Fourier Approximations

Sankalp Saoji (EE16B063)

Date: 16/02/2018

Abstract

In this week, we got to work on fourier series and related computations. We learnt to calculate the fourier series coefficients using integration through 'quad' and also through matrices using 'least squares'. At the end, we learnt to find the maximum deviation in the output of these two methods.

1 INTRODUCTION

In this week, we calculated the fourier series coefficients using,

1) Direct Integration with 'quad'

2) Matrices with 'least squares'

1.1 Functions Used:

We fit two functions, exp(x) and cos (cos(x)) over the interval [0, 2π) using the fourier series,

$a_0 + \sum_1^\infty \{a_n \cos(nx) + b_n \sin(nx)\}$

The coefficients a_n and b_n are given by,

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

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

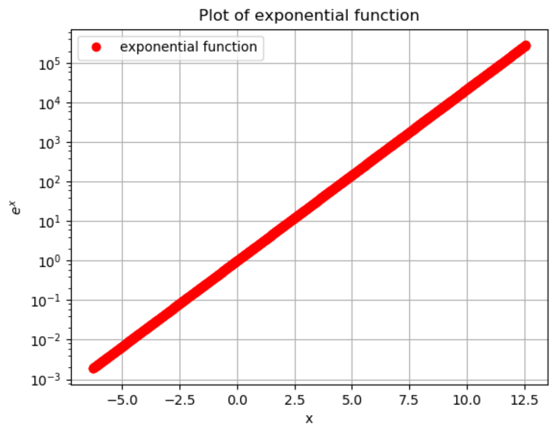
$b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(nx)dx$

2 Lab Questions

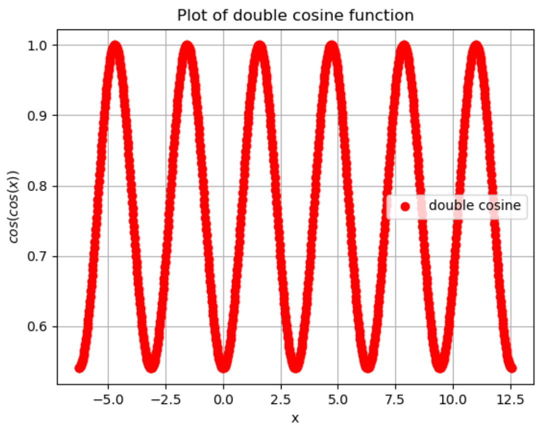
2.1 Function declaration and plotting:

In this question we were supposed to define the exponential and double cosine function in python over an interval [-2π, 4π].

```
# Importing modules

from scipy.integrate import quad
from pylab import *
from matplotlib import pyplot as plt
import numpy


    # Defining functions

    def ex(r1, r2):
            t = linspace(r1,r2,1000)
            m = numpy.exp(t)
            plt.figure(1)
            plt.semilogy(t, m, 'ro', label = 'exponential function')
            plt.xlabel('x')
            plt.ylabel('$e^x$')
            plt.title('Plot of exponential function')
            plt.grid(True)
            plt.legend()
            plt.show()

    def double_cos(r1, r2):
            t = linspace(r1,r2,1000)
            n = numpy.cos(cos(t))
            plt.figure(2)
            plt.plot(t, n, 'ro', label = 'double cosine')
            plt.xlabel('x')
            plt.ylabel('$cos(cos(x))$')
            plt.title('Plot of double cosine function')
            plt.grid(True)
            plt.legend()
            plt.show()
```



Since the e^x varies rapidly, we use semilogy for this plot. As we see from the graph, e^x is not periodic.



As we see from the graph, cos(cos(x)) is periodic.

Since Fourier series approximation is accurate for periodic functions we expect a accurate function to be generated for second function but not for the first function.

2.2 Getting a required amount of coefficients:

In this question we were supposed to get 51 coefficients of the series. We use a 'for' loop here. We create two new function and integrate them using 'quad'.

To integrate these, use the option in quad to pass extra arguments to the function being integrated:

rtnval=quad(u,0,2*pi,args=(k)).

What this does is it accepts a function u(x, ...); the integration is over x, but the k values is passed to the function by quad as the second argument. So we define the two

functions as having k as their second (not first) argument.

```
# Definig extra functions for the fourier series

def double_cosine_cosine(x,k):
        n = numpy.cos(cos(x))
        v = n*cos(k*x)
        return v

def double_cosine_sine(x,k):
        n = numpy.cos(cos(x))
        v = n*sin(k*x)
        return v
```

```
# Using 'for' loop and printing

print('Below are exp_cosine coefficient values')

for i in range(0,51):
        e = quad(exp_cosine, 0, 2*(math.pi), args=(i))[0]
        print('%d : %f'% (i, e))

print('Below are exp_sine coefficient values')

for j in range(0,51):
        f = quad(exp_sine, 0, 2*(math.pi), args=(j))[0]
        print('%d : %f'% (j, f))

print('Below are double_cosine_cosine coefficient values')

for k in range(0,51):
        g = quad(double_cosine_cosine, 0, 2*(math.pi), args = (k))[0]
        print('%d : %f'% (k, g))

print('Below are double_cosine_sine coefficient values')

for l in range(0,51):
        h = quad(double_cosine_cosine, 0, 2*(math.pi), args = (l))[0]
        print('%d : %f'% (l, h))
```

Following is the list of coefficients I obtained in my program for the two different fucntions.

```
Below are exponential cosine coefficient values:

0 : 534.491656
1 : 267.245828
2 : 106.898331
3 : 53.449166
4 : 31.440686
5 : 20.557371
6 : 14.445720
7 : 10.689833
8 : 8.222949
9 : 6.518191
10 : 5.291997
11 : 4.381079
12 : 3.686149
13 : 3.144069
14 : 2.713156
```

```
15 : 2.365007
16 : 2.079734
17 : 1.843075
18 : 1.644590
19 : 1.476496
20 : 1.332897
21 : 1.209257
22 : 1.102045
23 : 1.008475
24 : 0.926329
25 : 0.853821
26 : 0.789500
27 : 0.732180
28 : 0.680881
29 : 0.634788
30 : 0.593220
31 : 0.555605
32 : 0.521455
33 : 0.490359
34 : 0.461963
35 : 0.435964
36 : 0.412098
37 : 0.390140
38 : 0.369890
39 : 0.351177
40 : 0.333849
41 : 0.317771
42 : 0.302828
43 : 0.288914
44 : 0.275938
45 : 0.263816
46 : 0.252476
47 : 0.241851
48 : 0.231884
49 : 0.222519
50 : 0.213711

Below are exponential sine coefficient values:

0 : 0.000000
1 : −267.245828
2 : −213.796662
3 : −160.347497
4 : −125.762742
5 : −102.786857
6 : −86.674323
```

```
7 : −74.828832
8 : −65.783588
9 : −58.663718
10 : −52.919966
11 : −48.191871
12 : −44.233792
13 : −40.872891
14 : −37.984179
15 : −35.475110
16 : −33.275745
17 : −31.332269
18 : −29.602615
19 : −28.053429
20 : −26.657938
21 : −25.394400
22 : −24.244982
23 : −23.194921
24 : −22.231889
25 : −21.345513
26 : −20.527006
27 : −19.768869
28 : −19.064671
29 : −18.408857
30 : −17.796615
31 : −17.223744
32 : −16.686569
33 : −16.181857
34 : −15.706756
35 : −15.258734
36 : −14.835543
37 : −14.435176
38 : −14.055836
39 : −13.695910
40 : −13.353945
41 : −13.028631
42 : −12.718782
43 : −12.423320
44 : −12.141266
45 : −11.871730
46 : −11.613895
47 : −11.367017
48 : −11.130412
49 : −10.903452
50 : −10.685559

Below are double_cosine cosine coefficient values:
```

```
0  :  4.807879
1  :  −0.000000
2  :  −0.721960
3  :  −0.000000
4  :  0.015561
5  :  −0.000000
6  :  −0.000132
7  :  −0.000000
8  :  0.000001
9  :  0.000000
10  :  −0.000000
11  :  −0.000000
12  :  0.000000
13  :  −0.000000
14  :  −0.000000
15  :  −0.000000
16  :  0.000000
17  :  −0.000000
18  :  0.000000
19  :  0.000000
20  :  0.000000
21  :  0.000000
22  :  0.000000
23  :  0.000000
24  :  −0.000000
25  :  −0.000000
26  :  0.000000
27  :  0.000000
28  :  −0.000000
29  :  −0.000000
30  :  0.000000
31  :  0.000000
32  :  0.000000
33  :  0.000000
34  :  −0.000000
35  :  −0.000000
36  :  −0.000000
37  :  −0.000000
38  :  0.000000
39  :  −0.000000
40  :  −0.000000
41  :  0.000000
42  :  −0.000000
43  :  −0.000000
44  :  0.000000
```

```
45 : −0.000000
46 : −0.000000
47 : 0.000000
48 : −0.000000
49 : 0.000000
50 : −0.000000

Below are double_cosine sine coefficient values:

0 : 4.807879
1 : −0.000000
2 : −0.721960
3 : −0.000000
4 : 0.015561
5 : −0.000000
6 : −0.000132
7 : −0.000000
8 : 0.000001
9 : 0.000000
10 : −0.000000
11 : −0.000000
12 : 0.000000
13 : −0.000000
14 : −0.000000
15 : −0.000000
16 : 0.000000
17 : −0.000000
18 : 0.000000
19 : 0.000000
20 : 0.000000
21 : 0.000000
22 : 0.000000
23 : 0.000000
24 : −0.000000
25 : −0.000000
26 : 0.000000
27 : 0.000000
28 : −0.000000
29 : −0.000000
30 : 0.000000
31 : 0.000000
32 : 0.000000
33 : 0.000000
34 : −0.000000
35 : −0.000000
36 : −0.000000
```

```
37 : -0.000000
38 : 0.000000
39 : -0.000000
40 : -0.000000
41 : 0.000000
42 : -0.000000
43 : -0.000000
44 : 0.000000
45 : -0.000000
46 : -0.000000
47 : 0.000000
48 : -0.000000
49 : 0.000000
50 : -0.000000
```

2.3 Coefficients plotting for the two functions in loglog and semilog plots:

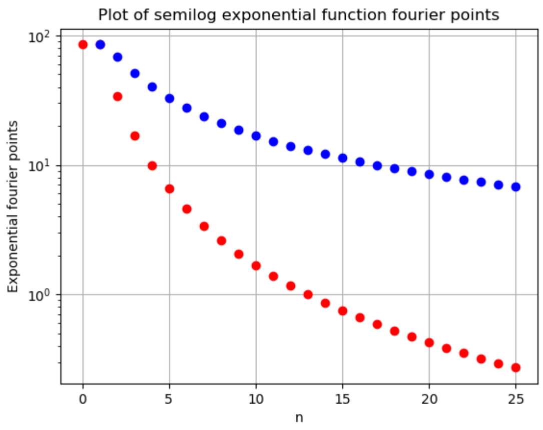
In this question we were supposed to plot the coefficients of the two functions vs 'n' in loglog and in semilog plots.

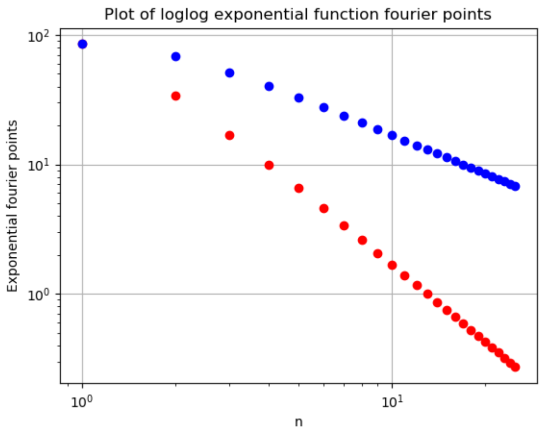
Below is the way I have plotted exponential fourier coefficients vs n.

```
#Plotting functions

a = []
b = []
value1 = (1/(2*math.pi))*quad(exp_cosine, 0, 2*(math.pi), args = (0))[0]
a.append(value1)
b.append(0)

for i in range(1,26):
        a.append((1/(math.pi))*quad(exp_cosine, 0, 2*(math.pi), args=(i))[0])

for j in range(1,26):
        b.append((1/(math.pi))*quad(exp_sine, 0, 2*(math.pi), args=(j))[0])

a = list(map(abs,a))
b = list(map(abs,b))

plt.figure(3)
plt.loglog(n, a, 'ro')
plt.loglog(n, b, 'bo')
plt.xlabel('n')
plt.ylabel('Exponential fourier points')
plt.title('Plot of loglog exponential function fourier points')
plt.grid(True)
plt.legend()

plt.figure(4)
plt.semilogy(n, a, 'ro')
plt.semilogy(n, b, 'bo')
plt.xlabel('n')
plt.ylabel('Exponential fourier points')
plt.title('Plot of semilog exponential function fourier points')
plt.grid(True)
plt.legend()
```

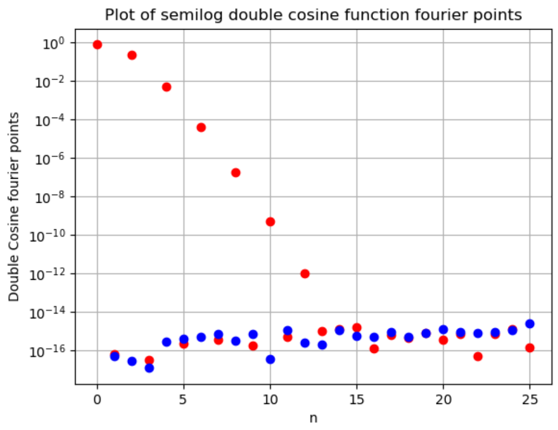
Below is the way I have plotted double cosine fourier coefficients vs n.

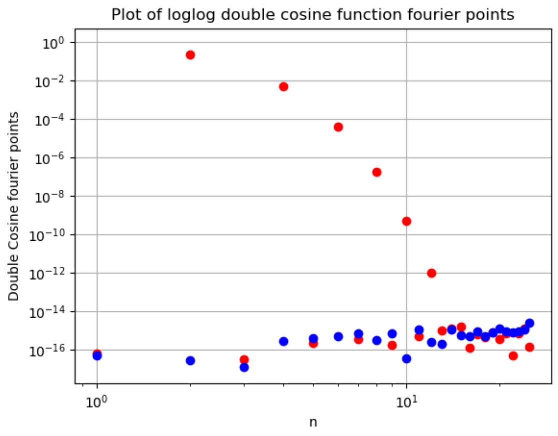
```
A = []
B = []

value2 = (1/(2*math.pi))*quad(double_cosine_cosine, 0,
2*(math.pi), args = (0))[0]
A.append(value2)
B.append(0)

for k in range(1,26):
        A.append((1/(math.pi))*quad(double_cosine_cosine, 0,
2*(math.pi), args = (k))[0])

for l in range(1,26):
        B.append((1/(math.pi))*quad(double_cosine_sine, 0,
2*(math.pi), args = (l))[0])

A = list(map(abs,A))
B = list(map(abs,B))

plt.figure(5)
plt.loglog(n, A, 'ro')
plt.loglog(n, B, 'bo')
plt.xlabel('n')
plt.ylabel('Double Cosine fourier points')
plt.title('Plot of loglog double cosine function fourier points')
plt.grid(True)
plt.legend()

plt.figure(6)
plt.semilogy(n, A, 'ro')
plt.semilogy(n, B, 'bo')
plt.xlabel('n')
plt.ylabel('Double Cosine fourier points')
plt.title('Plot of semilog double cosine function fourier points')
plt.grid(True)
plt.legend()
plt.show()
```





Above is the graph of exponential plot in semilog and loglog plots. Red dot show a$_n$ values and blue dots show b$_n$ values.





Above is the graph of exponential plot in semilog and loglog plots. Red dot show a$_n$ values and blue dots show b$_n$ values.

- The b_n coefficients in the second case are almost zero as the second function is even and the integral involved in calculation of b_n is zero for even function.
- Second function is periodic and thus the high order (high frequency) coefficients are almost zero, whereas first function is non periodic and has some non-zero high order coefficients. Thus in the first case, the coefficients do not decay as quickly as the coefficients for the second case.
- The Fourier coefficients of the first function decay as order $\frac{1}{n^2}$ and so the loglog plot is linear, whereas the Fourier coefficients of second function decay exponentially and so the semilog plot is linear.

2.4 Fourier coefficients using Least Squares Approach:

In this question, we were supposed to find the fourier coefficient with matrices using least squares appraoch. For each x_i, we calculate,

$$a_0 + \sum_1^{25} a_n cos(nx_i) + \sum_1^{25} b_n sin(nx_i) \approx f(x_i)$$

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

We solve the matrix equation,
Ac = b
Following is the way to create a matrix A.

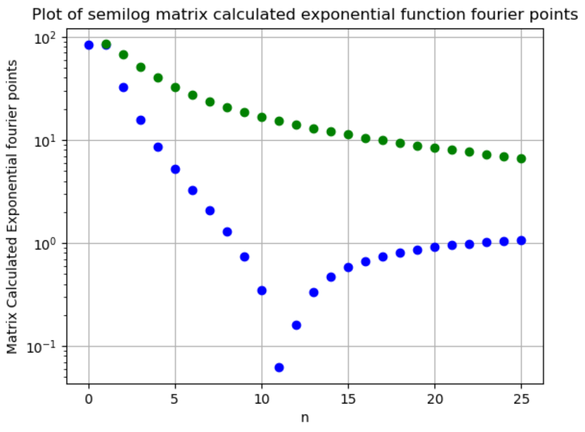
```
# Creating matrix A

x = linspace(0, 2*(math.pi), 401)
x = x[:-1]

b_exp = exp(x)
#order of defining a function is important

b_double_cosine = double_cosine(x)
# drop last term to have a proper periodic integral

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] = cos(k*x)
    # cos(kx) column
        A[:,2*k] = sin(k*x)
    # sin(kx) column

c_exp = lstsq(A,b_exp)[0]
# the [0] is to pull out the best fit vector. lstsq returns a list.
c_double_cosine = lstsq(A,b_double_cosine)[0]
```

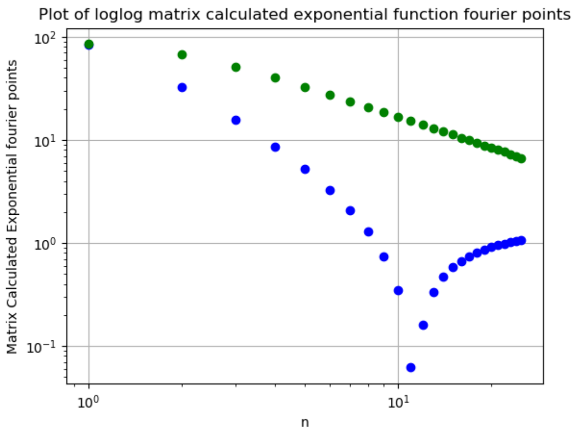
2.5 Plotting the coefficients calculated using least squares:

We execute,

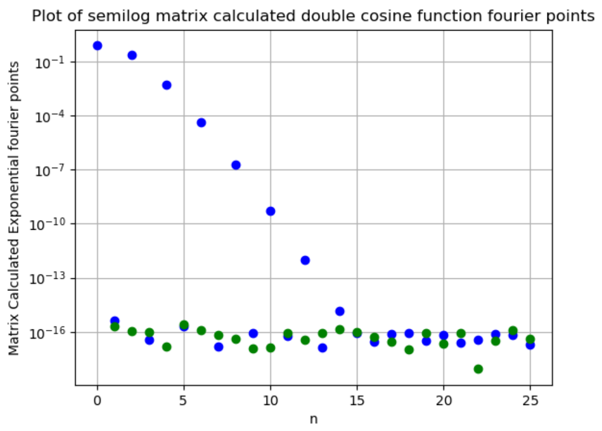
c=lstsq(A,b)[0]

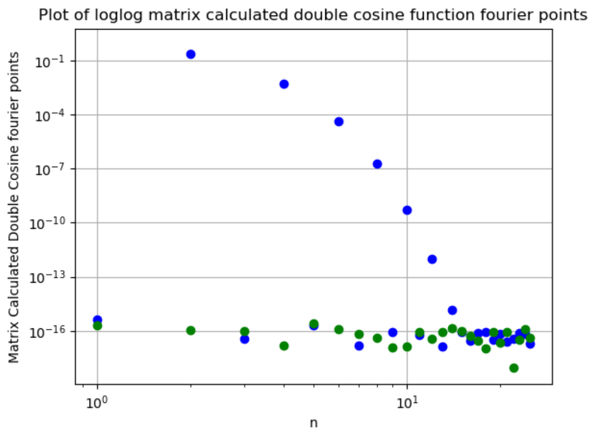
What this does is that it finds the “best fit” numbers that will satisfy Ac = b and give us the vlaues of c in a list.





Above two are the plots for exponential functions’ fourier coefficients calculated using least squares. Blue dot show a$_n$ values and green dots show b$_n$ values.





Above two are the plots for the double cosine functions' fourier coefficients. Blue dot show a$_n$ values and green dots show b$_n$ values.

2.6 Deviation of 'quad' calculated coefficients from the matrix calculated (using least squares) fourier coefficients:

In this question we were supposed to find the deviation of the outputs of the two methods. There is a certain deviation in the answers of these methods.

```
# Deviation calculation

print('a-terms deviation in exponential')
print(max(map(abs,list(set(a)-set(c_exp_a)))))
print('b-terms deviation in exponential')
print(max(map(abs,list(set(b)-set(c_exp_b)))))
print('a–terms deviation in double_cosine')
print(max(map(abs, list(set(A)–set(c_double_cosine_a)))))
print('b–terms deviation in double_cosine')
print(max(map(abs, list(set(B)–set(c_double_cosine_b)))))
```

This is the way we calculate the absolute value of the deviation.

```
a-terms deviation in exponential function,
85.0669890181
b-terms deviation in exponential function,
85.0669890181
a-terms deviation in double_cosine function,
0.765197686558
b-terms deviation in double_cosine function,
2.59494909399e-15
```

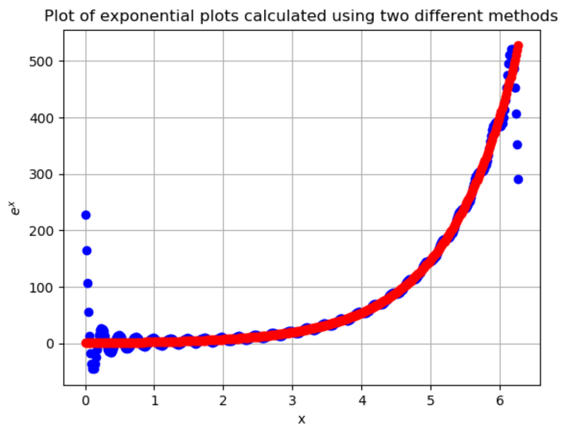
The coefficients are almost equal for $\cos(\cos(x))$, highlighting its periodic nature. However, there is some deviation in the high order coefficients obtained for e^x which can be explained due to its non-periodic nature.

2.7 Plot comparison:

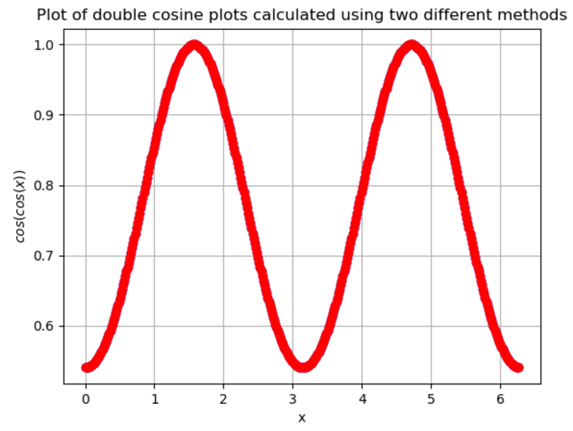
In this question, we were supposed to calculate the value of the functions using A and c matrices and then compare with the initial plots.

Below is the code to plot and compare th values of the two functions.

```
#Comparing plots

exp_mult = numpy.matmul(A, c_exp)
double_cosine_mult = numpy. matmul(A, c_double_cosine)

plt.figure(1)
plt.plot(x, exp_mult, 'bo')
plt.plot(x, b_exp, 'ro')

plt.figure(2)
plt.plot(x, double_cosine_mult, 'bo')
plt.plot(x, b_double_cosine, 'ro')
plt.show()
```



As we see from the above plot, there is some deviation in this plot. Red dots indicate the 'quad' calculated coefficients and blue dots indicate those coefficients calculated using 'least squares'.



As we see from the above plot, there is no deviation in this plot. The plots overlap each other perfectly well. Red dots indicate the 'quad' calculated coefficients and blue dots indicate those coefficients calculated using 'least squares'.

While reconstructing the functions, we get almost an exact match for $\cos(\cos(x))$ even when we consider only the first 51 coefficients. This can be supported by the fact that the coefficients decay off very quickly for this function, pointing that the function has very low contribution from the the high harmonics and thus it can be reconstructed almost identically using first few Fourier coefficients.

However, this is not the case for e^x. Due to its non-periodic nature, the Fourier coefficients do not decay rapidly signifying substancial contributions from the higher harmonics. But since we take only few harmonics into account we get to see a deviation in values from the actual function.