Assignment 4:Fourier Series

Abhishek Sekar
EE18B067

February 23, 2020

0.1 Introduction

The report discusses 7 tasks in Python to find Fourier Approximations of two functions,namely: e^x and $\cos(\cos(x))$ in two different methods.
By integration and by the least squares method.

We will fit two functions, e^x and $\cos(\cos(x))$ over the interval $[0,2\pi)$ using the fourier series

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

The equations used here to find the Fourier coefficients are as follows:

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

(2)

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

(3)

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

(4)

0.2 Python code

The following are the libraries that were imported

```python
#importing the necessary libraries

from numpy import *
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy.integrate import quad
from sys import exit
```

0.2.1 Question 1

- Define Python functions for the two functions e^x and $\cos(\cos(x))$ which return a vector (or scalar) value.

- Plot the functions over the interval $[2\pi,4\pi)$.
- Discuss periodicity of both functions
- Plot the expected functions from fourier series

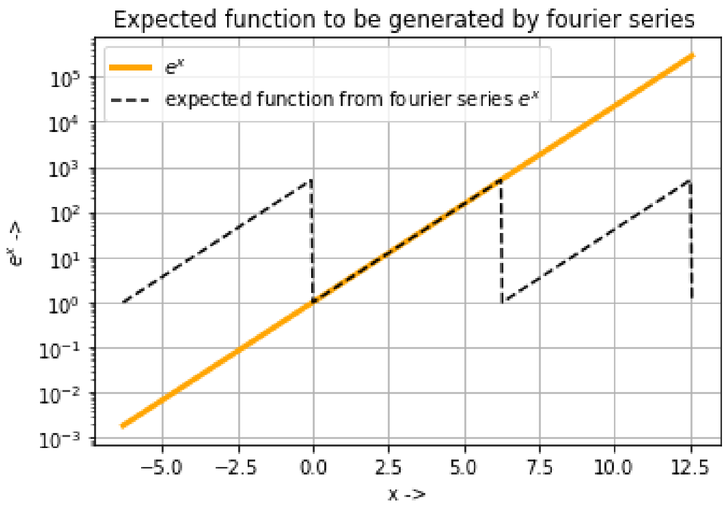
Code:

```python
#part1

def expo(x):           #function designed to calculate exp(x)

    return exp(x)

def coscos(x):       #function designed to calculate cos(cos(x))

```

```
    return np.cos(np.cos(x))

x = linspace(-2*np.pi, 4*np.pi, 400)   #linspace gives a linear

#distribution of numbers in the designated range

period=2*np.pi   #period of the function as per fourier series

#plot 1: semilog plot of exp(x) and the expected
#function from fourier series

plt.semilogy(x,expo(x),linewidth=3,label='$e^{x}$',color='orange')
plt.semilogy(x,expo(x%period),'--',label="expected function
from fourier series "$e^{x}$",color='black')
plt.xlabel("x ->")
plt.ylabel('$e^{x}$ ->')
plt.title('Expected function to be generated by fourier series ')
plt.grid()
plt.legend()
plt.show()

#plot 2: plot of cos(cos(x)) and the expected function
#from fourier series

plt.plot(x,coscos(x),linewidth=3,label='cos(cos(x))',color='yellow'
plt.plot(x,coscos(x%period),'--',label="expected function
"from fourier series",color='black')
```

```
plt.xlabel('x ->')
plt.ylabel('coscos(x) ->')
plt.title('Expected function to be generated by fourier series')
plt.grid()
plt.legend()
plt.axis([-5, 5, 0, 2])
plt.show()
```

The given snippet of code produces the following outputs on the next page.



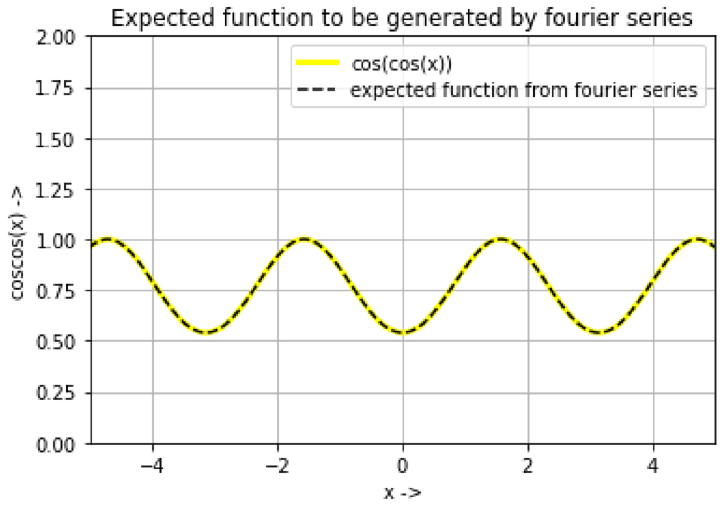


Figure 1: Plot for e^x and $cos(cos(x))$

Results and Observations :

- We observe that e^x is not periodic, whereas $\cos(\cos(x))$ is periodic as the expected and original function matched for the latter but not for e^x.
- Period of $\cos(\cos(x))$ is 2π as we observe from graph and e^x monotously increasing hence not periodic.
- We get expected function by:

- plotting expected function by dividing the x by period and giving remainder as input to the function, so that x values repeat after given period.
- That is f(x%period) is now the expected periodic function from fourier series.

0.2.2 Question 2

- Obtain the first 51 coefficients i.e $a_0, a_1, b_1, ...$ for e^x and $\cos(\cos(x))$ using scipy quad function
- And to calculate the function using those coefficients and comparing with original funcitons graphically.

Code:

```
#part2

def a_coeff(x,k,f):
    return f(x)*np.cos(k*x)

def b_coeff(x,k,f):
    return f(x)*np.sin(k*x)

def coeff_calc(f):  #this function calculates the fourier

#coefficients and packages them in a single array
    coeff_val=[]

coeff_val.append(scipy.integrate.quad(f, 0,
2*np.pi)[0]/(2*np.pi))
```

```python

    for i in range(1,26):
        coeff_val.append((scipy.integrate.quad(a_coeff, 0,
        2*np.pi, args=(i, f))[0])/np.pi)
        #giving function arguments inside quad
        coeff_val.append((scipy.integrate.quad(b_coeff, 0,
        2*np.pi, args=(i, f))[0])/np.pi)

    return coeff_val

def A_matrix(x):
    A = 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 fourier_function(c,num):  #this function obtains the
#parent function using the fourier series coefficients
#with them as the argument
    A=A_matrix(x)if(num==0)else A_matrix(x1) #num=1 for 1st
    #sq  function generation, done using conditional
    #operator
    b=dot(A,c)
    return b

#lists(arrays) store the various values present
exp_coeff=coeff_calc(expo)
coscos_coeff=coeff_calc(coscos)
exp_coeff_abs=np.abs(exp_coeff)   #np.abs provides the
#Magnitude of the argument
coscos_coeff_abs=np.abs(coscos_coeff)
exp_fourier=fourier_function(exp_coeff,0)
coscos_fourier=fourier_function(coscos_coeff,0)

#part 3

#plot 3:  semilog plot of exp(x) obtained using fourier series

#coefficients
plt.semilogy(x, exp_fourier,'ro', label="Function obtained
"using Fourier Coefficients")
plt.ylim([pow(10, -1), pow(10, 4)])
plt.xlabel('x ->')
plt.ylabel('fourier $e^{x}$ ->')
plt.title('Fourier series function: $e^{x}$')
plt.legend()
plt.grid()
plt.show()

#plot4: plot of cos(cos(x)) obtained using fourier series
```

```

#coefficients
plt.plot(x, coscos_fourier,'ro',label="Function obtained using Four
plt.legend(loc='upper right')
plt.xlabel('x ->')
plt.ylabel('fourier cos(cos(x))->')
plt.title('Fourier series function: cos(cos(x))')
plt.axis([-5, 5, -0.5, 2])
plt.grid()
plt.show()
```

The given code snippet produce the following graphs.

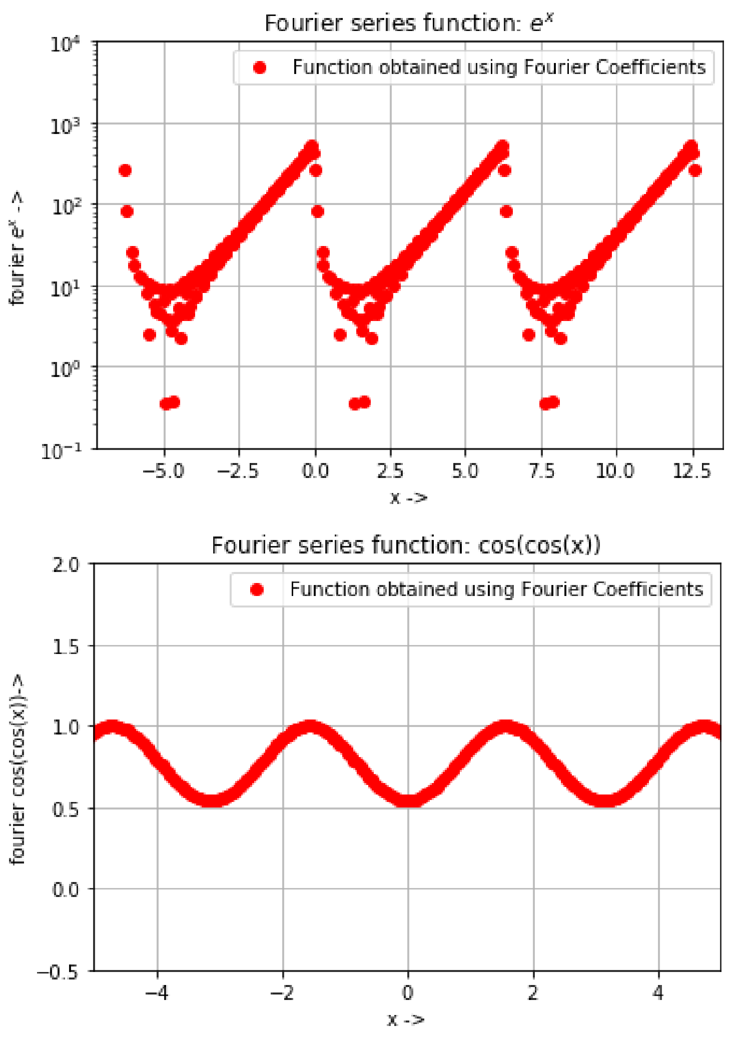


Figure 2: Plot for Fourier Approximation of e^x and $cos(cos(x))$

0.2.3 Question3

- Two different plots for each function using “semilogy” and “loglog” and plot the magnitude of the coefficients vs n
- And to analyse them and to discuss the observations. ## Plots:
- For each function magnitude of a_n and b_n coefficients which are computed using integration are plotted in same figure in semilog as well as loglog plot for simpler comparisons.

Code:

```
    #Note: an and bn are plotted in the same graph for simplicity

#plot5: semilog plot of the magnitude spectrum of exp(x)
#coefficients
plt.semilogy((exp_coeff_abs[1::2]),'ro', label="$a_{k}$s using
 Integration")   # By using array indexing methods we
 #separate all odd indexes starting from 1 (ak)
plt.semilogy((exp_coeff_abs[2::2]),'bo', label="$b_{k}$s using
 Integration")   # and all even indexes starting from 2 (bk)
plt.legend()
plt.title("Fourier coefficients of $e^{x}$ (semi-log graph)")
plt.xlabel("k ->")
plt.ylabel("Magnitude of coeffients ->")
plt.grid()
plt.show()


#plot6: loglog plot of the magnitude spectrum of exp(x)
#coefficients
plt.loglog((exp_coeff_abs[1::2]),'ro',label="$a_{k}$s using"
"Integration")
plt.loglog((exp_coeff_abs[2::2]),'bo',label="$b_{k}$s using"
 "Integration")
plt.legend(loc='upper right')
plt.title("Fourier coefficients of $e^{x}$ (Log-Log graph)")
plt.xlabel("k ->")
plt.grid()
plt.ylabel("Magnitude of coeffients ->")
plt.show()


#plot7: semilog plot of the magnitude spectrum of cos(cos(x)) coe
plt.semilogy((coscos_coeff_abs[1::2]),'ro',label="$a_{k}$s using"
 "Integration")
plt.semilogy((coscos_coeff_abs[2::2]),'bo',label="$b_{k}$s using"
 "Integration")
plt.legend(loc='upper right')
plt.title("Fourier coefficients of cos(cos(x)) (semi-log graph)")
plt.xlabel("k ->")
plt.grid()
plt.ylabel("Magnitude of coeffients ->")
plt.show()


#plot8: loglog plot of the magnitude spectrum of cos(cos(x)) coef
plt.loglog((coscos_coeff_abs[1::2]),'ro',label="$a_{k}$s using"
 "Integration")
plt.loglog((coscos_coeff_abs[2::2]),'bo',label="$b_{k}$s using"
 "Integration")
plt.legend(loc='upper right')
plt.title("Fourier coefficients of cos(cos(x)) (log-log graph)")
plt.xlabel("k ->")
plt.grid()
plt.ylabel("Magnitude of coeffients ->")
plt.show()
```

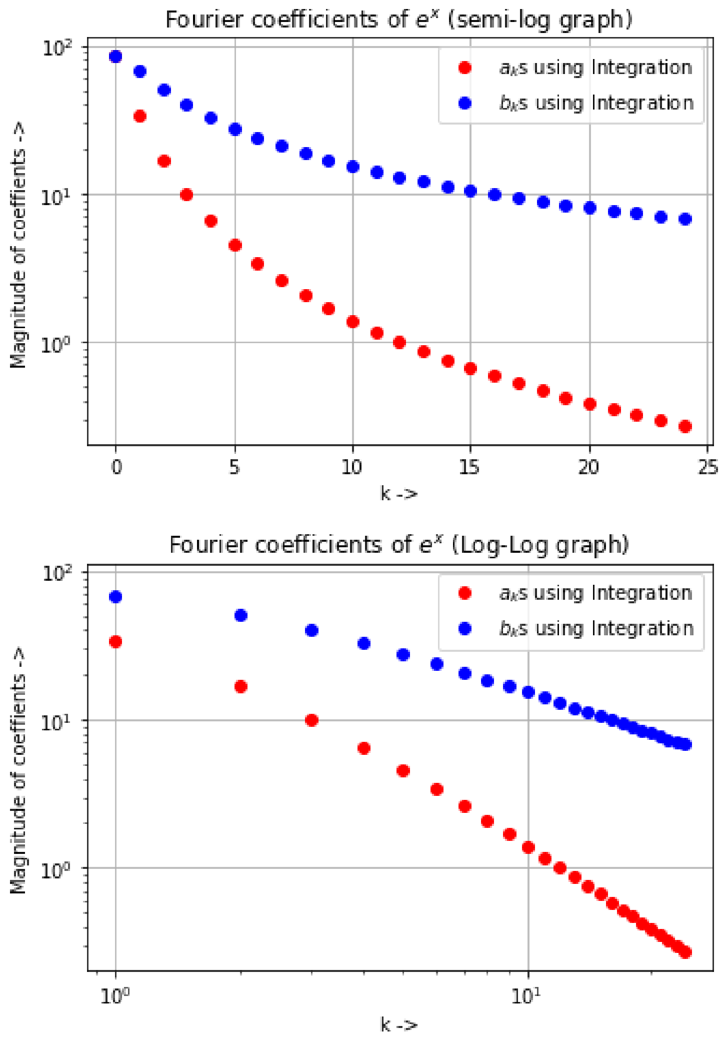


Figure 3: Semi-Log ,Log-Log plots of Fourier coefficients of e^x (Integration)

Results and Observations :

- The b_n coefficients in the second case should be nearly zero. Why does this happen?

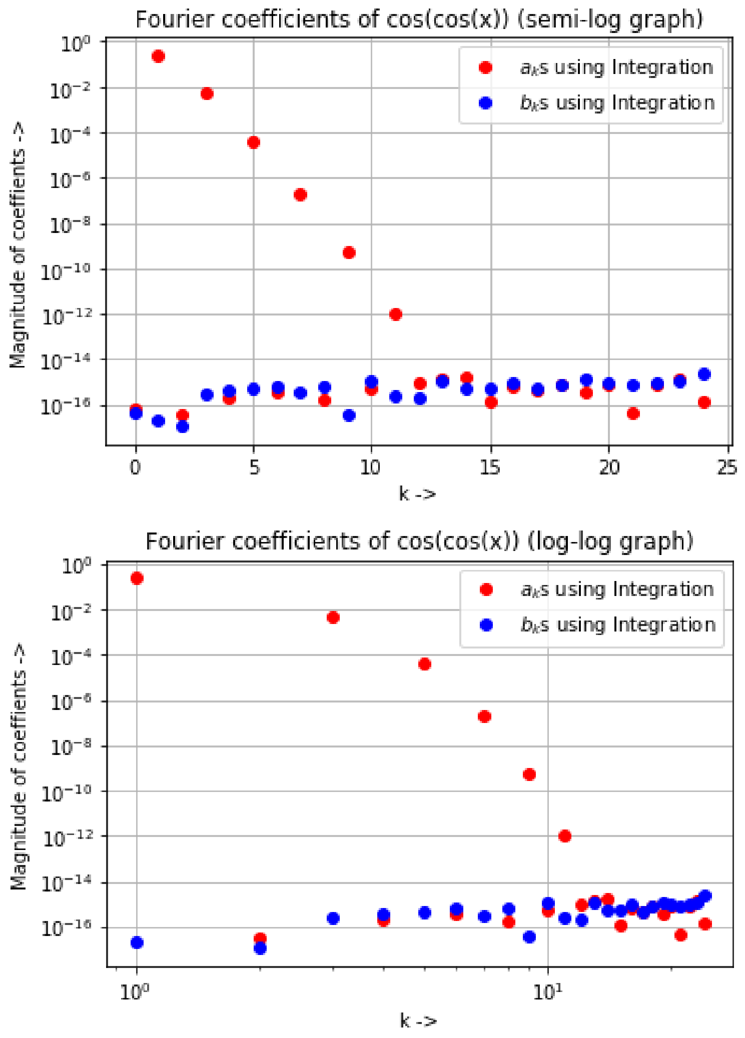


Figure 4: Semi-Log ,Log-Log plots of Fourier coefficients of $cos(cos(x))$ (Integration)

– Because $\cos(\cos(x))$ is an even function and for finding b_n we use Eq.(4) so the whole integral can be integrated in any interval with length of 2π, so for convenience we choose $[-\pi, \pi)$, then the integrand is odd since $\sin(nx)$ is there. so the integral becomes zero analytically. Where as here we compute using quad function which uses numerical methods so b_n is very small but not exactly zero.

- In the first case, the coefficients do not decay as quickly as the coefficients for the second case. Why not?
  - Rate of decay of fourier coefficients is determined by how smooth the
    function is,if a function is infinitely differentiable then its fourier coefficients decays very faster, where as if k^{th} derivative of function is discontinous the coefficients falls as $\frac{1}{n^{k+1}}$. to atleast converge.So in first case i.e is e^x is not periodic hence discontinous at $2n\pi$ so the function itself is discontinous so coefficients falls as $\frac{1}{n}$ so we need more coefficients for more accuracy,coefficients doesn't decay as quickly as for $\cos(\cos(x))$ as it is infinitely differentiable and smooth so we need less no of coefficients to reconstruct the function so it decays faster.
- Why does loglog plot in Figure 4 look linear, wheras the semilog plot in Figure 5 looks linear?
  - Because the coefficients of e^x varies as n^k where as $\cos(\cos(x))$ varies exponentially with 'n' means α^n , thats why loglog looks linear in first case and semilog in second case.

0.2.4 Question 4 & 5

- Uses least squares method approach to find the fourier coefficients of e^x and $\cos(\cos(x))$
- Evaluate both the functions at each x values and call it b. Now this is approximated by
  $a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx)$
- such that

$$a_0 + \sum_{n=1}^{\infty} a_n \cos(nx_i) + b_n \sin(nx_i) \approx f(x_i) \tag{5}$$

- To implement this we use matrices to find the coefficients using Least Squares method using inbuilt python function 'lstsq'

Code:

```
    #part 4

def lstsq_coeff(f): #function to calculate the coefficients using

    x1 = linspace(0,2*np.pi,401)
    x1 = x1[:-1]            # drop last term to have a proper peri
    b = f(x1)
    A = A_matrix(x1)        #uses this function declared earlier to
    c = scipy.linalg.lstsq(A, b)[0]  # the '[0]' is to pull out th
    return c
```

```


# storing coefficients in respective vectors.
coeff_exp = lstsq_coeff(expo)
coeff_coscos = lstsq_coeff(coscos)

coeff_exp_abs = np.abs(coeff_exp)
coeff_coscos_abs= np.abs(coeff_coscos)

#part5

#plot9:semilog plot of the magnitude spectrum of exp(x) coefficie
plt.semilogy((coeff_exp_abs[1::2]),'go',label="$a_{k}$s
using Least Squares")
plt.semilogy((coeff_exp_abs[2::2]),'bo',label="$b_{k}$s
using Least Squares")
plt.grid()
plt.legend(loc='upper right')
plt.title("Fourier coefficients of $e^{x}$ (semilog graph)")
plt.xlabel("k ->")
plt.ylabel("Magnitude of coeffients ->")
plt.show()

#plot10:loglog plot of the magnitude spectrum of exp(x) coefficie
plt.loglog((coeff_exp_abs[1::2]),'go',label="$a_{k}$s
using Least Squares ")
plt.loglog((coeff_exp_abs[2::2]),'bo',label="$b_{k}$s
using Least Squares")
plt.grid()
plt.legend(loc='lower left')
plt.title("Fourier coefficients of $e^{x}$ (log-log graph)")
plt.xlabel("k ->")
plt.ylabel("Magnitude of coeffients ->")
plt.show()

#plot11:semilog plot of the magnitude spectrum of cos(cos(x)) coe
plt.semilogy((coeff_coscos_abs[1::2]),'go',label="$a_{k}$s using
Least Squares")
plt.semilogy((coeff_coscos_abs[2::2]),'bo',label="$b_{k}$s using
 Least Squares")
plt.grid()
plt.legend(loc='upper right')
plt.title("Fourier coefficients of cos(cos(x)) (semilog graph)")
plt.xlabel("k ->")
plt.ylabel("Magnitude of coeffients ->")
plt.show()

#plot12:loglog plot of the magnitude spectrum of cos(cos(x)) coef
plt.loglog((coeff_coscos_abs[1::2]),'go',label="$a_{k}$s
using Least Squares")
plt.loglog((coeff_coscos_abs[2::2]),'bo',label="$b_{k}$s
using Least Squares")
plt.grid()
plt.legend(loc='best')
plt.title("Fourier coefficients of cos(cos(x)) (log-log graph)")
plt.xlabel("k ->")
plt.ylabel("Magnitude of coeffients ->")
plt.show()
```

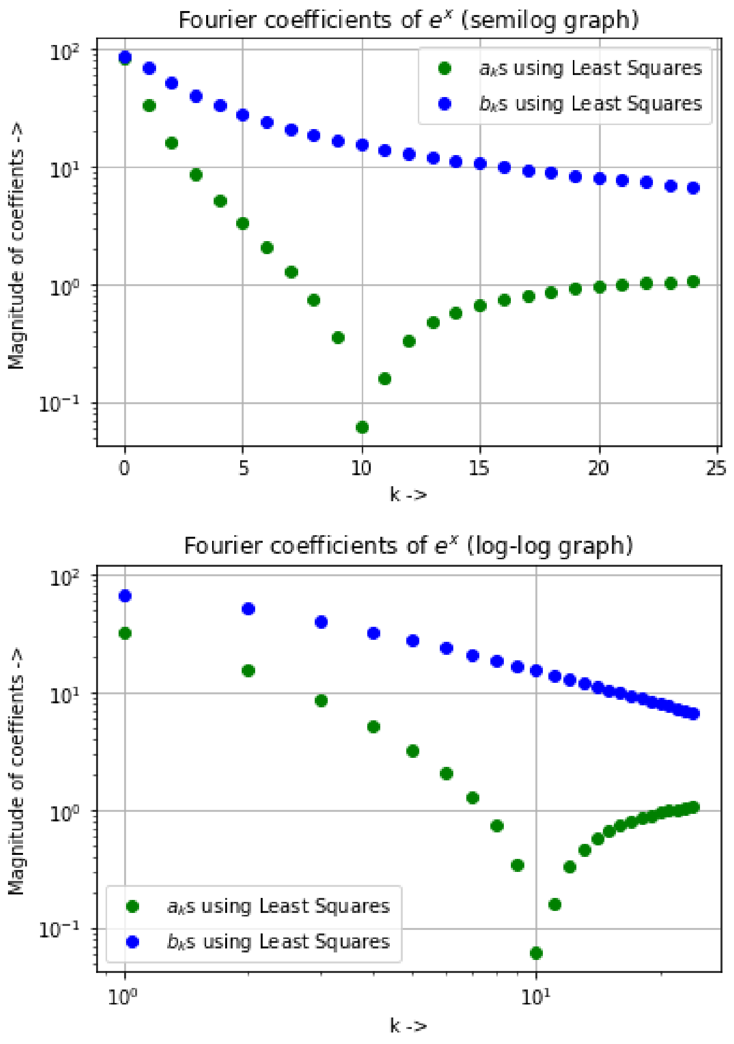


Figure 5: Semi-Log ,Log-Log plots of coefficients of e^x(Least Squares)

0.2.5 Question 6

- To compare the answers got by least squares and by the direct

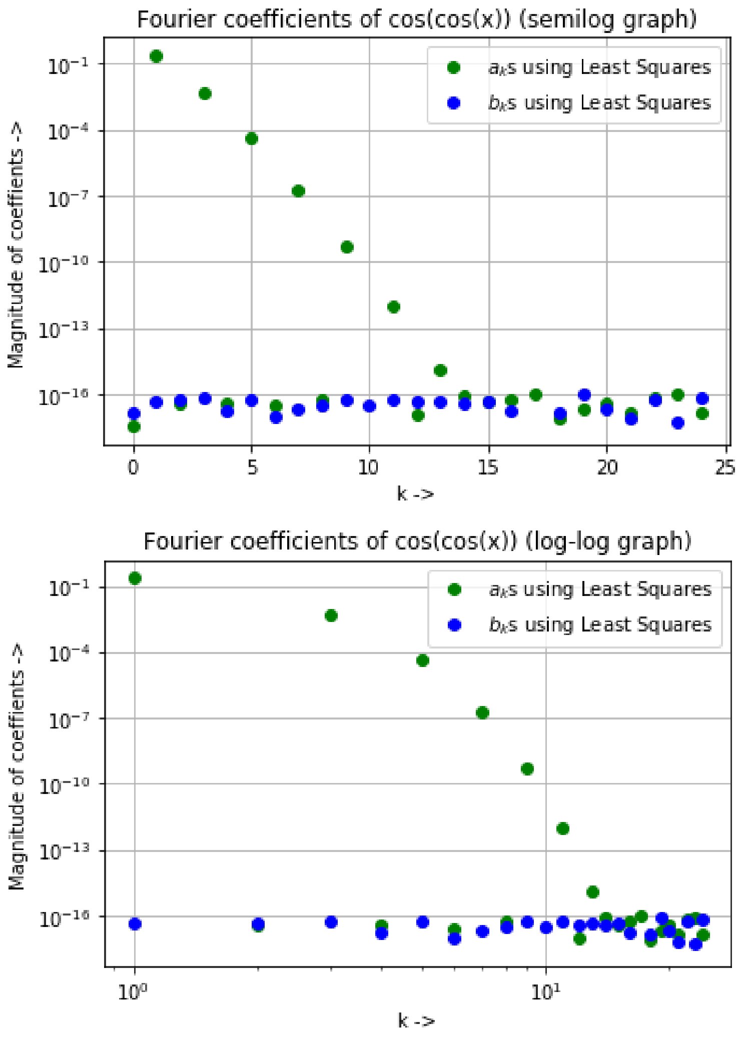


Figure 6: Semi-Log ,Log-Log plots of coefficients of $cos(cos(x))$(Least Squares)

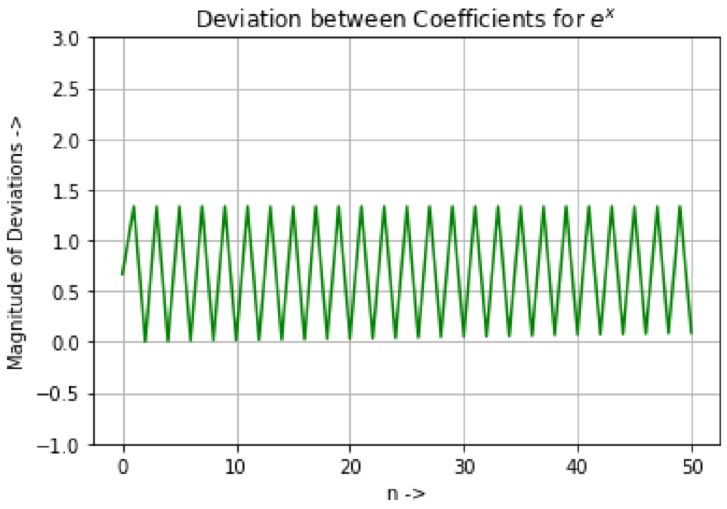
integration.

- And finding deviation between them and find the largest deviation using Vectors

Code:

```
#part 6
```

```

def coeff_compare(f): # Function to compare the coefficients got b

    max_dev = 0
    deviations = np.abs(exp_coeff - coeff_exp) if(f==1) else
     np.abs(coscos_coeff - coeff_coscos)
     #conditional operator 1 : exp(x),2 : cos(cos(x))
    max_dev = np.amax(deviations) #finds maximum value of deviatio

    return deviations, max_dev

dev1, maxdev1 = coeff_compare(1)
dev2, maxdev2 = coeff_compare(2)

print("Maximum deviation obtained in
comparing coefficients of $e^{x}$ : ", around(maxdev1,4))
print("Maximum deviation obtained in comparing
coefficients of cos(cos(x)) : ", around(maxdev2,4))


#plot13:Plotting the deviation vs n for exp(x)

plt.plot(dev1, 'g')
plt.title("Deviation between Coefficients for $e^{x}$")
plt.grid()
plt.xlabel("n ->")
plt.ylabel("Magnitude of Deviations ->")
plt.ylim([-1,3])
plt.show()


#plot 14: Plotting the deviation vs n for cos(cos(x))

plt.plot(dev2, 'g')
plt.title("Figure 8 : Deviation between coefficients
for $\cos(\cos(x))$")
plt.grid()
plt.xlabel("n ->")
plt.ylabel("Magnitude of Deviations ->")
plt.show()
```



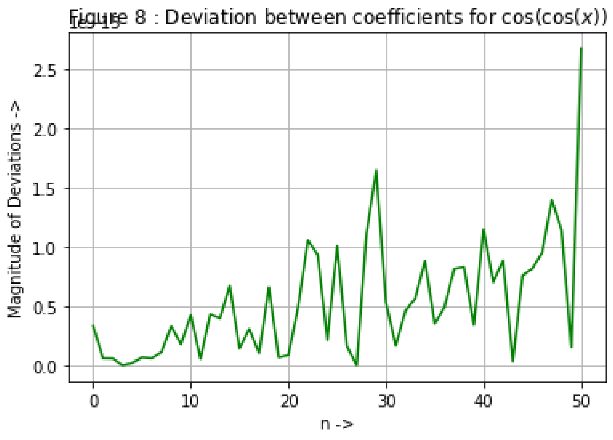


Figure 7: Deviation plots of Fourier coefficients of both the functions found via integration and least-squares

Results and Discussion :

- The maximum deviation for (rounded to four decimal places):
  - exp(x) = 1.3327
  - cos(cos(x)) = 0.0

0.2.6 Question 7

- Computing Ac i.e multiplying Matrix A and Vector C from the estimated

  values of Coeffient Vector C by Least Squares Method.

- To Plot them (with green circles) in Figures 1 and 2 respectively for

  the two functions.

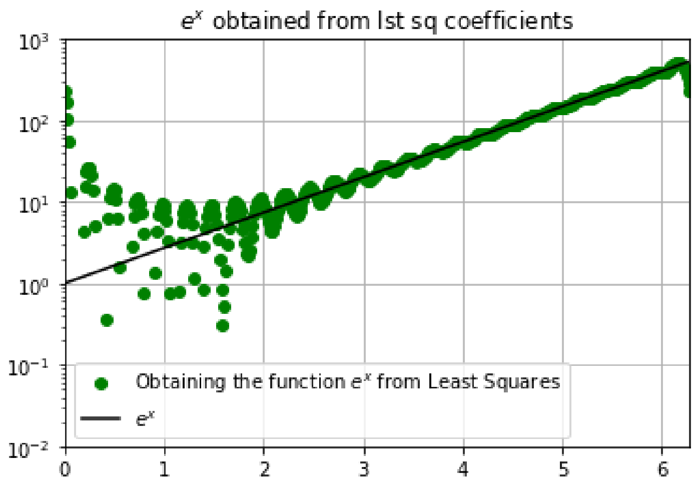
Code:

```
#part 7

# Define x1 from 0 to 2pi

x1 = linspace(0, 2*np.pi, 400)

exp_fn_lstsq = fourier_function(coeff_exp,1) #uses fourier_functio
coscos_fn_lstsq = fourier_function(coeff_coscos,1)



# comparing plots between lst square and the actual function
#plot 15 for exp(x)
plt.semilogy(x1, exp_fn_lstsq, 'go',label="Obtaining the
function $e^{x}$ from Least Squares")
plt.semilogy(x1,expo(x1),label='$e^{x}$',color='black')
plt.legend()
plt.grid()
plt.title('$e^{x}$ obtained from lst sq coefficients')
plt.axis([0,2*np.pi,pow(10, -2), pow(10, 3)])
```

```python
plt.show()

#plot 16 for cos(cos(x))
plt.plot(x1,coscos(x1),label='cos(cos(x))',color='black')
plt.legend()
plt.grid()
plt.title('cos(cos(x)) obtained from lst sq coefficients')
plt.axis([0, 2*np.pi,0.5, 1.3])
plt.show()
```

Results and Discussion :

- As we observe that there is a significant deviation for e^x as it has discontinuites at $2n\pi$ which can be observed in Figure 1 and so there will be **Gibbs phenomenon** i.e there will be oscillations around the discontinuity points and their ripple amplitude will decrease as we go close to discontinuity. In this case it is at 2π for e^x.
- As we observe that rimples are high in starting and reduces and oscillate with more frequency as we go towards 2π. This phenomenon is called **Gibbs Phenomenon**
- Due to this. the orginal function and one which is reconstructed using least squares will not fit exactly.
- And as we know that Fourier series is used to define periodic signals in frequency domain and e^x is a aperiodic signal so you can't define an aperiodic signal on an interval of finite length (if you try, you'll lose information about the signal), so one must use the Fourier transform for such a signal.



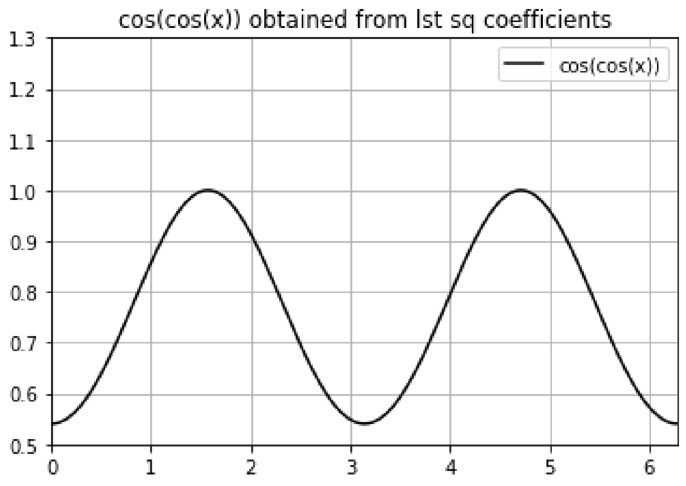


Figure 8: Plots of the functions got by Inverse Fourier Transform

- Thats why there are significant deviations for e^x from original function.
- Whereas for $\cos(\cos(x))$ the curves fit almost perfectly because the function itself is a periodic function and it is a continous function in entire x range so we get very negligible deviation and able to reconstruct the signal with just the fourier coefficients.

0.3 Conclusion

We see that the fourier estimation of e^x does not match significantly with the function close to 0, but matches near perfectly in the case of $\cos(\cos(x))$. This is due to the presence of a discontiuity at $x = 0$ for the periodic extension of e^x. This discontiuity leads to non-uniform convergence of the fourier series, with different rates for both the functions.

The difference in the rates of convergence leads to the phenomenon**Gibb's**
phenomenon, which is the ringing observed at discontiuities in the
fourier estimation of a discontiuous function.
This explains the mismatch in the fourier approximation for e^x.
Thus we can conclude that the Fourier Series Approximation Method works extremely well
for smooth periodic functions, but gives bad results for discontinuos periodic functions