0]
            return_array += [a/math.pi]
        for i in range(1, 26):
            b = quad(exp_x_sin_kx, 0, 2*math.pi, args=(i))[0]
            return_array += [b/math.pi]
        return_array[0] = return_array[0]/2
    elif func == "cos(cos(x))":
        for i in range(0, 26):
            a = quad(cos_cos_x_cos_kx, 0, 2*math.pi, args=(i))[0]
            return_array += [a/math.pi]
        for i in range(1, 26):
            b = quad(cos_cos_x_sin_kx, 0, 2*math.pi, args=(i))[0]
            return_array += [b/math.pi]
        return_array[0] = return_array[0]/2
    else:
        return Error("Fourier series for this function is not defined in this python fun
    return np.array(return_array)
```

This code returns the first 25 fourier coefficients of any function that's given as the input in the given format as in part 3. We, in the next part calculate the 25 coefficients for $cos(cos(x))$, e^x.

0.3.3 Q3

First We split the array we got in Q2 into two sub arrays and plot them separately with respect to n.This is to plot two consecutive elements in the array under same n. Here we plot

the coefficients found in a semilog graph and a log-log graph in red circles vs n.

```
fs_coef_exp_x = return_coeff_of_function("exp(x)")
fs_coef_cos_cos_x = return_coeff_of_function("cos(cos(x))")
n = np.arange(0, 26, 1)


# plotting the coefficient values

# exp(x) coefficient plotting in semilogy plot
plt.title("plot of coefficients of cos(cos(x)) in semilogy")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.semilogy(n, np.abs(fs_coef_cos_cos_x[:26]), "bo", label="a")
plt.semilogy(n[1:], np.abs(fs_coef_cos_cos_x[26:]), "ro", label="b")
plt.legend(loc="upper right")
plt.show()

# cos(cos(x)) coefficient plotting in semilogy plot
plt.title("plot of coefficients of exp(x) in semilogy")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.semilogy(n, np.abs(fs_coef_exp_x[:26]), "bo", label="a")
plt.semilogy(n[1:], np.abs(fs_coef_exp_x[26:]), "ro", label="b")
plt.legend(loc="upper right")
plt.show()


# exp(x) coefficients plotting in loglog plot
plt.title("plot of coefficients of cos(cos(x)) in loglog")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.loglog(n, np.abs(fs_coef_cos_cos_x[:26]), "bo", label="a")
plt.loglog(n[1:], np.abs(fs_coef_cos_cos_x[26:]), "ro", label="b")
plt.legend(loc="upper right")
plt.show()

# cos(cos(x)) coefficients plotting in loglog plot
plt.title("plot of coefficients of exp(x) in loglog")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.loglog(n, np.abs(fs_coef_exp_x[:26]), "bo", label="a")
plt.loglog(n[1:], np.abs(fs_coef_exp_x[26:]), "ro", label="b")
plt.legend(loc="upper right")
```

```
plt.show()
```

I used subplots to plot the graph , because it made comparison easier. These are the results I got

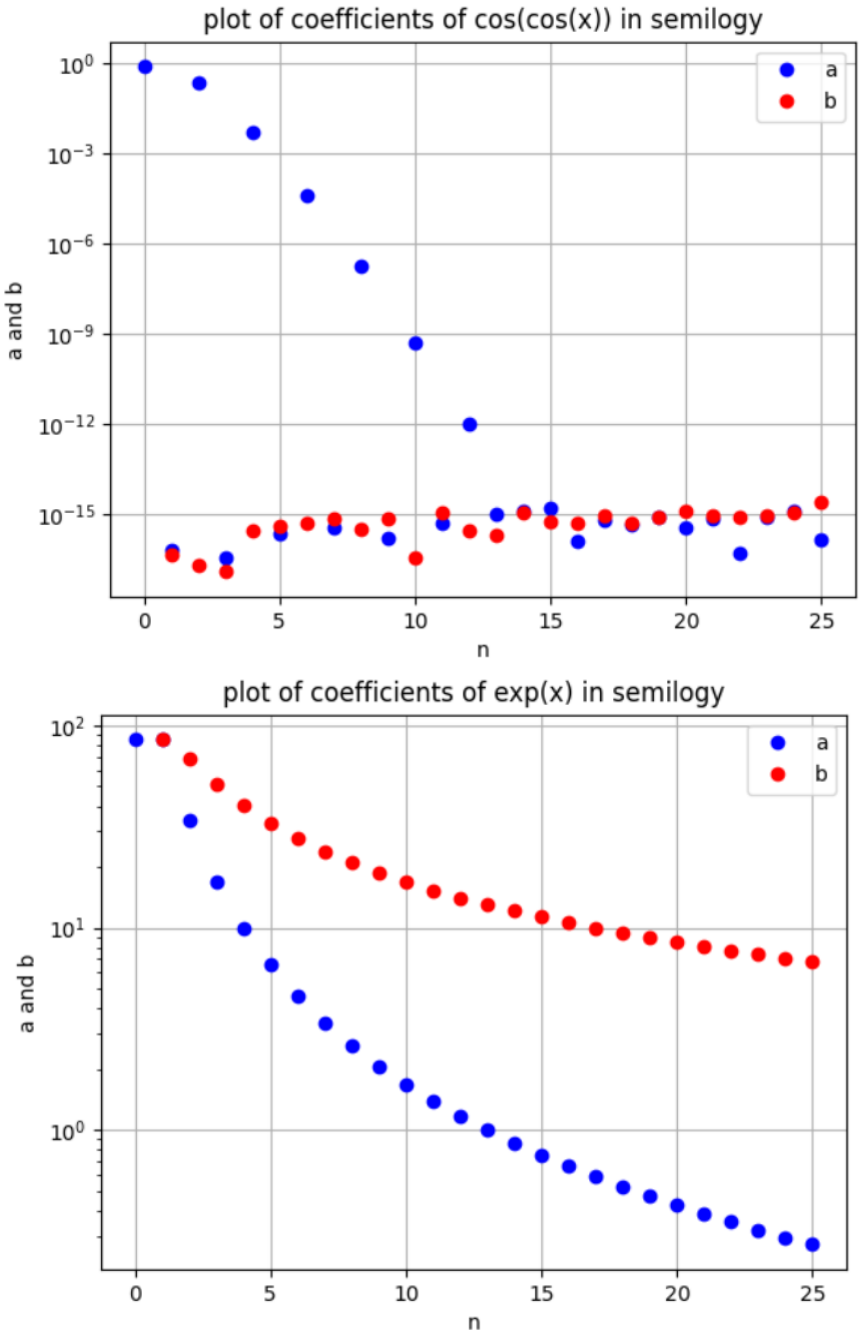

Figure 3: Semilog Plots

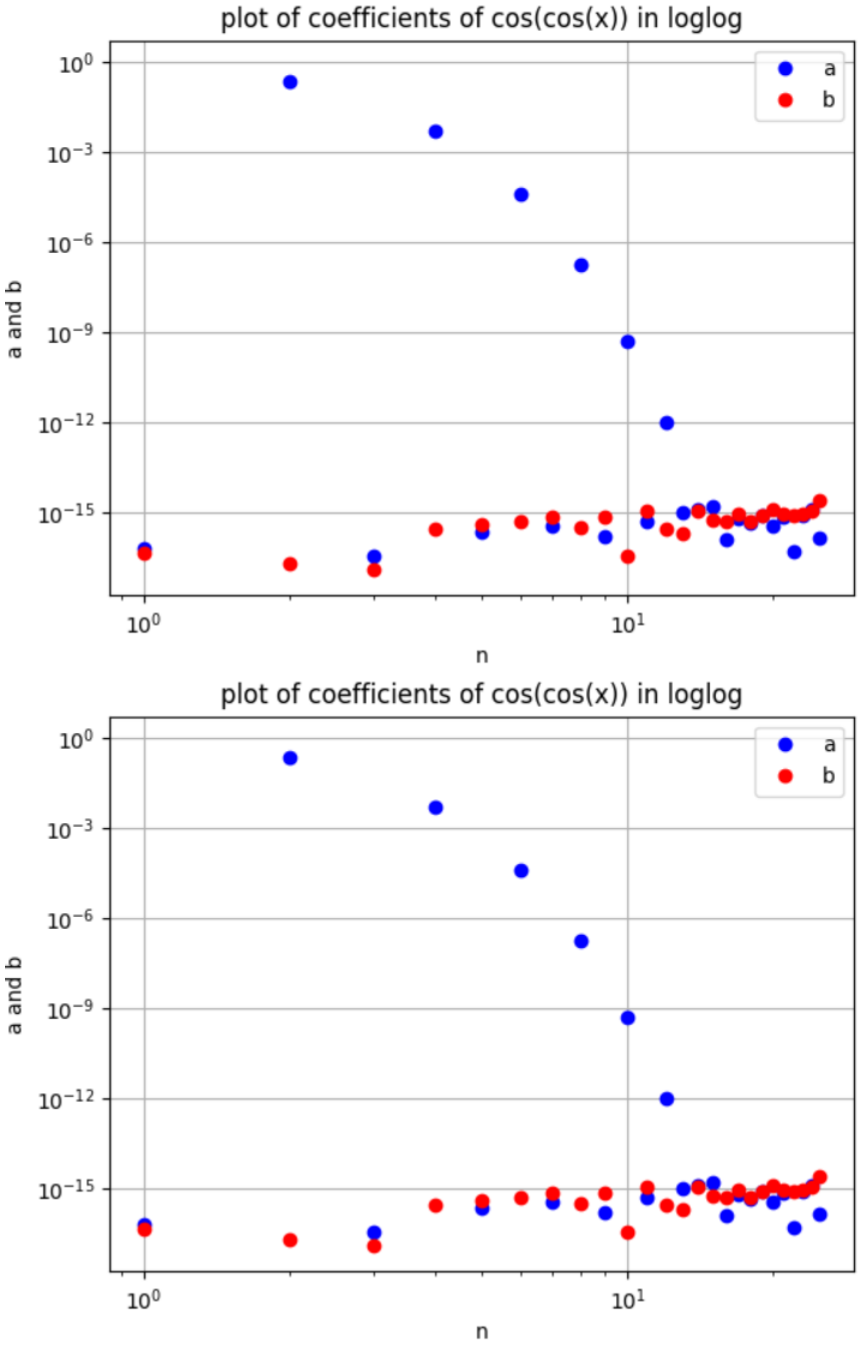

Figure 4: log-log plots

(a) b_n for $cos(cos(x))$ is given by

$$b_n = \int_0^{2\pi} cos(cos(x))sin(nx)\,dx$$

$$b_n = \int_0^{2\pi} cos(cos(2\pi - x))sin(n(2\pi - x))\,dx$$

$$b_n = \int_0^{2\pi} -cos(cos(x))sin(nx)\,dx$$

this implies $b_n = -b_n$ Hence, $b_n = 0$

Hence, the values we found are very close to zero

(b) When we look at the coefficients of second function, The formula is

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

We know that,

$$\int e^{ax} cos(bx) = \frac{e^{ax}}{a^2+b^2}(bsin(bx) + acos(bx)) + c \tag{3}$$

So,

$$\frac{1}{2\pi}\int_0^{2\pi} e^x cos(nx) = \frac{1}{2\pi}\frac{e^x}{1+n^2}(nsin(nx) + cos(nx)) \tag{4}$$

Hence, a_n here is proportional to $\frac{1}{1+n^2}$ and this will be the case for b_n too and the actual function isn't periodic and sinusoidal too. So it's difficult to accurately reach the function with less number of frequencies. So the frequency spectrum is broader for the former.Whereas in the case of double cos, The Fourier transform would yield something like this, clearly saying that after very less number of frequencies the strength of the spectrum is almost zero.Since it's a sinusoidal, less number of frequencies would give us the actual function accurately. This is also reason for one of the further questions, where we get a lot of deviation by using least square method for e^x

(c) Since a_n and b_n for e^x are proportional to $\frac{1}{1+n^2}$, $log(\frac{1}{1+n^2})$ can be approximated to $-2logn$ which is proportional to $logn$ in turn. Hence log-log plot is approximately linear. Now , For the Semilog plot of $cos(cos(x))$, since it's almost exponential, $loga_n$ is proportional to n. Hence, Semilog plot of double cosine function is linear.

0.3.4 Q4 and Q5

Now, we are going to use Least Square Methods to find coefficients. We are creating two matrices here as shown below and then using *lstsq* to find the coefficients. But these coefficients aren't accurate and the function formed from this function will highly deviate from the actual function. So I am forming a vector with the actual function values and a matrix with trigonometric coefficients using a for loop which iterates 25 times and amends the matrix.

$$\begin{pmatrix} 1 & \cos x_1 & \sin x_1 & \dots & \cos 25x_1 & \sin 25x_1 \\ 1 & \cos x_2 & \sin x_2 & \dots & \cos 25x_2 & \sin 25x_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \cos x_{400} & \sin x_{400} & \dots & \cos 25x_{400} & \sin 25x_{400} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ \dots \\ a_{25} \\ b_{25} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \dots \\ f(x_{400}) \end{pmatrix}$$

```
def populate_A():
    x = np.linspace(0, 2*math.pi, 401)
    x = x[:-1]
    A = np.zeros((400, 51))  # allocate space for A
    A[:, 0] = 1  # col 1 is all ones
    for k in range(1, 26):
        A[:, 2*k-1] = np.cos(k*x)  # cos(kx) column
        A[:, 2*k] = np.sin(k*x)  # sin(kx) column
    return A

def converting_to_original_form(c):
    return_list = [c[0]]
    for i in range(1, len(c)):
        if i % 2 != 0:
            return_list += [c[i]]
    for i in range(1, len(c)):
        if i % 2 == 0:
            return_list += [c[i]]
    return np.array(return_list)

x = np.linspace(0, 2*math.pi, 401)
x = x[:-1]  # drop last term to have a proper periodic integral
b_exp_x = np.exp(x)  # exp(x) has been written to take a vector
# cos(cos(x)) has been written to take a vector
b_cos_cos_x = np.cos(np.cos(x))
A = populate_A()
c_exp_x = converting_to_original_form(np.linalg.lstsq(A, b_exp_x)[0])
c_cos_cos_x = converting_to_original_form(np.linalg.lstsq(A, b_cos_cos_x)[0])
```

After this, we have to plot the coefficients and compare them with the previous coefficients. To differentiate the new points, They are plotted in green circles.

```
# exp(x) both coefficient plotting in semilogy plot
plt.title("plot of coefficients of cos(cos(x)) in semilogy")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.semilogy(n, np.abs(fs_coef_cos_cos_x[:26]), "ro", label="a")
plt.semilogy(n[1:], np.abs(fs_coef_cos_cos_x[26:]), "ro", label="b")
plt.semilogy(n, np.abs(c_cos_cos_x[:26]), "go", label="a_new")
plt.semilogy(n[1:], np.abs(c_cos_cos_x[26:]), "go", label="b_new")
plt.legend(loc="upper right")
plt.show()

# cos(cos(x)) both coefficient plotting in semilogy plot
plt.title("plot of coefficients of exp(x) in semilogy")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
```

```
plt.semilogy(n, np.abs(fs_coef_exp_x[:26]), "ro", label="a")
plt.semilogy(n[1:], np.abs(fs_coef_exp_x[26:]), "ro", label="b")
plt.semilogy(n, np.abs(c_exp_x[:26]), "go", label="a_new")
plt.semilogy(n[1:], np.abs(c_exp_x[26:]), "go", label="b_new")
plt.legend(loc="upper right")
plt.show()


# exp(x) coefficients plotting in loglog plot
plt.title("plot of coefficients of cos(cos(x)) in loglog")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.loglog(n, np.abs(fs_coef_cos_cos_x[:26]), "ro", label="a")
plt.loglog(n[1:], np.abs(fs_coef_cos_cos_x[26:]), "ro", label="b")
plt.loglog(n, np.abs(c_cos_cos_x[:26]), "go", label="a_new")
plt.loglog(n[1:], np.abs(c_cos_cos_x[26:]), "go", label="b_new")
plt.legend(loc="upper right")
plt.show()

# cos(cos(x)) coefficients plotting in loglog plot
plt.title("plot of coefficients of exp(x) in loglog")
plt.xlabel("n")
plt.ylabel("a and b")
plt.grid()
plt.loglog(n, np.abs(fs_coef_exp_x[:26]), "ro", label="a")
plt.loglog(n[1:], np.abs(fs_coef_exp_x[26:]), "ro", label="b")
plt.loglog(n, np.abs(c_exp_x[:26]), "go", label="a_new")
plt.loglog(n[1:], np.abs(c_exp_x[26:]), "go", label="b_new")
plt.legend(loc="upper right")
plt.show()
```

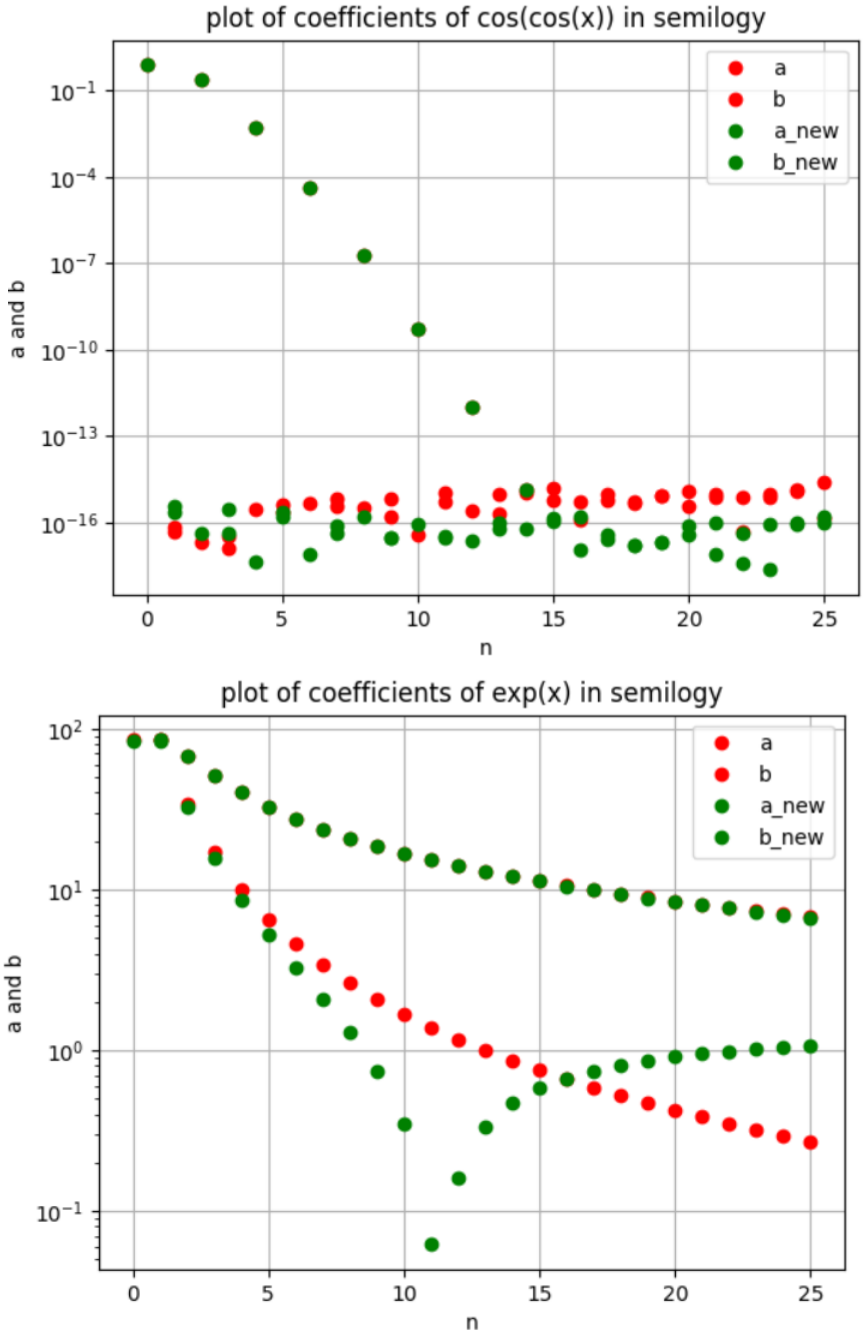

Figure 5: Coefficients with lstsq vs Coefficients with Integration

0.3.5 Q6

No, The coefficients formed with both the methods are not same. We can see the difference here. This is because the least squares method assumes no other frequency above n = 25 contributes to the function, but that's not the case here.

```
# To find the maximum absolute difference between the two sets of coefficients
print("The maximum error in a of exp(x) is : ",
      np.max(np.abs(c_exp_x[:26]-fs_coef_exp_x[:26])))
print("The maximum error in b of exp(x) is : ",
      np.max(np.abs(c_exp_x[26:]-fs_coef_exp_x[26:])))
```

```
print("The maximum error in a of cos(cos(x)) is : ", np.max(
    np.abs(c_cos_cos_x[:26]-fs_coef_cos_cos_x[:26])))
print("The maximum error in b of cos(cos(x)) is : ", np.max(
    np.abs(c_cos_cos_x[26:]-fs_coef_cos_cos_x[26:])))
```

And the result we get after running this code is:

```
The maximum error in a of exp(x) is :  1.3327308703353395
The maximum error in b of exp(x) is :  0.08768050912536829
The maximum error in a of cos(cos(x)) is :  1.7175489006562513e-15
The maximum error in b of cos(cos(x)) is :  2.7586698812539523e-15
```

0.3.6 Q7

Now we calculate the function values with the new coefficients and plot it in green and the old function which is the actual function in red. We can see that $cos(cos(x))$ is perfectly matching whereas e^x isn't.

```
def converting_to_required_format(fs_coef):
    return_list = np.zeros(51)
    return_list[0] = fs_coef[0]
    for i in range(1, 26):
        return_list[2*i-1] = fs_coef[i]
    for i in range(26, len(fs_coef)):
        return_list[2*(i-25)] = fs_coef[i]
    return return_list


x = np.linspace(0, 2*math.pi, 401)
x = x[:-1]
y_cos_cos_x = np.cos(np.cos(x))
y_exp_x = np.exp(x)
A = populate_A()
c_cos_cos_x = converting_to_required_format(fs_coef_cos_cos_x)
c_exp_x = converting_to_required_format(fs_coef_exp_x)
y_cos_cos_x_from_fs = np.matmul(A, c_cos_cos_x)
y_exp_x_from_fs = np.matmul(A, c_exp_x)


plt.title("Plot of exp(x)")
plt.grid()
plt.semilogy(x, y_exp_x_from_fs, "ro", label="exp(x) from fourier series")
plt.semilogy(x, y_exp_x, "blue", label="exp(x) function")
plt.show()

plt.title("Plot of cos(cos(x))")
plt.grid()
```

```
plt.plot(x, y_cos_cos_x_from_fs, "ro", label="cos(cos(x)) from fourier series")
plt.plot(x, y_cos_cos_x, "blue", label="cos(cos(x)) function")
plt.show()
```

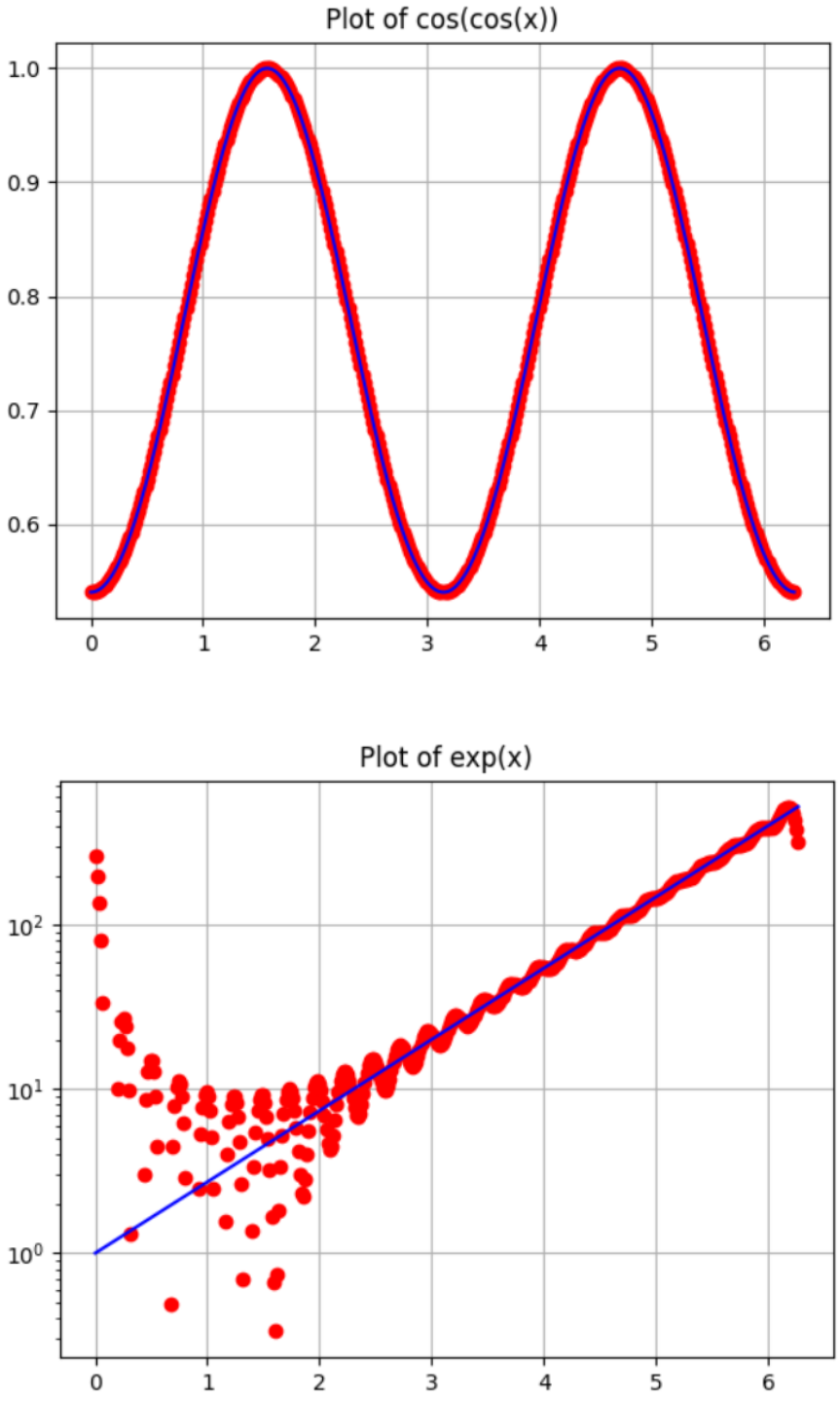

Figure 6: Zoomed-in Graph

0.4 Conclusions

- I wrote code in such a way that it finds the first 25 pairs of coefficients of any function if you pass it as argument. I learnt a couple of things about from this code like
  - A Fourier Series is used to represent any function in a given domain in terms of sum of periodic trigonometric functions of different frequencies.
  - Fourier Coefficients are decreasing for the function with n as the peak graph in frequency domain goes down
  - Fourier Coefficients just give the approximate trigonometric estimation of the function, not the accurate unless we take the sum till infinity
- Since We tried to approximate $e^x; [0, 2\pi)$ here, we took it as periodic with period 2π. Since it's non-periodic, there are a lot of non-differentiable points. At these points Fourier approximation(till 25 coefficients) is deviating a lot from actual function.To Summarise, Fourier Series deviates a lot for non-periodic function more than periodic function.
- Least Square Method for finding coefficients is faster than integration but is less accurate than latter.
- Any function with actual period 2π will have $b_n = 0$
- **Gibbs Phenomenon** is the phenomenon due to which the fourier approximations deviate more for non-periodic functions

I have learnt how to estimate any function's fourier coefficients using Integration and Least Square Method and checking the accuracy of the function via plotting various graphs.