# EE2703: Applied Programming Lab
# Assignment 4
# Fourier Approximations

Kaushik Ravibaskar
EE20B057

February 26, 2022

## Contents

# 1 Aim

The assignment mainly deals with the following points:

- To fit the functions $e^x$ and $\cos(\cos(x))$ using **fourier series** approximation.
- To find the fourier coefficients using **least square** analysis and comparing them with the coefficients obtained by integration.
- To plot the results obtained at each step in a meaningful manner and take inferences from their nature.

# 2 Introduction

The **Fourier Series** is a very powerful tool to express any periodic function as a summation of sinusoids. The **Fourier Series** is given by:

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

where the coefficients for a $2\pi$ periodic function are given as follows:

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

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

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

In this assignment, we are taking the above mentioned functions over the interval of $[0, 2\pi)$. As we can see, $\cos(\cos(x))$ is already periodic over the entire domain of real numbers, but for $e^x$, we had to make it periodic by repeating its nature in $[0, 2\pi)$ over the entire domain.

# 3 Plotting the functions

Firstly, we are plotting the actual function between $[-2\pi, 4\pi)$ using the **pylab** module. Since, $e^x$ isn't a periodic function, the expected fourier plot for it has been made periodic by clipping the $[0, 2\pi)$ nature of it and extending it. The code is as follows:

```
#for exponential function
x1_arr = py.linspace(-2*ma.pi, 4*ma.pi, 301)
```

```
a_arr = py.linspace(-2*ma.pi, 0, 101)
b_arr = py.linspace(0, 2*ma.pi, 101)
c_arr = py.linspace(2*ma.pi, 4*ma.pi, 101)
py.semilogy(x1_arr[:-1], f1(x1_arr[:-1]), label = 'true plot')
py.semilogy(a_arr[:-1], f1(a_arr[:-1]+2*ma.pi), 'y--', label = 'expected plot')
py.semilogy(b_arr[:-1], f1(b_arr[:-1]), 'y--')
py.semilogy(c_arr[:-1], f1(c_arr[:-1]-2*ma.pi), 'y--')
py.xlabel('x' + r'$\rightarrow$')
py.ylabel('log(exp(x))' + r'$\rightarrow$')
py.title('Figure 1: exp(x) on a semilog-y plot')
py.savefig('Q1(a).png', dpi = 1000)
py.grid(True)
py.legend(loc = 'upper right')
py.show()

#for cosine_like function
x2_arr = py.linspace(-2*ma.pi, 4*ma.pi, 301)
py.plot(x2_arr[:-1], f2(x2_arr[:-1]), label = 'true plot')
py.plot(x2_arr[:-1], f2(x2_arr[:-1]), 'y--', label = 'expected plot')
py.xlabel('x' + r'$\rightarrow$')
py.ylabel('cos(cos(x))' + r'$\rightarrow$')
py.title('Figure 2: Plot of cos(cos(x))')
py.savefig('Q1(b).png', dpi = 1000)
py.grid(True)
py.legend(loc = 'upper right')
py.show()
```

The plot for the two functions is as follows:

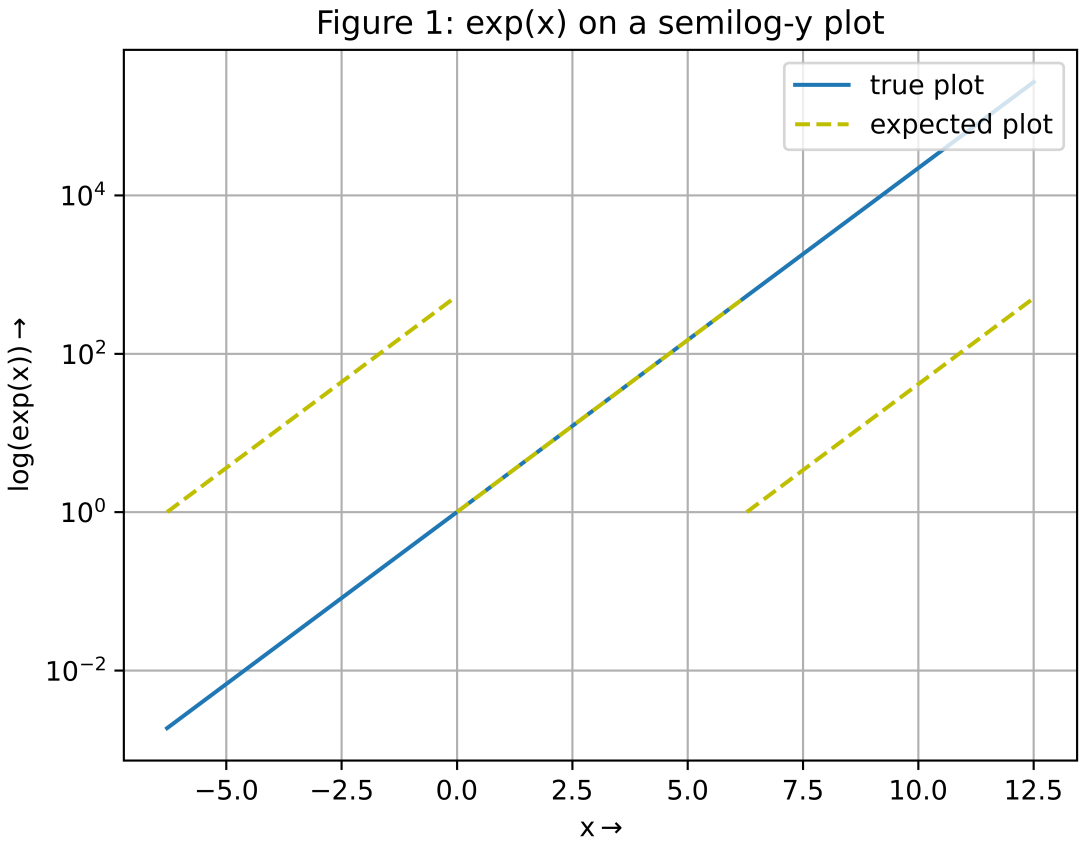


Figure 1: exp(x) on a semilog-y plot
true plot
expected plot
log(exp(x)) →
$10^4$
$10^2$
$10^0$
$10^{-2}$
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
x →

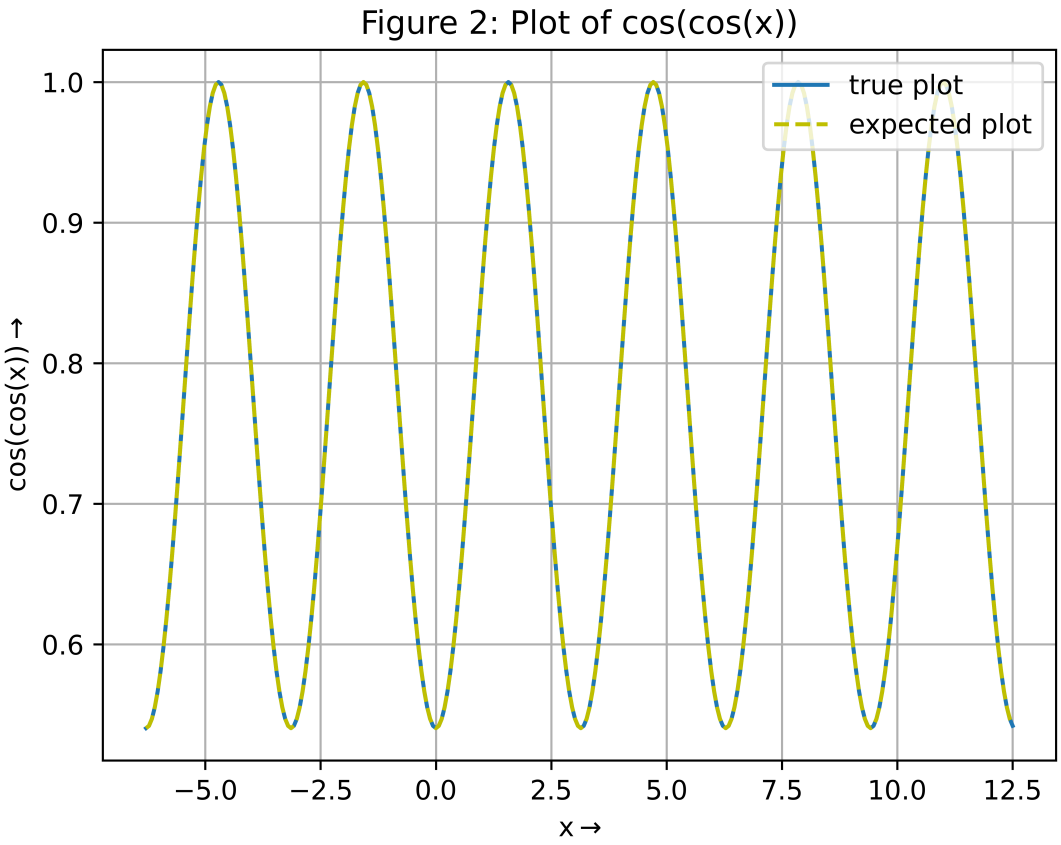


Figure 2: Plot of cos(cos(x))
true plot
expected plot
1.0
0.9
0.8
0.7
0.6
cos(cos(x)) →
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
x →

# 4 Finding Fourier coefficients for the given functions

We can obtain the first 51 coefficients of the fourier series of the given functions using the formula mentioned in the introduction above. Python provides a built-in function to obtain integration results. We will obtain the first $26 a_n(s)$, and first $25 b_n(s) s$, using a **for loop** as follows:

```
#a_0 coefficient
a1_coeff[0] = quad(u, 0, 2*ma.pi, args=(0, 'f1'))[0]/(2*ma.pi)

#a_coefficients
for index in range(1, 26):
    a1_coeff[index] = quad(u, 0, 2*ma.pi, args=(index, 'f1'))[0]/(ma.pi)
#b_coefficients
for index in range(1, 26):
    b1_coeff[index-1] = quad(v, 0, 2*ma.pi, args=(index, 'f1'))[0]/(ma.pi)
```

In here, we have defined two new functions, namely $u(x,k) = f(x)\cos kx$ for $a_n$ coefficients and $v(x,k) = f(x)\sin kx$ for $b_n$ coefficients, so that they can be used to find the coefficients in an elegant manner.

```
#defining a_coeff integrand
def u(x,k,selector):

    if(selector == 'f1'):
        return f1(x)*np.cos(k*x)
    else:
        return f2(x)*np.cos(k*x)

#defining b_coeff integrand
def v(x,k,selector):

    if(selector == 'f1'):
        return f1(x)*np.sin(k*x)
    else:
        return f2(x)*np.sin(k*x)
```

# 5 Plotting the Fourier coefficients for the given functions

We will now plot the fourier coefficients obtained in the previous sections for the two functions. For these plots, n is over the interval $[0, 50]$ and the coefficients are taken as follows:

$$Coeff = \begin{bmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_{25} \\ b_{25} \end{bmatrix} \tag{1}$$

The plots are as follows:

Figure 3: Coefficients of fourier series of exp(x) in a semilog plot

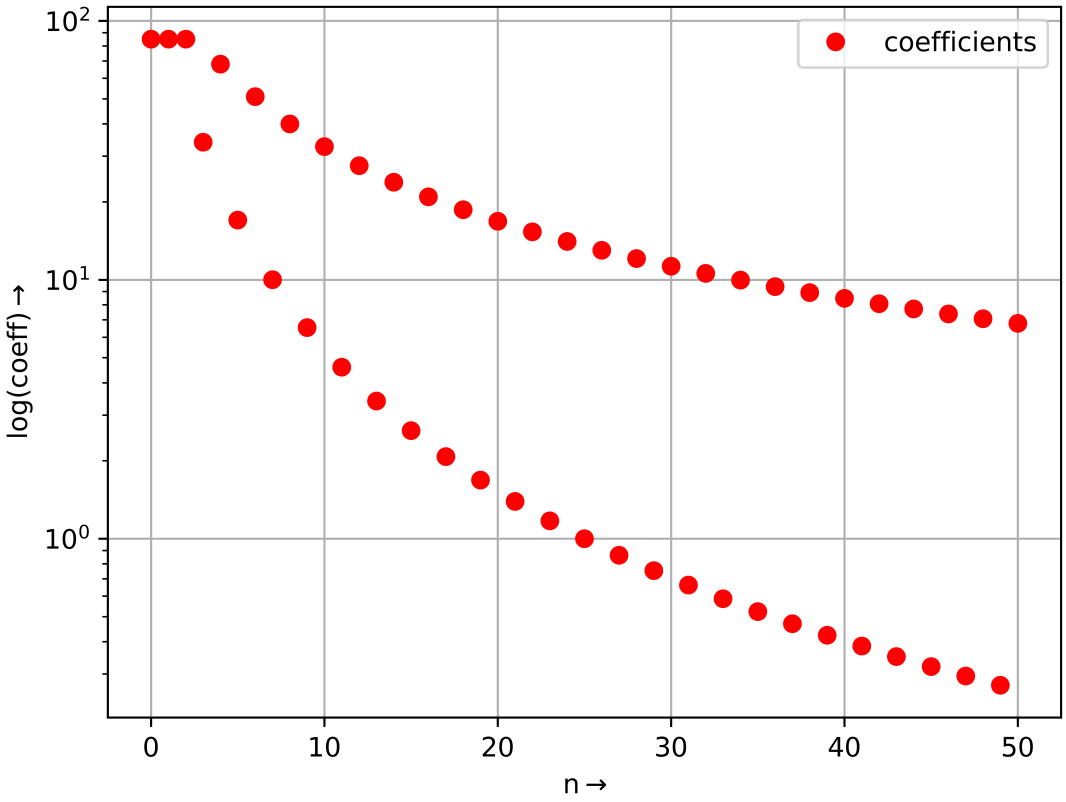


Figure 4: Coefficients of fourier series of exp(x) in loglog plot

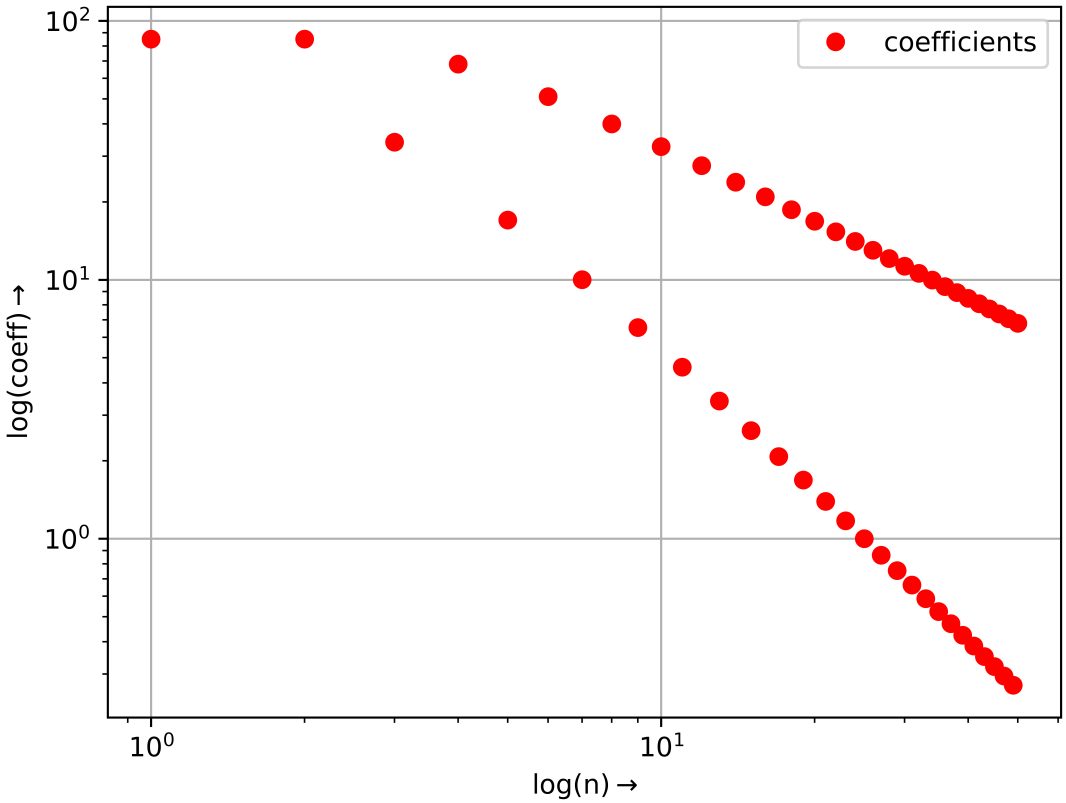


Figure 5: Coefficients of fourier series of cos(cos(x)) in semilog plot

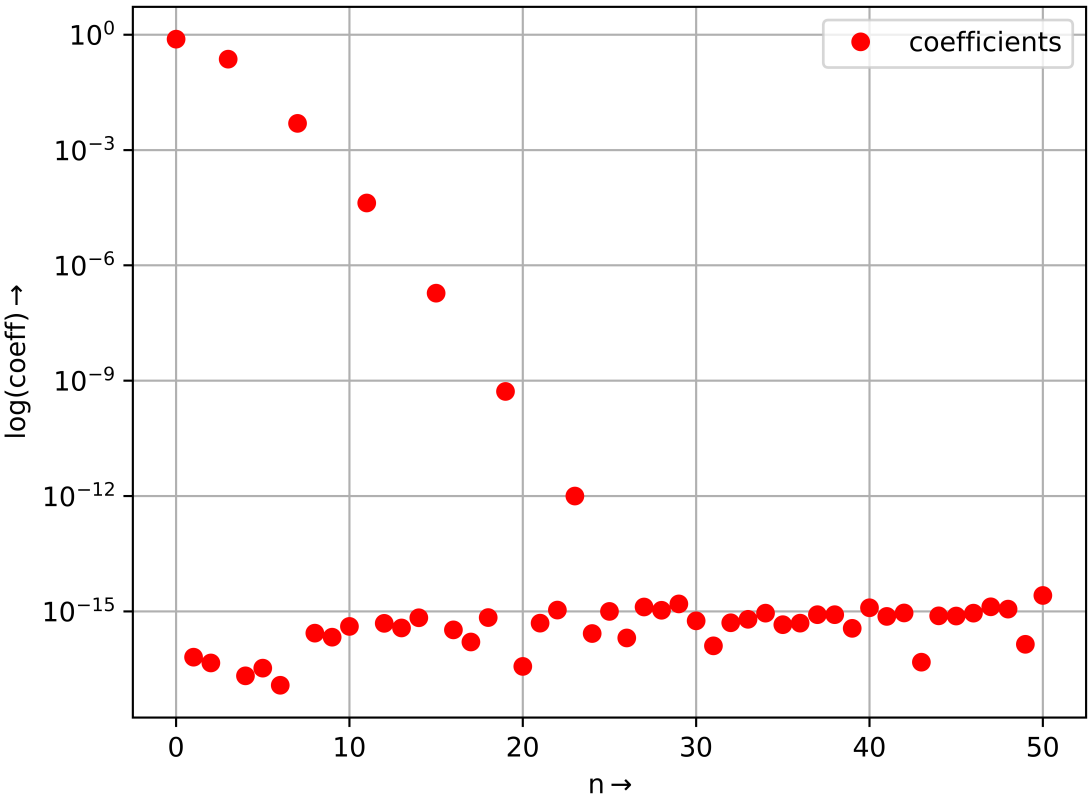
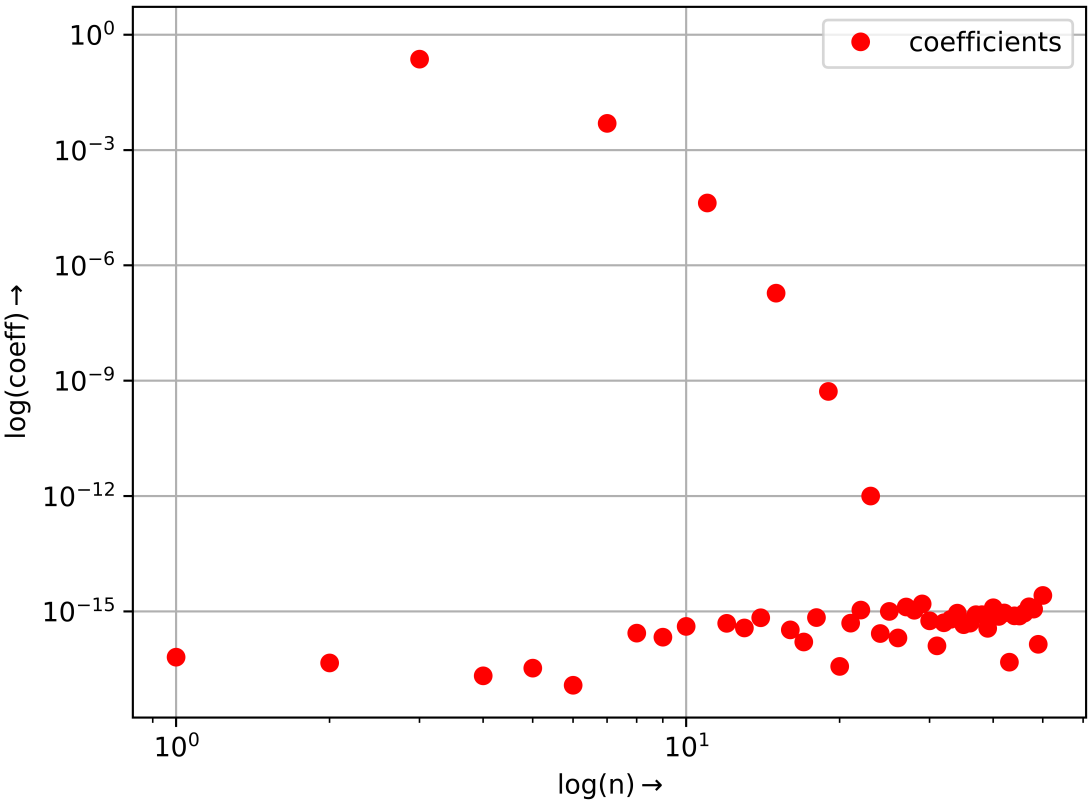


Figure 6: Coefficients of fourier series of cos(cos(x)) in loglog plot



Some observations are as follows:

- If we do integration by parts for $b_n$ for $\cos(\cos(x))$ and apply $King's Rule$ in the integration process, the *sine* term in the integral changes its sign and hence makes the integral zero.

$$b_n = 0$$

  Hence, the $b_n$ values are nearly zero in the plot.

- It can be mathematically proved that, the coefficients of $e^x$ are proportional to the polynomials of n (in a decaying manner), whereas for $\cos(\cos(x))$, they are proportional to exponential of n (in a decaying manner). As exponential decays faster, so does the coefficients for the second case.

- As stated above, for $e^x$, loglog plot would give a linear nature, whereas for $\cos(\cos(x))$ semilog plot must be linear. This satisfies with the achieved plot to some extent.

# 6 Using 'Least Square' approach to find Fourier coefficients

Now we will use **least square** approach to find fourier coefficients for the given functions. First, we will initialize a vector $x$, which will take 400 values between $[0, 2\pi)$ and generate a $b$ vector by $b = f(x)$. Then, we will contruct a matrix $M$ as follows:

$$M = \begin{bmatrix} 1 & \text{cosx}_1 & \text{sinx}_1 & ... & \text{cos25x}_1 & \text{sin25x}_1 \\ 1 & \text{cosx}_2 & \text{sinx}_2 & ... & \text{cos25x}_2 & \text{sin25x}_2 \\ ... & ... & ... & ... & ... & ... \\ 1 & \text{cosx}_{400} & \text{sinx}_{400} & ... & \text{cos25x}_{400} & \text{sin25x}_{400} \end{bmatrix} \tag{2}$$

We will also define a vector $p$ which corresponds to the fourier coefficients which we have to compute using the **least square** method.

$$p = \begin{bmatrix} \text{a}_0 \\ \text{a}_1 \\ \text{b}_1 \\ ... \\ \text{a}_{25} \\ \text{b}_{25} \end{bmatrix} \tag{3}$$

Now we are interested in the relation:

$$M * p = b$$

Hence, we will be using **least square** approach to find the best fit for p, that is fourier coefficients for the given functions. The code to do the process is as follows:

```
#constructing M matrix
M = py.zeros((400, 51))
M[:,0] = 1
for index in range(1,26):
    M[:,(2*index-1)] = py.cos(index*x_arr)
    M[:,2*index] = py.sin(index*x_arr)

#constructing RHS_array for the two functions
rhs1 = f1(x_arr)   #exp function
rhs2 = f2(x_arr)   #cosine-like function

#using 'Least Square' method to compute the 'best fit' coefficients
ans_lsq_arr_1 = sp.linalg.lstsq(M,rhs1)[0]
ans_lsq_arr_2 = sp.linalg.lstsq(M,rhs2)[0]
```

# 7 Plotting the True and Approximated Fourier coefficients

The plots comparing the fourier coefficients for the given functions obtained using the two methods are as follows:

Figure 3: Coefficients of fourier series of exp(x) in semilog plot

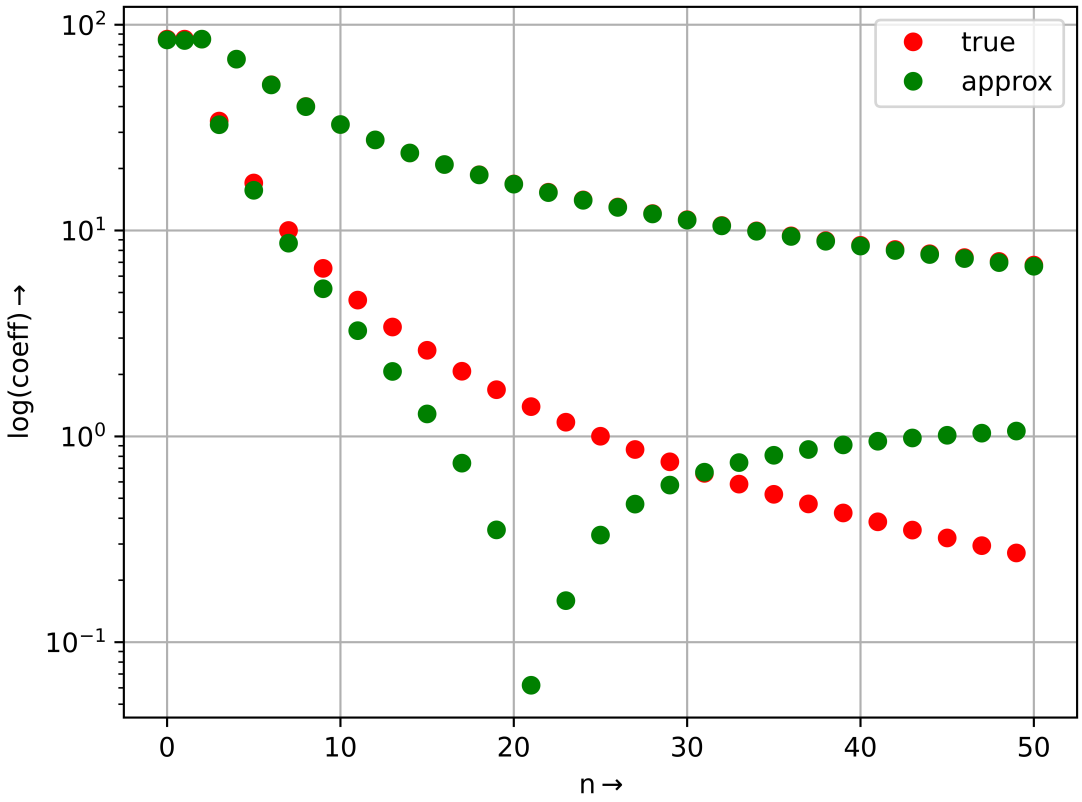


Figure 4: Coefficients of fourier series of exp(x) in loglog plot

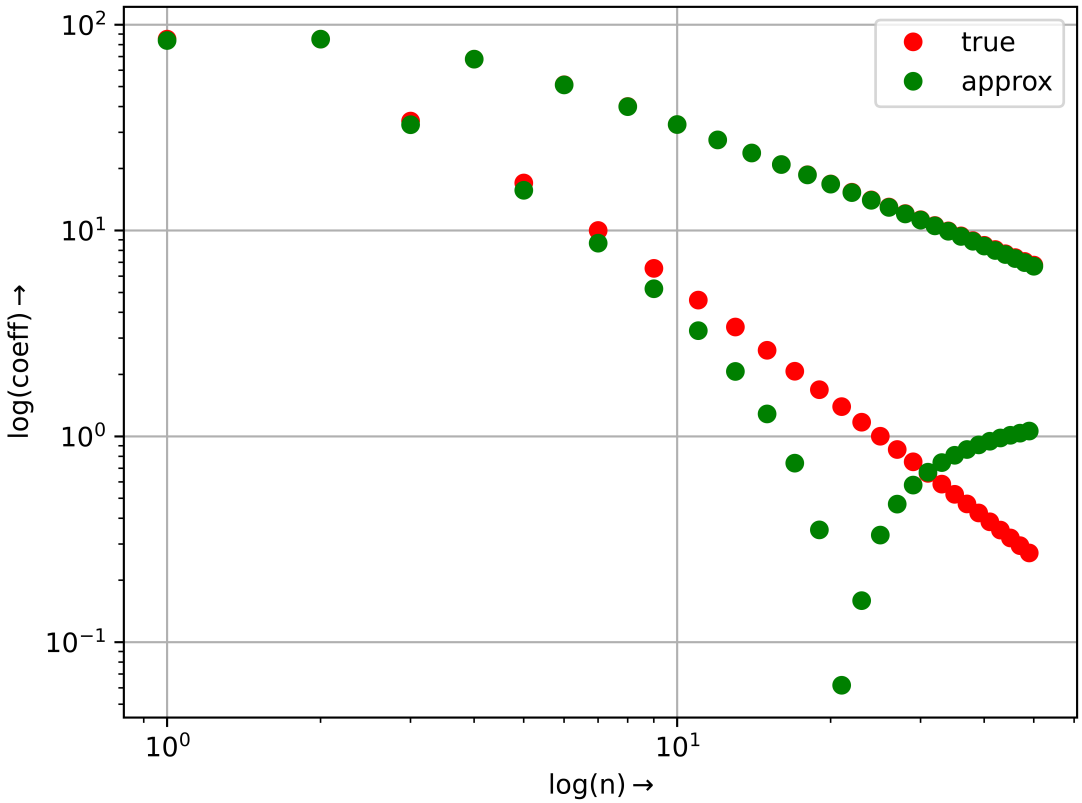


Figure 5: Coefficients of fourier series of cos(cos(x)) in semilog plot

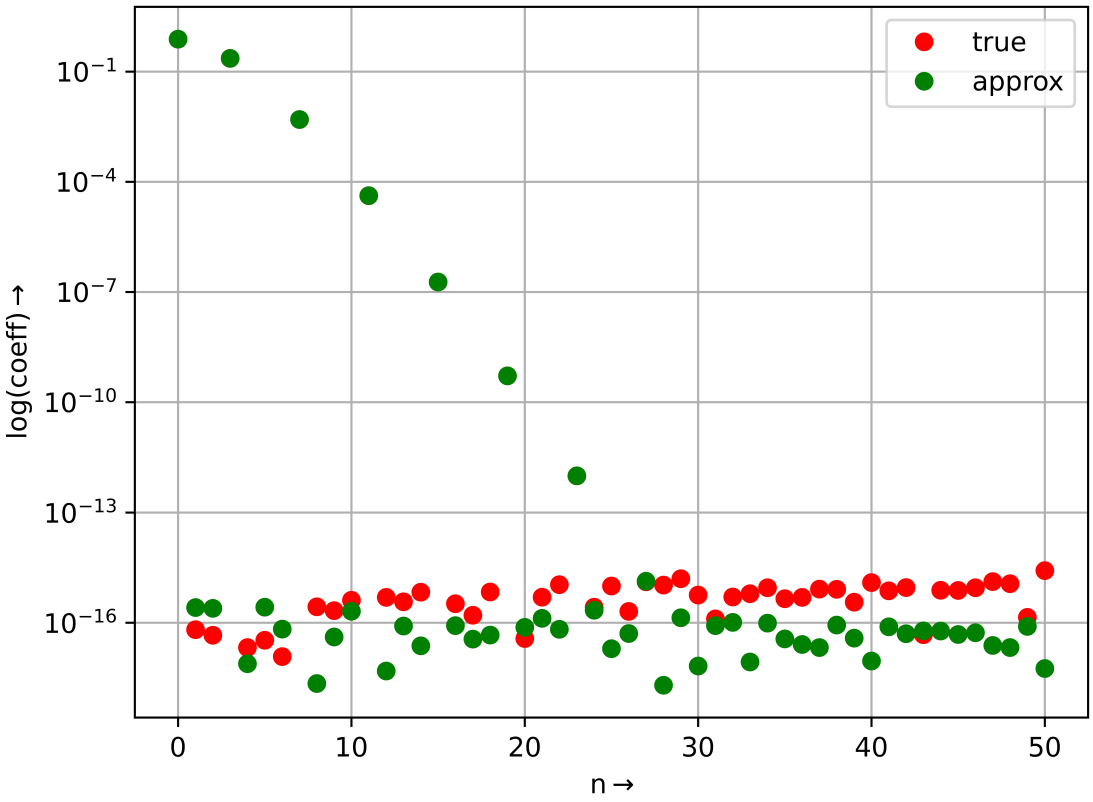
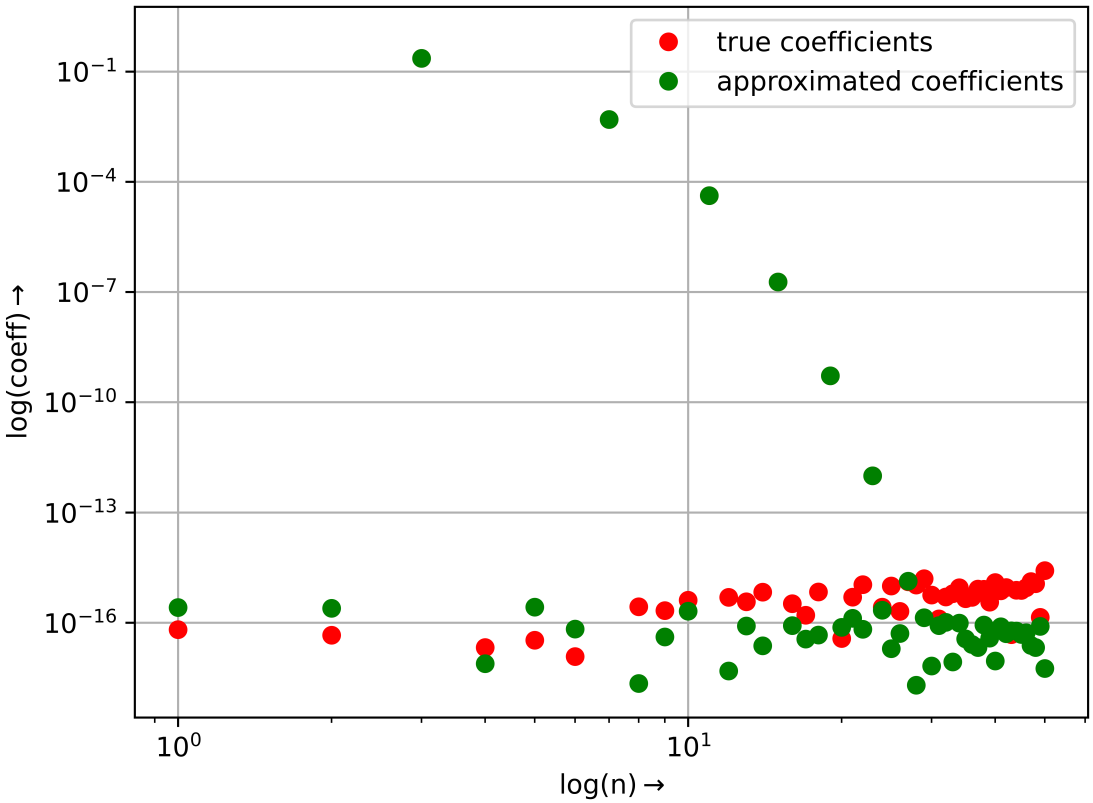


Figure 6: Coefficients of fourier series of cos(cos(x)) in loglog plot



# 8 Comparing the Integral and Least Square coefficients

A comparison from the python program between the **true** and **least square** coefficients is as follows:

```
The maximum absolute deviation for f1(x) coefficients
is 1.3327308703354248
The maximum absolute deviation for f2(x) coefficients
is 2.5996231400227228e-15
Hence, the two coefficients are not same.
```

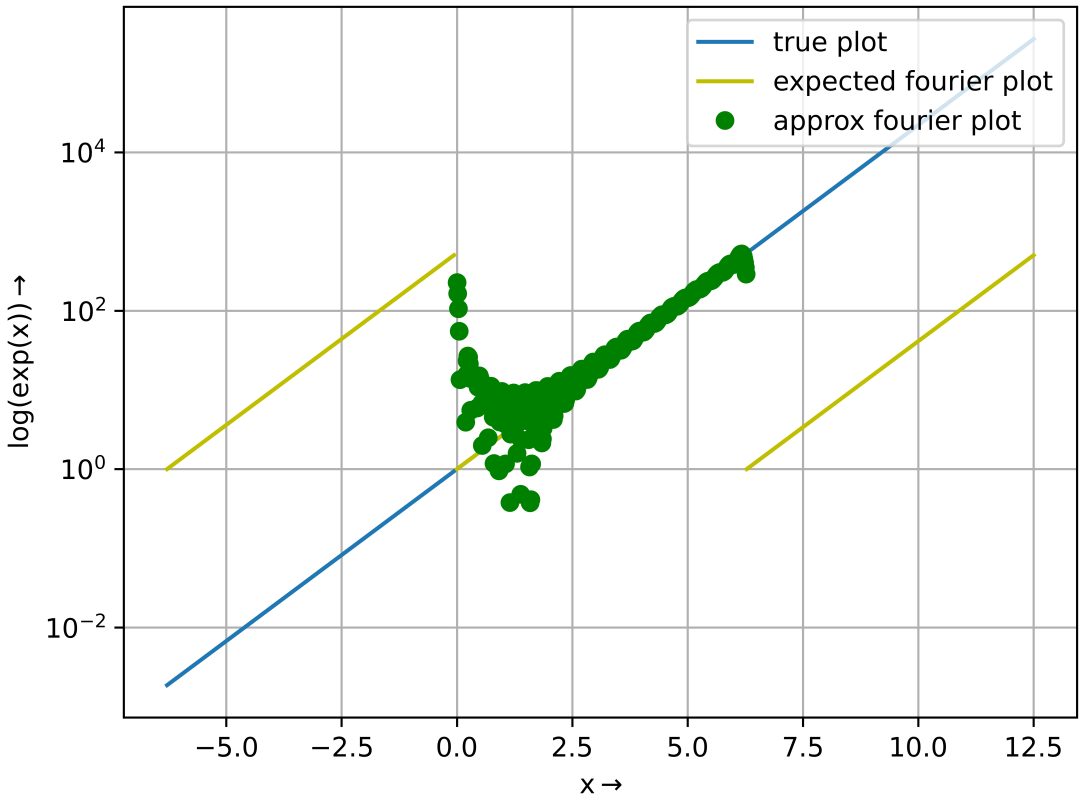
Thus we can conclude that the two sets of coefficients aren't the same. The code that was used to compute the **largest deviation** (in absolute terms) is as follows:

```
err_coeff1 = abs(ans_arr1 - ans_lsq_arr_1)
err_coeff2 = abs(ans_arr2 - ans_lsq_arr_2)
max_dev_1 = py.amax(err_coeff1)
max_dev_2 = py.amax(err_coeff2)
```

# 9 Plotting the True and Approximated functions

We will compute $M * p$ to obtain the **approximated** values of the function. We will plot them with the true plot as follows:

Figure 1: Plot of exp(x) in semilog scale



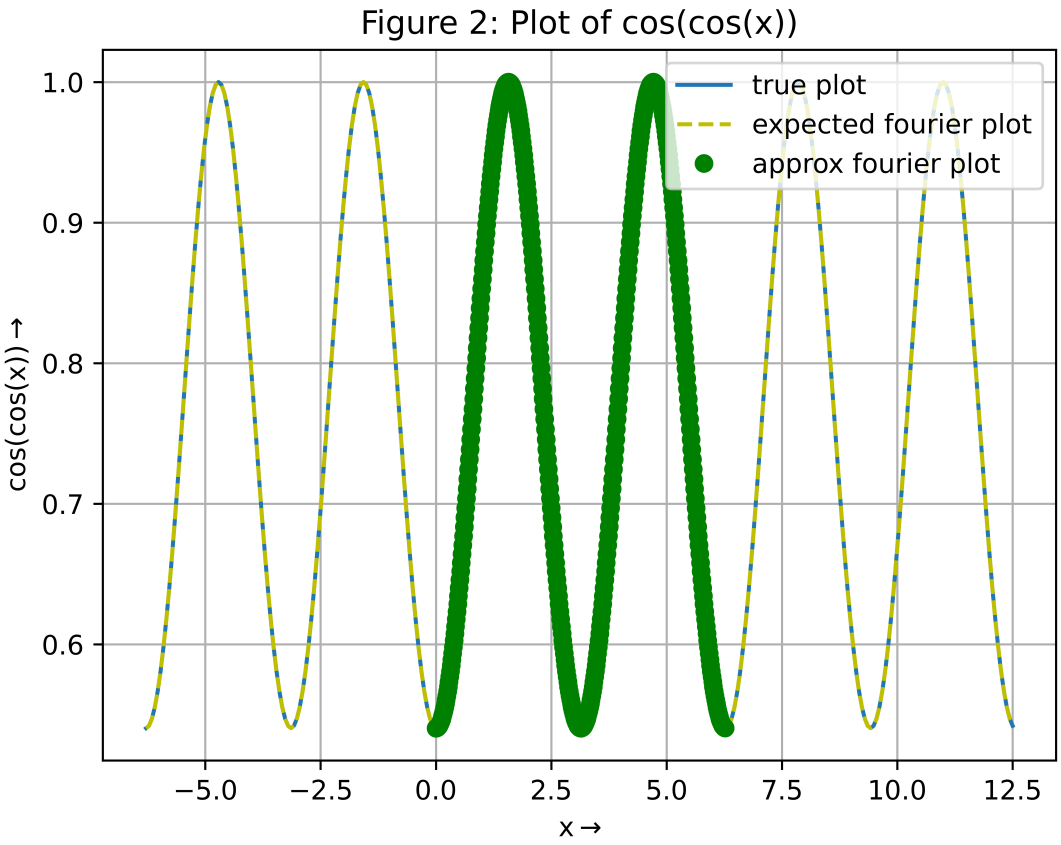


Figure 2: Plot of cos(cos(x))
true plot
expected fourier plot
approx fourier plot
cos(cos(x)) →
x →
1.0
0.9
0.8
0.7
0.6
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

The inference that we can make from the plots is as follows:

- Function $e^x$ isn't properly periodic over the interval, hence the least square method didn't give the best fit of coefficients for the sample values. Thus the **approximated** plot has quite a deviation from the **true** plot.
- Function $\cos(\cos(x))$ is properly periodic, hence the least square method was able to give the best fit of coefficients for the sample values. Thus the **approximated** plot appreciable overlaps with the **true** plot.

# 10 Conclusion

We can conclude that:

- The fourier coefficients obtained from **least square** method is different from the **integrally** obtained one as the former one gives a **best fit** possible for the given data samples.
- Hence for properly periodic function, **least square** approach gives an alternative way to obtain fourier coefficients with a lot of precision for less number of samples.
- But if we increase the number of samples to be provided to the **Least Square** function, then we can obtain fairly well precise coefficients for non periodic cases like $e^x$ too.