EE2703: Applied programming Lab Assignment 4

GUGULOTHU NAVEEN

EE20B039

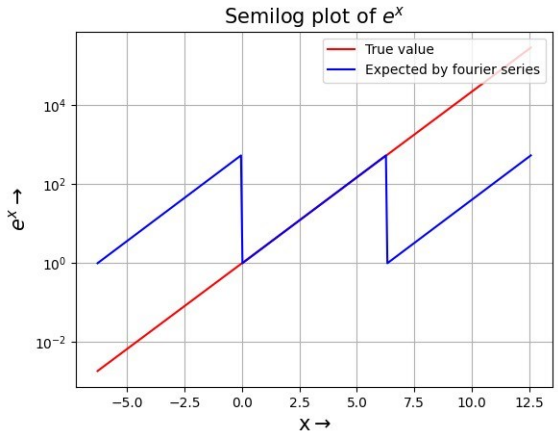
1 Assignment

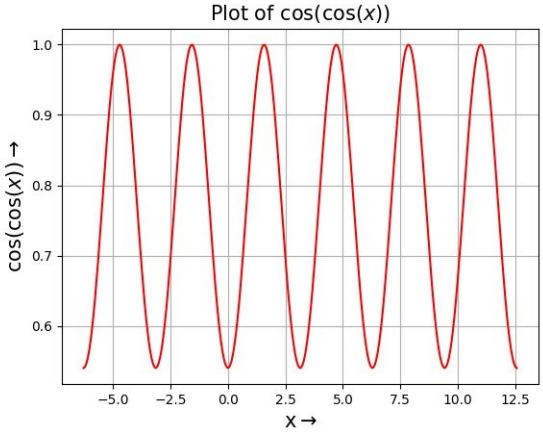
Q1. Defining and Plotting the Functions

We need to ensure that the function can output correct values even for an Ndimensional vector as an input. The built-in functions in numpy can be used to accomplish the same :

```
import numpy as np def exp(x ):
return np. exp(x)
def coscos (x)          :
            return np. cos (np. cos (x))
```

The above functions are plotted over the interval [-2π, 4π) using 300 points sampled uniformly over this interval. The following plots are obtained :





Q2. Finding the Fourier series coefficients: Integration approach

The Fourier series coefficients are obtained using the quad function in the scipy library. The following code snippet evaluates the first n coefficients of the Fourier series expansion of e^x and $cos(cos(x))$:

```
func diet = { 'exp(x) ' : exp , ' cos ( cos (x)) ' :                coscos}
def find coeff (n, label ): coeff = np. zeros
    (n) func = func _dict [ label ]
    u = lambda x , k : func (x)*np. cos (k*x) v = lambda x , k : func
    (x)*np. sin (k*x) coeff [0]= quad( func ,0 ,2*np. pi )[0]/(2*np. pi
    )
    for i in range(1 ,n ,2): coeff [ i ] = quad(u,0 ,2*np. pi , args=(( i +1)/2))[0]/np. pi
    for i in range(2 ,n ,2):
        coeff [ i ] = quad(v,0 ,2*np. pi , args=(i /2))[0]/np. pi
    return coeff
```

Q3. Semilog and Loglog plot for both functions

We thus obtain the first 51 coefficient of the two functions. They are saved in the following form:

a_0

a_1

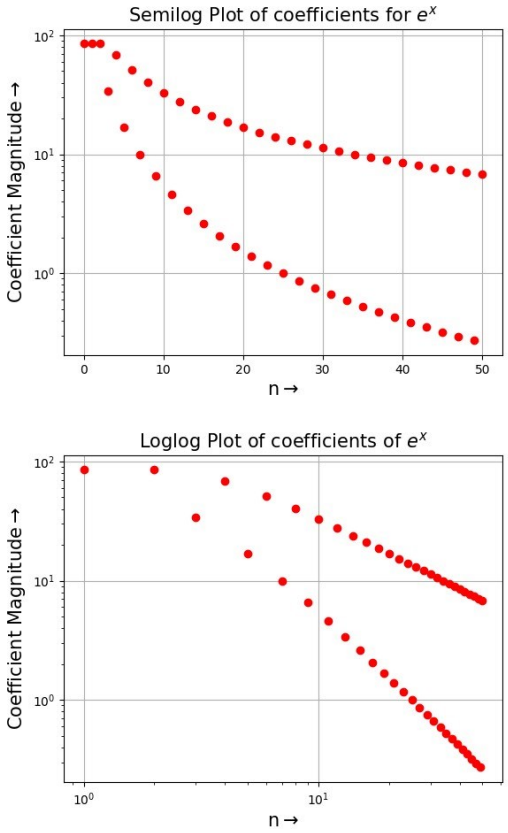
b_1

...

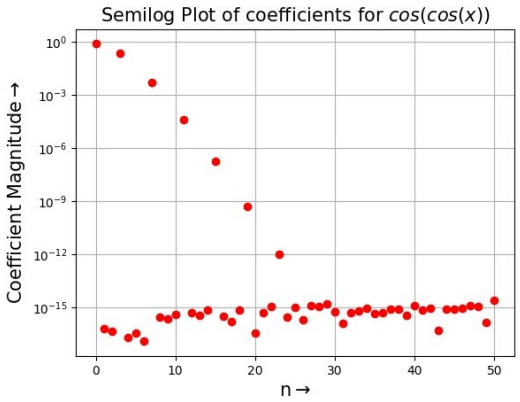
a_{25}

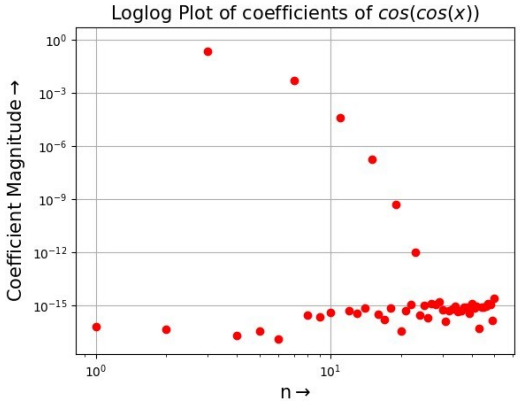
b_{25}

Plot of the fourier coefficients of e^x in semilog and loglog scale are:



Similarly, the plots for cos(cos(x)) are as follows:





We notice that the b_n coefficients in the second case are expected to be zero as the function cos(cos(x)) is even and thus the odd sinusoidal components are zero. However, the values obtained are non-zero because of the limitation in the numerical accuracy upto which π can be stored in memory.

In the first case, the function, having an exponentially increasing gradient, contains a wide range of frequencies in its fourier series. In the second case, cos(cos(x)) has a relatively low frequency of $\frac{1}{\pi}$ and thus contribution by the higher sinusoids is less, manifesting in the quick decay of the coefficients with n.

$$|a_n|, |b_n| \propto \frac{1}{n^2+1} \quad \text{(1) Thus for large n,}$$

$$log(|a_n|), log(|b_n|) \propto log(\frac{1}{n^2+1}) \approx -2log(n) \quad \text{(2)}$$

Thus, the *loglog* plot becomes approximately linear as n increases. Similarly, the coefficients of cos(cos(x)) would be approximately exponential with n . Thus the *semilog* plot is linear.

Q4. Finding the Fourier series coefficients: "Least Squares approach"

Building on last weeks work we now try a Least Squares Approach to this problem. We linearly choose 400 values of x in the range $[0,2\pi)$. It can be Noted that far better approximations were achieved when a larger value($\sim$10000) was used instead of 400. We try to find the Fourier coefficients using linear regression on these 400 values. The matrix equation is

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

We create the matrix on the left side and call it *A*. So basically we want to solve

$$Ac = b$$

where *c* are the fourier coefficients.
Code for implementing the above equation is:

```
x = np. linspace (0 ,2*np. pi ,401)
x = x[: =1]
A = np. zeros ((400 ,51)) A[ : , 0 ]
= 1 for i in range(1 ,26):
        A[: ,2* i =1] = np. cos ( i *x)
        A[:,2    * i]=np.sin(i          *x)

b -exp=exp(x)
b -coscos=coscos(x)

c -exp=np.linalg.lstsq(A,b                -exp,     rcond=None)[0]
c -coscos=np.linalg.lstsq(A,b                    -coscos,rcond=None)[0]
```

Comparing Runtime of Integration and Least Square approaches

The least squares estimation method provides a faster way of evaluating a large number of Fourier coefficients. In this case of evaluating 51 coefficients, the time taken to evaluate the coefficients in each method are as follows:

```
"""runtime : integration approach""" start = timeit . default timer () coeff exp =
find coeff (51 , 'exp(x) ' ) elapsed = timeit . default timer () = start print(
'Runtime with integration approach = ' , elapsed )

"""runtime : least square approach""" start = timeit .
default timer () for i in range(1 ,26):
        A[: ,2* i =1] = np. cos ( i *x)
        A[: ,2* i ] = np. sin ( i *x) b exp =
exp(x)
c exp = np. linalg . lstsq (A, b exp , rcond=None )[0] elapsed = timeit . default timer
() = start print( 'Runtime with least square approach = ' , elapsed )
```

Code Output:

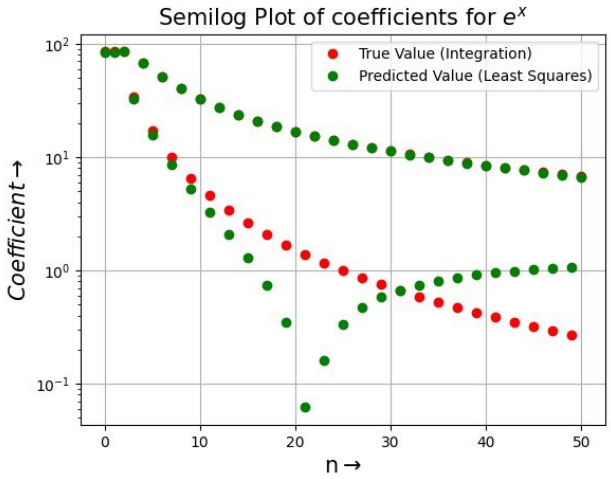
Runtime with integration approach = 0.16507640000000023 Runtime with least square approach = 0.003682799999999986

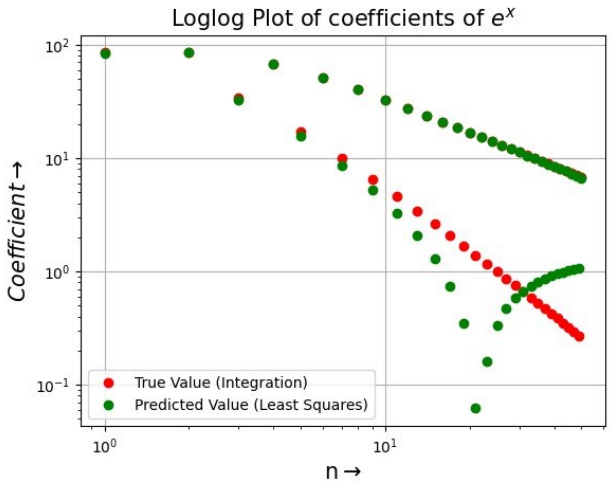
Conclusion:

Clearly in the direct integration approach the runtime is almost 50 times the taken for the least squares approach.

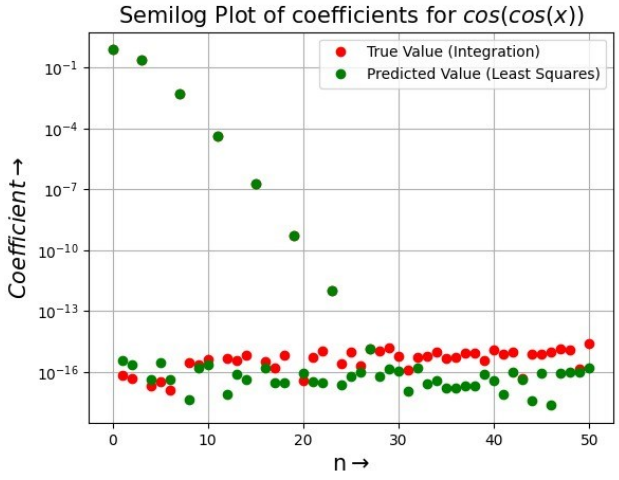
Q5. Plots comparing the coefficients by "Integration" and "Least Square" approaches

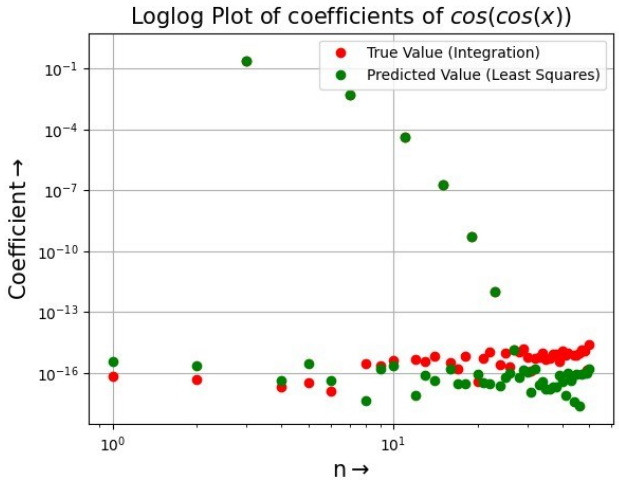
Plot of the fourier coefficients of e^x in semilog and loglog scale are:





Similarly, the plots for cos(cos(x)) are as follows:





It can be seen that the coefficients for cos(cos(x)) are in much closer agreement than the coefficients for e^x and is also expected as the former is periodic and the periodic extension of the latter has an increasing gradient and would require higher sinusoids for a better representation. Since the least squares approach is only approximate, the deviation is expected.

Q6. Calculating the deviation b/w Least square and direct integration coefficients

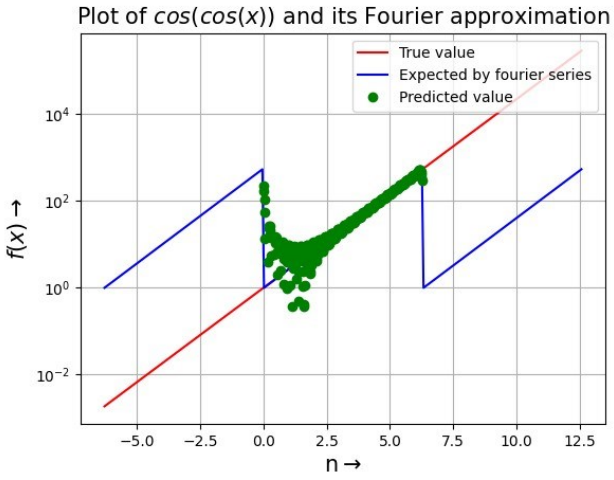
```
deviation -exp= abs ( coeff -exp — c-exp)
deviation -coscos= abs ( coeff -coscos — c-coscos)

max -dev -exp=np. max( deviation -exp)
max -dev -coscos=np. max( deviation -coscos) \\
```

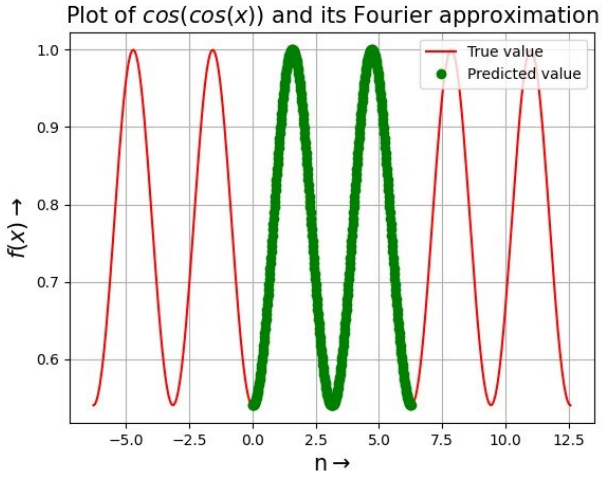
where, coeff exp and coeff coscos are the values obtained through integration and c exp and c coscos are the values obtained through the least squares approach.

The maximum deviation in the coefficient of e^x =1.332730870335439 whereas the same metric for cos(cos(x)) = 2.674677035413032e-15 Our Predictions for e^x are very poor compared to that of cos(cos(x)). This can be fixed by sampling at a larger number of points.

Q7. Plots comparing the function values obtained by the "least squares" method with the "true" value

The matrix product Ac gives an estimate of the function values at $x_1,x_2,....x_{400}$ chosen earlier. These estimates, along with the actual value of the functions yield the following plots:





It should be noted that e^x is a non periodic function and Fourier series exists only for periodic functions. Hence we have considered a variation of e^x with period 2π that has the actual value of e^x only in the range $[0,2\pi)$. Hence it is acceptable that there is a large discrepancy in the predicted value of e^x at these boundaries.

3 Conclusion

The Fourier series coefficients of e^x and cos(cos(x)) were found in two ways: direct integration and estimation using the least squares approach. It was verified that the odd sinusoidal components of cos(cos(x)) are nearly zero. The least squares approach provided a more computationally inexpensive way of estimating the coefficients. It was found that the least squares estimate deviated more for e^x than cos(cos(x)), due to the high gradient characteristics of the former and periodic behaviour of the latter.