Assignment 4: Applied Programming Lab

Saurabh Vaishampayan, EP17B028

25 February 2020

1 Abstract

To implement algorithm for determining Fourier coefficients by integration method, and approximate determination by least squares fitting. We observe the effects of these on two functions: e^x and cos(cos(x)), plot coefficients and reconstructed graphs, discuss errors and validity

2 Question 1

2.1 Description

We define functions that have inputs and outputs as vectors. Since we are given the description of the signal in one period, the complete signal is obtained by shifting the earlier signal. In the context of vectors this is obtained by modulo operation, where numerator is array index and denominator is number of points in the fundamental period 5mm

2.2 Codes

```python
#Question 1
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate
from scipy.linalg import lstsq

def exp_array(x):
    return np.exp(x)

def coscos_array(x):
    return np.cos(np.cos(x))

x_2pi = np.linspace(0,2*np.pi,101)
x_2pi = x_2pi[:-1]

y1_2pi = exp_array(x_2pi)
y2_2pi = coscos_array(x_2pi)

x = np.linspace(-2*np.pi,4*np.pi,301)
```

```python
x = x[:-1]

y1 = np.zeros(len(x))    #periodic exp(x)
y2 = np.zeros(len(x))    #periodic cos(cos(x))

for i in range(len(x)):
    y1[i] = y1_2pi[i%100]
    y2[i] = y2_2pi[i%100]

#Plot exp in semilogy scale and cos(cos(x)) in linear scale
plt.plot(x,y1)
plt.yscale('log')
plt.ylabel(r'$e^{x}$',size=20)
plt.xlabel('x',size=20)
plt.title(r'$2\pi$'+' Periodic '+r'$e^{x}$',size=20)
plt.grid(True)
plt.savefig('Figure 1.png')
plt.show()

plt.plot(x,y2)
plt.title(r'$2\pi$'+' Periodic '+r'$cos(cos(x))$',size=20)
plt.ylabel(r'$cos(cos(x))$',size=20)
plt.xlabel('x',size=20)
plt.grid(True)
plt.savefig('Figure 2.png')
plt.show()
```

2.3 Plots

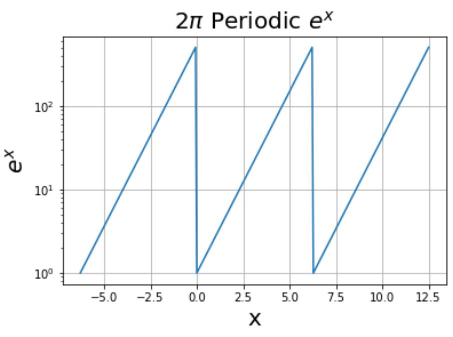


Figure 1: e^x in semilogy scale from $[-2\pi, 4\pi)$

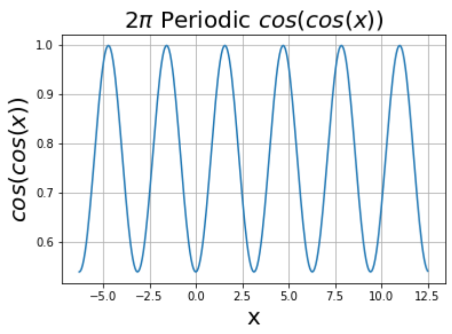


Figure 2: cos(cos(x)) in Linear scale from $[-2\pi, 4\pi)$

5mm

3 Question 2

3.1 Description

Coefficients are obtained by integration of the function with the appropriate weight(cos(nx) or sin(nx)). We have defined a function that takes in the required function and number of desired fourier coefficients as input and returns the array of coefficients in the format asked. This array can be parsed to yield sine and cosine coefficient arrays by another function

3.2 Codes

```
def u(x,f,k):
    y = f(x)*np.cos(k*x)
    return y

def v(x,f,k):
    y = f(x)*np.sin(k*x)
    return y

#This function takes in as input a function, number of desired coefficients
#and returns the array of fourier coefficients of sin, cosine and both
#appended in the format asked

def coeff(f,Ncoeff):
    coeffarray = np.zeros(2*Ncoeff+1)   #complete array of coefficients
    for k in range(1,len(coeffarray)):
        if(k%2==1):
            coeffarray[k] = (1/np.pi)*scipy.integrate.quad(u,0,2*np.pi,args
    =(f,k//2+1))[0]
        else:
            coeffarray[k] = (1/np.pi)*scipy.integrate.quad(v,0,2*np.pi,args
    =(f,k//2))[0]
            coeffarray[0] = (1/(2*np.pi))*scipy.integrate.quad(f,0,2*np.pi)
    [0]

    return coeffarray

def coeffparse(w):
    a = np.zeros(int((len(w)+1)/2))
    b = np.zeros(len(a)-1)
    a[0] = w[0]
    for i in range(1,len(w)):
        if(i%2==1):
            a[i//2+1] = w[i]
        else:
            b[i//2-1] = w[i]
    return a,b

expcoeff = coeff(exp_array,25)
aexp,bexp = coeffparse(expcoeff)
```

```

coscoscoeff = coeff(coscos_array,25)
acoscos,bcoscos = coeffparse(coscoscoeff)
```

4 Question 3

4.1 Description

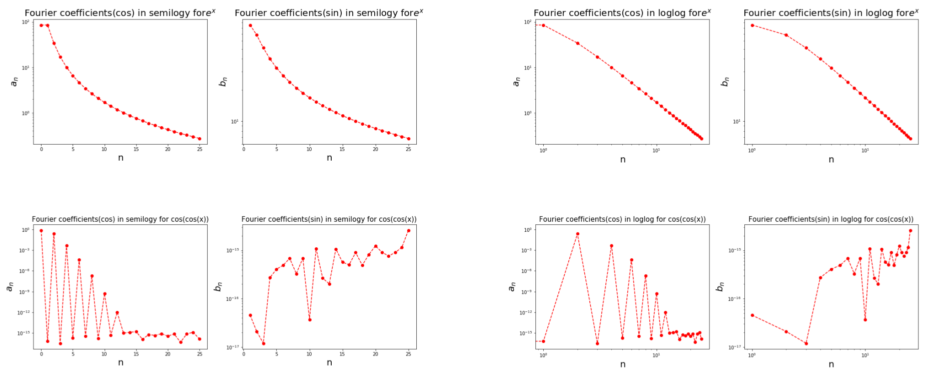
We borrow the array of coefficients output from question 2 and plot in loglog, semilog scales for both the sine and cosine coefficients for both the functions

4.2 Codes

```
#Question 3
x2 = np.arange(1,26,1)
x1 = np.arange(0,26,1)

#Coefficients in semilog scale for exp(x)
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(aexp),'ro')
ax[0].plot(x1,np.abs(aexp),'r--')
ax[0].set_title('Fourier coefficients(cos) in semilogy for'+r'$e^{x}$',size
    =20)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')

ax[1].plot(x2,np.abs(bexp),'ro')
ax[1].plot(x2,np.abs(bexp),'r--')
ax[1].set_title('Fourier coefficients(sin) in semilogy for'+r'$e^{x}$',size
    =20)
ax[1].set_ylabel(r'$b_{n}$',size=20)
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
plt.savefig('Figure 3.png')
plt.show()

#Coefficients in loglog scale for exp(x)
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(aexp),'ro')
ax[0].plot(x1,np.abs(aexp),'r--')
ax[0].set_title('Fourier coefficients(cos) in loglog for'+r'$e^{x}$',size
    =20)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')
ax[0].set_xscale('log')

ax[1].plot(x2,np.abs(bexp),'ro')
ax[1].plot(x2,np.abs(bexp),'r--')
ax[1].set_title('Fourier coefficients(sin) in loglog for'+r'$e^{x}$',size
    =20)
ax[1].set_ylabel(r'$b_{n}$',size=20)
```

```
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
ax[1].set_xscale('log')
plt.savefig('Figure 4.png')
plt.show()

#Coefficients in semilog scale for cos(cos(x))
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(acoscos),'ro')
ax[0].plot(x1,np.abs(acoscos),'r--')
ax[0].set_title('Fourier coefficients(cos) in semilogy for cos(cos(x))',
    size=15)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')

ax[1].plot(x2,np.abs(bcoscos),'ro')
ax[1].plot(x2,np.abs(bcoscos),'r--')
ax[1].set_title('Fourier coefficients(sin) in semilogy for cos(cos(x))',
    size=15)
ax[1].set_ylabel(r'$b_{n}$',size=20)
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
plt.savefig('Figure 5.png')
plt.show()

#Coefficients in loglog scale for cos(cos(x))
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(acoscos),'ro')
ax[0].plot(x1,np.abs(acoscos),'r--')
ax[0].set_title('Fourier coefficients(cos) in loglog for cos(cos(x))',size
    =15)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')
ax[0].set_xscale('log')

ax[1].plot(x2,np.abs(bcoscos),'ro')
ax[1].plot(x2,np.abs(bcoscos),'r--')
ax[1].set_title('Fourier coefficients(sin) in loglog for cos(cos(x))',size
    =15)
ax[1].set_ylabel(r'$b_{n}$',size=20)
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
ax[1].set_xscale('log')
plt.savefig('Figure 6.png')
plt.show()
```

4.3 Plots



5 Question 4 and 5

5.1 Description

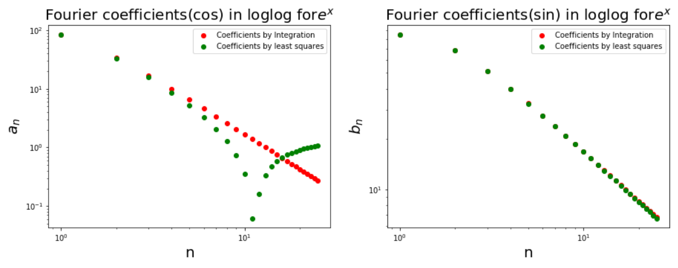
Now we try to obtain fourier coefficients by least squares method. The problem is to find the weights for sine and cosine terms. We construct a matrix with entries as values of the sine and cosine functions at different values of x. Now we construct a column vector whose entries are values of the desired function at different values of x and solve using least squares

5.2 Codes

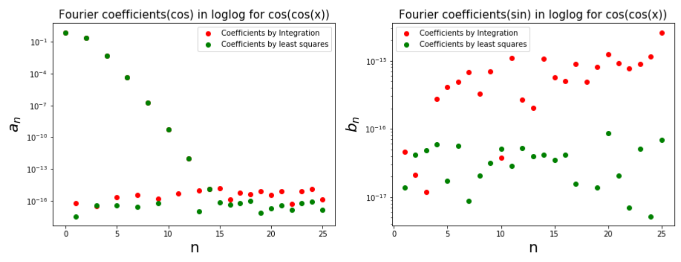
```python
#Question 4 and 5
def fourierlstsq(x,f,Ncoeff):
    A = np.zeros((len(x),2*Ncoeff+1))
    b = f(x)
    A[:,0] = 1
    for cols in range(1,Ncoeff+1):
        A[:,2*cols-1] = np.cos(cols*x)
        A[:,2*cols] = np.sin(cols*x)
    c = lstsq(A,b)[0]
    return c, A

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

c_exp, A_exp = fourierlstsq(x,exp_array,25)
a_cexp,b_cexp = coeffparse(c_exp)

c_coscos, A_coscos = fourierlstsq(x,coscos_array,25)
a_ccoscos,b_ccoscos = coeffparse(c_coscos)
```

```

#Coefficients in loglog scale for exp(x): Integration vs lstsq
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(aexp),'ro',label='Coefficients by Integration')
ax[0].plot(x1,np.abs(a_cexp),'go',label='Coefficients by least squares')
ax[0].set_title('Fourier coefficients(cos) in loglog for'+r'$e^{x}$',size
    =20)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')
ax[0].set_xscale('log')
ax[0].legend()

ax[1].plot(x2,np.abs(bexp),'ro',label='Coefficients by Integration')
ax[1].plot(x2,np.abs(b_cexp),'go',label='Coefficients by least squares')
ax[1].set_title('Fourier coefficients(sin) in loglog for'+r'$e^{x}$',size
    =20)
ax[1].set_ylabel(r'$b_{n}$',size=20)
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
ax[1].set_xscale('log')
ax[1].legend()
plt.savefig('Figure 7.png')
plt.show()

#Coefficients in semilogy scale for cos(cos(x)): Integration vs lstsq
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].plot(x1,np.abs(acoscos),'ro',label='Coefficients by Integration')
ax[0].plot(x1,np.abs(a_ccoscos),'go',label='Coefficients by least squares')
ax[0].set_title('Fourier coefficients(cos) in loglog for cos(cos(x))',size
    =15)
ax[0].set_ylabel(r'$a_{n}$',size=20)
ax[0].set_xlabel('n',size=20)
ax[0].set_yscale('log')
ax[0].legend()

ax[1].plot(x2,np.abs(bcoscos),'ro',label='Coefficients by Integration')
ax[1].plot(x2,np.abs(b_ccoscos),'go',label='Coefficients by least squares')
ax[1].set_title('Fourier coefficients(sin) in loglog for cos(cos(x))',size
    =15)
ax[1].set_ylabel(r'$b_{n}$',size=20)
ax[1].set_xlabel('n',size=20)
ax[1].set_yscale('log')
ax[1].legend()
plt.savefig('Figure 8.png')
plt.show()
```

5.3 Plots



Fourier coefficients comparison for: e^x in loglog scale



Fourier coefficients comparison for: cos(cos(x)) in semilogy scale

6 Question 6

6.1 Description

Comparison of coefficients, finding maximum deviation for each of the two functions

6.2 Codes

```
#Question 6
expdiff = np.abs(expcoeff-c_exp)
coscosdiff = np.abs(coscoscoeff-c_coscos)
maxdevexp = max(expdiff)
maxdevcoscos = max(coscosdiff)
print('Maximum deviation in exponential coefficients array is ',maxdevexp)
print('Maximum deviation in cos(cos(x)) coefficients array is ',
    maxdevcoscos)
```

6.3 Output

Maximum deviation in exponential coefficients array is 1.332730870335439
Maximum deviation in cos(cos(x)) coefficients array is 2.674677035413032e-15

7 Question 7

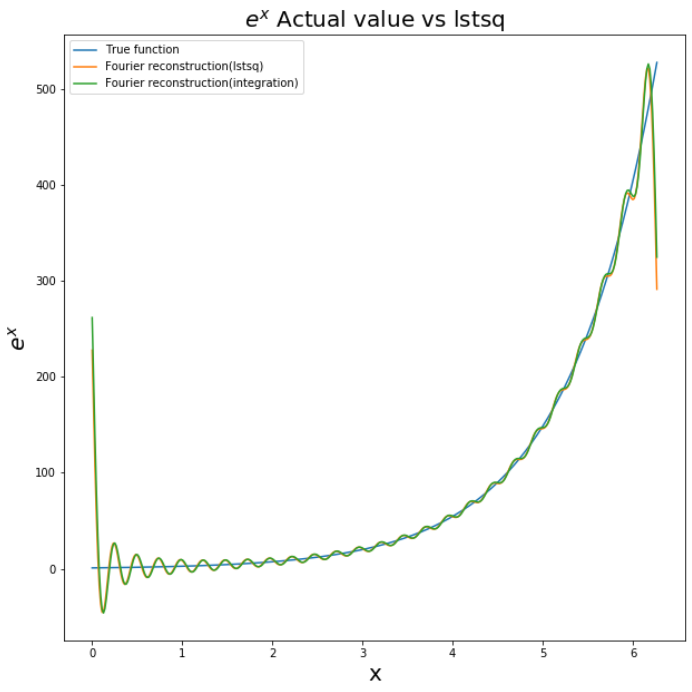
7.1 Description

Plotting the original function alongside the reconstructed signals using Fourier series obtained by the two methods of integration and least squares

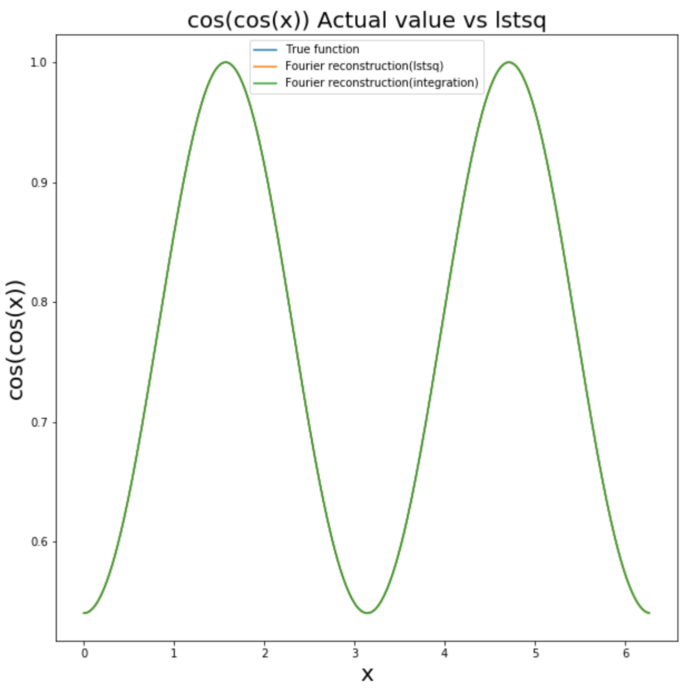
7.2 Codes

```
plt.figure(figsize=(10,10))
plt.plot(x,exp_array(x),label='True function')
plt.plot(x,np.dot(A_exp,c_exp),label='Fourier reconstruction(lstsq)')
plt.plot(x,np.dot(A_exp,expcoeff),label='Fourier reconstruction(integration
    )')
plt.title(r'$e^{x}$'+' Actual value vs lstsq',size=20)
plt.ylabel(r'$e^{x}$',size=20)
plt.xlabel('x',size=20)
plt.legend()
plt.savefig('Figure 9.png')
plt.show()

plt.figure(figsize=(10,10))
plt.plot(x,coscos_array(x),label='True function')
plt.plot(x,np.dot(A_coscos,c_coscos),label='Fourier reconstruction(lstsq)')
plt.plot(x,np.dot(A_coscos,coscoscoeff),label='Fourier reconstruction(
    integration)')
plt.title('cos(cos(x))'+' Actual value vs lstsq',size=20)
plt.ylabel('cos(cos(x))',size=20)
plt.xlabel('x',size=20)
plt.legend()
plt.savefig('Figure 10.png')
plt.show()
```

7.3 Plots



Comparison of true function and graphs reconstructed from fourier series(integration and least squares) for: e^x



Comparison of true function and graphs reconstructed from fourier series(integration and least squares) for cos(cos(x))

8 Conclusions

1. Linear Least squares works by assuming the given signal is a linear combination of the basis signals with an added **uncorrelated** noise, however the signal minus the sum of first N fourier series isn't an uncorrelated signal(it is the sum of remaining terms of the Fourier series uptil infinity). But if Number of coefficients is large the residue is small and it is a good approximation

2. For signals having discontinuities(in the function or some derivative onwards), higher frequencies are more prominent than signals differentiable everywhere.

3. cos(cos(x)) is differentiable everywhere and hence coefficients fall off rapidly. The exponential falloff is derived from properites of Bessel's functions(fourier coefficients are actually samples of it

4. e^x defined with a period of 2π is discontinuous at edges since value at 2π isn't equal to that at zero. The coefficients are proportional to n^k where k is roughly equal to -2, which can be derived from the form of integral

5. As expected the coefficients show comparatively large deviations for exponential compared to cos(cos(x)), which are almost zero(around 1 in 100 trillion). However even these deviations is negligible compared to the coefficients themselves.

6. The reconstructed graphs agree extremely well for cos(cos(x)). For e^x the graphs agree fairly well away from the points of discontinuity but diverge significantly near the boundaries(signatures of Gibb's phenomenon)

7. In some cases least squares can be used to estimate fourier coefficients quicker than by regular integration