Assignment No. 4

Srivenkat A (EE18B038)

February 26, 2020

1 Abstract

The assignment focuses on Fourier Approximations. Mathematically, a Fourier series is an expansion of a periodic function f(x) in terms of an infinite sum of sines and cosines. Here, we approximate the infinite summation to a finite summation and we get a approximate function for the given function in limited number of terms of sines and cosines.

- Determining function's periodicity
- Finding fourier coefficients by two different methods
- Doing integration of functions
- Plotting graphs

2 Introduction

We analyse the following 2 functions, exp(x) and cos(cos(x)). These are fitted over the interval $[0, 2\pi)$ using the fourier series that is approximated as a summation only till N coefficients.

3 Formulae

Fourier Series:

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

Coefficients:

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

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

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

4 Tasks

4.1 Task 1:

Two functions exp_create(time) and coscos_create(time) are defined for calculating exponential or cosofcos(x)) which returns a vector (or scalar) for a vector (or scalar) input. Inbuilt numpy functions are used for the finding the values of the functions The function that is expected from the fourier series is also plotted in Figure 1 and 2.

```
def exp_create(time):
    return np.exp(time)

def cosofcos(x, switch = 2):
    return np.cos(x) if switch == 1 else np.cos(np.cos(x))
```

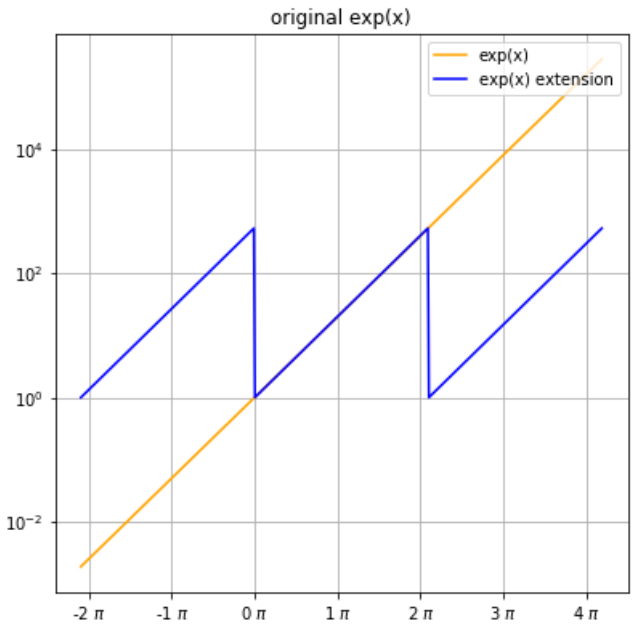

Figure 1:

The periodicity is determined by the function ***perkey,data)*** which takes input as the function and returns whether or not if it is periodic.

```
data = np.round(data,3)
check = 0
per = -1
for i in range(1,len(data)):
    per = -1
```

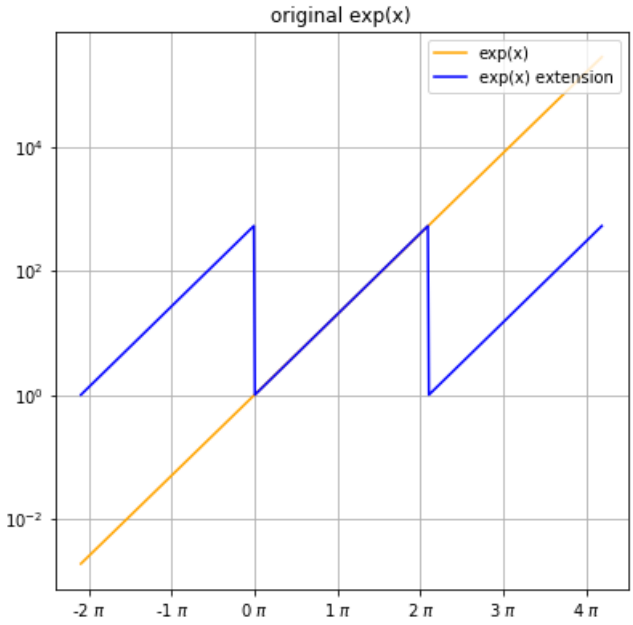

Figure 2:

```
    if data[i] == data[0]:
        per = i
        for j in range(1,len(data)-per):
            if data[j+per] != data[j]:
                break
            elif j == len(data)-per-1:
                check = 1
                break
    if check == 1:
        print('{0}(x) function is periodic'.format(func_array[key]))
        break
if check == 0:
    print('{0}(x) function is not periodic'.format(func_array[key]))
```

4.2 Task 2

Using the formulae for finding coefficients for fourier series, the first 26 coefficients are obtained. The *scipy.integrate.quad* was used for the integration. Four functions u1(x,k), u2(x,k) and v1(x,k), v2(x,k) was created to pass as arguments to the integrate function. The obtained coefficients are stored in variable coeff1 and coeff2.

4.3 Task 3

The magnitudes of the coefficients was plotted against n using both "loglog" and "semilogy" plots, and shown in Figure 3,4,5 and 6.

4.3.1 a)

The b_n of cos(cos(x)) is nearly zero because the integrand is a odd function about π (midpoint of range of integration independent variable) The very small values obtained are due to the limited accuracy to store the value PI and approximations taken in integration.

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

4.3.2 b)

The coefficients do not decay easily because exp(x) takes high values in the interval 0 to 2π near 2π. Basically, the exp(x) is ever increasing and has to be made of many frequencies. But cos(cos(x)) is periodic with frequency of $\frac{1}{\pi}$, and can be made up of fewer fundamental frequencies.

4.3.3 c)

For exp(x), we observe from below equations

$$a_k = \left(\frac{1}{\pi}\right)\frac{e^{2\pi}-e^0}{1+k^2}\ ,\ b_k = \left(\frac{-k}{\pi}\right)\frac{e^{2\pi}-e^0}{1+k^2}$$

that a_k and b_k is proportional to $\frac{1}{1+k^2}$. So, $log(a_k)$ and $log(b_k) \propto -log(k)$. Hence, it is linear for loglog plot in Figure 4. Whereas for $cos(cos(x))$, $loga_k$ and $logb_k$ are $\propto$ n, and hence the semilogy plot is linear.

4.4 Task 4

Using lstsq (least squares method), 51 coefficients for two functions were found for the range 0 to 2π in 400 steps using the matrix equation (Ac = b)

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

The least squares estimate is found out by the following function

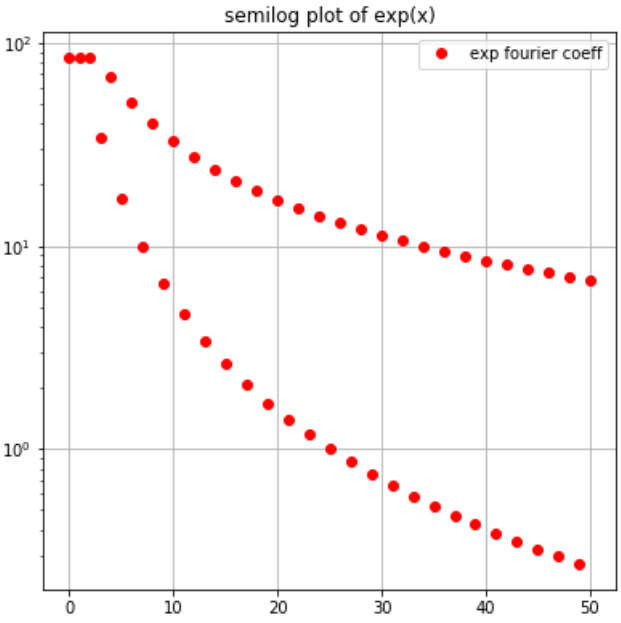

Figure 3:

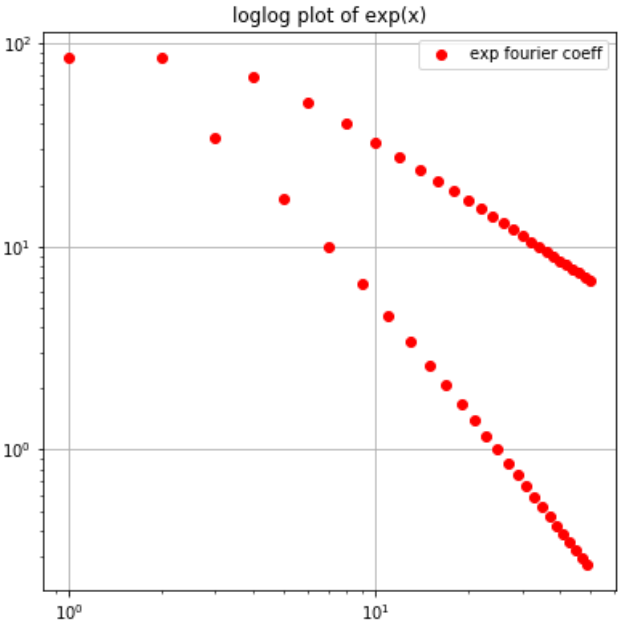

Figure 4:

```
A = np.zeros(shape=[400,51])
b = np.zeros(shape=[400,1])
```

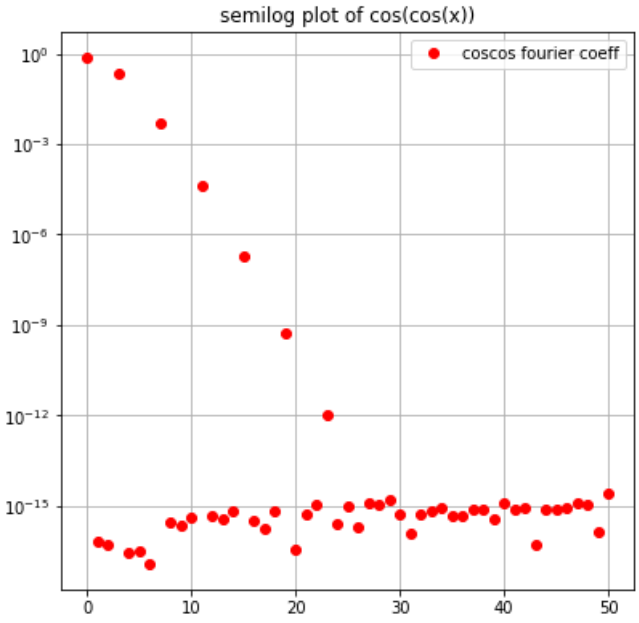

Figure 5:

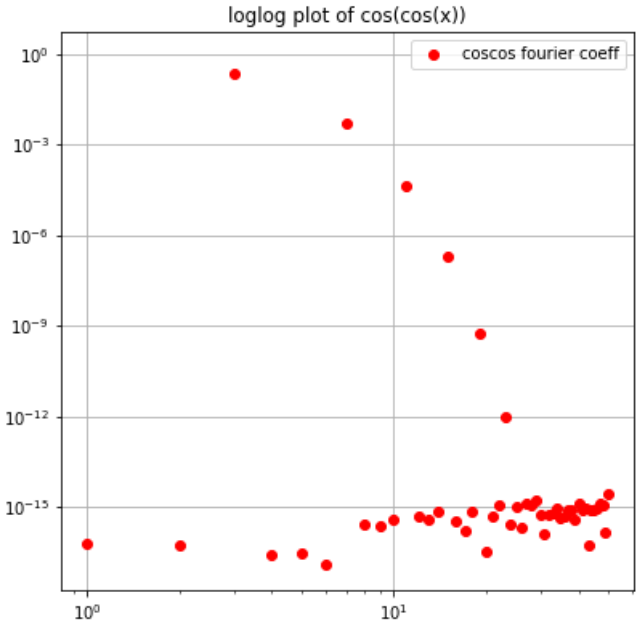

Figure 6:

```
A[:,0] = 1
b = np.transpose(func(key,x))
```

```
for k in range(1,26):
    A[:,2*k-1] = np.cos(k*x)
    A[:,2*k] = np.sin(k*x)
return sp.linalg.lstsq(A,b)[0],A,b
```

4.5 Task 5

The *lstsq* was used to solve the Ac = b equation and coefficient matrix e_vect_coeff and cc_vect_coeff were obtained. These were plotted in the Figure 3,4,5 and 6 using green circles.

4.6 Task 6

The absolute difference in the coefficients were found as:

```
Max error in finding fourier coefficients from integration is 1.3327308703353395 in
```

They agree with a difference that is negligible compared to the order of the values. The error is zero for cos(cos(x)) coefficients. The largest deviation for exp(x) is 1.33 and for cos(cos(x)) is 0 (of the order e-15).

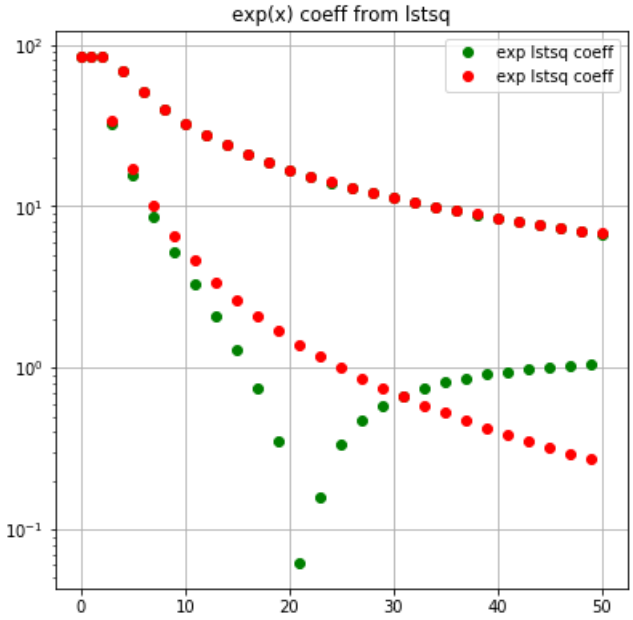

Figure 7:

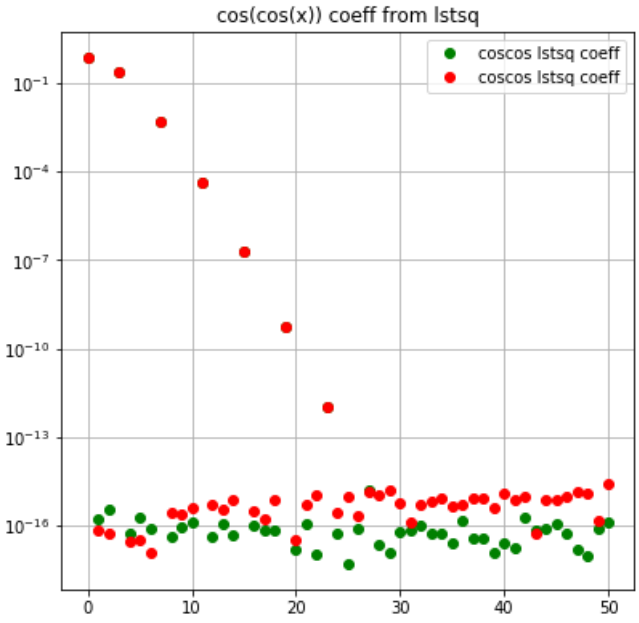

Figure 8:

4.7 Task 7

The fourier series obtained from least squares method by multiplying A matrix and coefficient matrix c as (Ac) is plotted in Figure 9 and 10 with green circles. The periodic functions are represented using fourier series. When we plot exp(x) using fourier series, then the deviation occurs as it is not periodic. But when we plot cos(cos(x)) using fourier series, it is very much aligned as it is a periodic function.

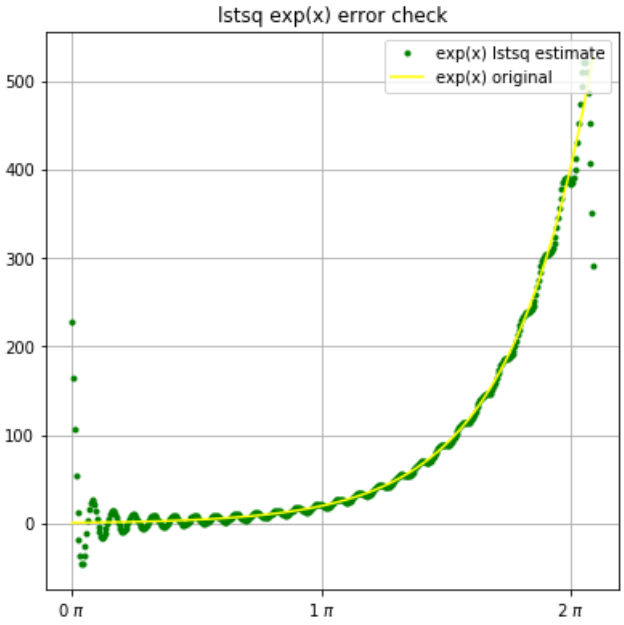

Figure 9:

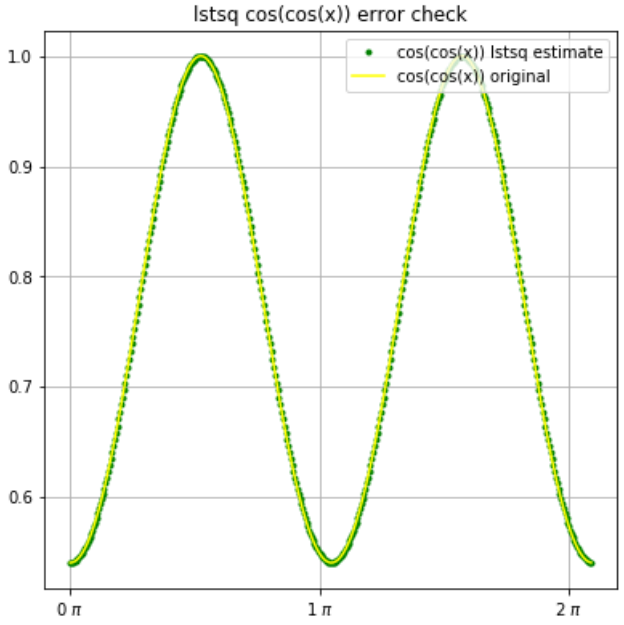

Figure 10:

5 Python code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 12 22:12:05 2020

@author: srivenkat
"""
import numpy as np
import scipy as sp
import scipy.integrate as integrate
import math as ma
import matplotlib.pyplot as plt
import matplotlib.ticker as tck

def exp_create(time):
    return np.exp(time)

def coscos_create(time):
    return np.cos(np.cos(time))

def func(key,x):
    if key == 0:
        return np.exp(x)
    else:
        return np.cos(np.cos(x))

#fourier series generation by direct integration
def u_gen(x,k,key):
    return np.cos(k*x)*func(key,x)

def v_gen(x,k,key):
    return np.sin(k*x)*func(key,x)

def fourier_coeff(key,l,a):
    a.append(integrate.quad(u_gen,0,2*ma.pi,args=(0,key))[0]/(2*ma.pi))
    for k in range(1,l):
        a.append(integrate.quad(u_gen,0,2*ma.pi,args=(k,key))[0]/ma.pi)
        a.append(integrate.quad(v_gen,0,2*ma.pi,args=(k,key))[0]/ma.pi)

def fourier_synth(t,a):
    f_val = a[0]
    n = 1
    while n < len(a):
        f_val += a[n]*sp.cos((n+1)*t/2) + a[n+1]*sp.sin((n+1)*t/2)
```

```
        n = n + 2
    return f_val

def per_check(key,data):
    data = np.round(data,3)
    check = 0
    per = -1
    for i in range(1,len(data)):
        per = -1
        if data[i] == data[0]:
            per = i
            for j in range(1,len(data)-per):
                if data[j+per] != data[j]:
                    break
                elif j == len(data)-per-1:
                    check = 1
                    break
        if check == 1:
            print('{0}(x) function is periodic'.format(func_array[key]))
            break
    if check == 0:
        print('{0}(x) function is not periodic'.format(func_array[key]))

time = np.linspace(-2*ma.pi,4*ma.pi,501)
l = 26
func_array = ['exp','coscos']
period = np.linspace(0,2*ma.pi,167)
exp_x = exp_create(time)
cos_cos_x = coscos_create(time)
exp_x_per = exp_create(period)
exp_x_perext = np.concatenate([exp_x_per,exp_x_per])
exp_x_perext = np.concatenate([exp_x_perext,exp_x_per])
e_coeff = []
f_seq_e = []
cc_coeff = []
f_seq_cc = []

fourier_coeff(0,l,e_coeff)
fourier_coeff(1,l,cc_coeff)

for t in time:
    f_seq_e.append(fourier_synth(t,e_coeff))
    f_seq_cc.append(fourier_synth(t,cc_coeff))
```

```
figure , graph1 = plt.subplots(figsize=(6.4,6.4))
graph1.semilogy(time/3,exp_x,'orange',label='exp(x)')
graph1.semilogy(time/3,exp_x_perext,'blue',label='exp(x) extension')
graph1.legend(loc='upper right')
graph1.set_title('original exp(x)')
graph1.xaxis.set_major_formatter(tck.FormatStrFormatter('%g $\pi$'))
graph1.xaxis.set_major_locator(tck.MultipleLocator(base=1.0))
graph1.grid('true')

figure , graph2 = plt.subplots(figsize=(6.4,6.4))
graph2.plot(time/3,cos_cos_x,'orange',label='cos(cos(x))')
graph2.legend(loc='upper right')
graph2.set_title('original cos(cos(x))')
graph2.xaxis.set_major_formatter(tck.FormatStrFormatter('%g $\pi$'))
graph2.xaxis.set_major_locator(tck.MultipleLocator(base=1.0))
graph2.grid('true')

plt.show()

print('')   #just for output alignment
per_check(0,exp_x)
per_check(1,cos_cos_x)

figure , graph3 = plt.subplots(figsize=(6.4,6.4))
graph3.semilogy(e_coeff,'ro',label='exp fourier coeff')
graph3.set_title('semilog plot of exp(x)')
graph3.legend(loc='upper right')
graph3.grid('true')

figure , graph4 = plt.subplots(figsize=(6.4,6.4))
graph4.loglog(e_coeff,'ro',label='exp fourier coeff')
graph4.set_title('loglog plot of exp(x)')
graph4.legend(loc='upper right')
graph4.grid('true')

figure , graph5 = plt.subplots(figsize=(6.4,6.4))
graph5.semilogy(cc_coeff,'ro',label='coscos fourier coeff')
graph5.set_title('semilog plot of cos(cos(x))')
graph5.legend(loc='upper right')
graph5.grid('true')

figure , graph6 = plt.subplots(figsize=(6.4,6.4))
graph6.loglog(cc_coeff,'ro',label='coscos fourier coeff')
graph6.set_yscale('log')
```

```
graph6.set_xscale('log')
graph6.set_title('loglog plot of cos(cos(x))')
graph6.legend(loc='upper right')
graph6.grid('true')

plt.show()

#fourier series generation by least squares alg
vect_x = np.linspace(0,2*ma.pi,400)
exp_vect = exp_create(vect_x)
coscos_vect = coscos_create(vect_x)

def lstsq_coeff(key,x):
    A = np.zeros(shape=[400,51])
    b = np.zeros(shape=[400,1])
    A[:,0] = 1
    b = np.transpose(func(key,x))
    for k in range(1,26):
        A[:,2*k-1] = np.cos(k*x)
        A[:,2*k] = np.sin(k*x)
    return sp.linalg.lstsq(A,b)[0],A,b

e_vect_coeff,e_cmat,e_bmat = lstsq_coeff(0,vect_x)
cc_vect_coeff,cc_cmat,cc_bmat = lstsq_coeff(1,vect_x)

vect_seq_e = np.dot(e_cmat,e_vect_coeff)
vect_seq_cc = np.dot(cc_cmat,cc_vect_coeff)

figure, graph7 = plt.subplots(figsize=(6.4,6.4))
graph7.semilogy(e_vect_coeff,'go',label='exp lstsq coeff')
graph7.set_title('exp(x) coeff from lstsq')
graph7.legend(loc='upper right')
graph7.grid('true')

figure, graph8 = plt.subplots(figsize=(6.4,6.4))
graph8.semilogy(cc_vect_coeff,'go',label='coscos lstsq coeff')
graph8.set_title('cos(cos(x)) coeff from lstsq')
graph8.legend(loc='upper right')
graph8.grid('true')

e_error = abs(np.subtract(e_coeff,e_vect_coeff))
e_maxerr = np.amax(e_error)
cc_error = abs(np.subtract(cc_coeff,cc_vect_coeff))
cc_maxerr = np.amax(cc_error)
```

```
print("\nMax error in finding fourier coefficients from integration is {}

figure, graph9 = plt.subplots(figsize=(6.4,6.4))
graph9.plot(vect_x/3,vect_seq_e,'g.',label='exp(x) lstsq estimate')
graph9.plot(vect_x/3,e_bmat,'yellow',label='exp(x) original')
graph9.legend(loc='upper right')
graph9.set_title('lstsq exp(x) error check')
graph9.xaxis.set_major_formatter(tck.FormatStrFormatter('%g $\pi$'))
graph9.xaxis.set_major_locator(tck.MultipleLocator(base=1.0))
graph9.grid('true')

figure, graph10 = plt.subplots(figsize=(6.4,6.4))
graph10.plot(vect_x/3,vect_seq_cc,'g.',label='cos(cos(x)) lstsq estimate'
graph10.plot(vect_x/3,cc_bmat,'yellow',label='cos(cos(x)) original')
graph10.legend(loc='upper right')
graph10.set_title('lstsq cos(cos(x)) error check')
graph10.xaxis.set_major_formatter(tck.FormatStrFormatter('%g $\pi$'))
graph10.xaxis.set_major_locator(tck.MultipleLocator(base=1.0))
graph10.grid('true')

plt.show()
```