Assignment 4: Fourier Approximations

Manognya Param Koolath

February 26, 2019

The task

We are given two functions f_1 and f_2, whose first 51 fourier coefficients are to be evaluated. We do this using two methods. In the first method, we use the formula to evaluate the coefficients. In the second method, we fit a function minimising the least squares loss. Given,

$$f_1(x) = e^x$$

$$f_2(x) = \cos(\cos x)$$

$f1$ will be referred to as the first function and $f2$ as the second.

Procedure

Any function can be represented in the following form using fourier coefficients:

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

where a_0, a_1, a_2,... and b_1, b_2,... are called the fourier coefficients of $f(x)$.

Method 1

We use the following formulae to evaluate the fourier coefficients:

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

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

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

We use the built in python integrator function `quad` from the scipy library to integrate these functions.

Method 2

This is using a Least Squares approach to the problem. First we define a vector x going from 0 to 2π in 400 steps. The function evaluated at these values is a vector b. This function is to be approximated by Eq. (1). This is turned into a matrix problem:

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

We call the matrix on the left as A. We have to solve for c in the equation:

$$Ac = b$$

We do this by using the python function `lstsq` from the scipy library.

We compare the fourier coefficients obtained by the two methods and plot them in different formats. We also compare the estimated functions using the fourier coefficients obtained by the two methods.

Python code

The code is properly commented. All explanations are commented out normally and statements used for debugging purposes are commented out using

```
##
```

Assignment 4 Code

```
# Assignment 4
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as sp
from scipy.linalg import lstsq
import scipy.integrate as spint

#To calculate exp(x)
def f1(x):
        return np.exp(x)
```

```
#To calculate cos(cos (x))
def f2(x):
        return np.cos(np.cos(x))

#The function to pass to the quad function to calculate
#a values for the first function
def u1(x,k):
        return f1(x)*np.cos(k*x)

#The function to pass to the quad function to calculate
#b values for the first function
def v1(x,k):
        return f1(x)*np.sin(k*x)

#The function to pass to the quad function to calculate
#a values for the second function
def u2(x,k):
        return f2(x)*np.cos(k*x)

#The function to pass to the quad function to calculate
#b values for the second function
def v2(x,k):
        return f2(x)*np.sin(k*x)

x = np.linspace(-2*np.pi,4*np.pi,600)
f1_y = f1(x)
f2_y = f2(x)

#The plot showing the first function in semi log scale
plt.semilogy(x,f1_y)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$exp(x)$',size=20)
plt.title(r'The first function in semi log scale along y axis')
plt.show()

#The plot showing the first function
plt.plot(x,f1_y)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$exp(x)$',size=20)
plt.title(r'The first function')
plt.show()
```

```
#The plot showing the second function
plt.plot(x,f2_y)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$cos(cos(x))$',size=20)
plt.title(r'The second function')
plt.show()

a0_1=(spint.quad(f1,0,2*np.pi)[0])/(2*np.pi)
a0_2=(spint.quad(f2,0,2*np.pi)[0])/(2*np.pi)
## print(a0_1,a0_2)
#ab_1 will store the fourier coefficients obtained
#by integration of the first function.
ab_1=np.full(51,0.0)
#ab_2 will store the fourier coefficients obtained
#by integration of the second function.
ab_2=np.full(51,0.0)
ab_1[0]=a0_1
ab_2[0]=a0_2
## print(ab_2[0])

for i in range (25):
        ab_1[2*i+1] = (spint.quad(u1,0,2*np.pi,args=(i+1))[0])/(np.pi)
        ab_1[2*i+2] = (spint.quad(v1,0,2*np.pi,args=(i+1))[0])/(np.pi)
        ab_2[2*i+1] = (spint.quad(u2,0,2*np.pi,args=(i+1))[0])/(np.pi)
        ab_2[2*i+2] = (spint.quad(v2,0,2*np.pi,args=(i+1))[0])/(np.pi)
## print(ab_1)
#ab_x has the index values to plot the fourier coefficients obtained.
ab_x = np.arange(51)
ab_1 = np.abs(ab_1)
ab_2 = np.abs(ab_2)
## print(ab_2)

# plt.semilogy(ab_x,ab_1,'ro')
# plt.grid(True)
# plt.ylabel(r'Fourier coefficients',size=20)
# plt.title(r'Fourier coefficients of the first function')
# plt.show()

# plt.loglog(ab_x,ab_1,'ro')
# plt.grid(True)
# plt.ylabel(r'Fourier coefficients',size=20)
# plt.title(r'Fourier coefficients of the first function')
```

```
# plt.show()

# plt.semilogy(ab_x,ab_2,'ro')
# plt.grid(True)
# plt.ylabel(r'Fourier coefficients',size=20)
# plt.title(r'Fourier coefficients of the second function')
# plt.show()

# plt.loglog(ab_x,ab_2,'ro')
# plt.grid(True)
# plt.ylabel(r'Fourier coefficients',size=20)
# plt.title(r'Fourier coefficients of the second function')
# plt.show()

#Calculating fourier coefficients by least squares fitting method:
x_ls = np.linspace(0,2*np.pi,401)
x_ls = x_ls[:-1]
b_1 = f1(x_ls)
b_2 = f2(x_ls)
A = np.full((400,51),1.0)
for k in range(1,26):
        A[:,2*k-1] = np.cos(k*x_ls)
        A[:,2*k] = np.sin(k*x_ls)
c_1 = lstsq(A,b_1)[0]
c_2 = lstsq(A,b_2)[0]

## print(c_1)
## print(c_2)

#The plots showing fourier coefficients obtained by the two methods done:
plt.semilogy(ab_x,ab_1,'ro')
plt.semilogy(ab_x,np.abs(c_1),'go')
plt.grid(True)
plt.xlabel(r'Index',size=20)
plt.ylabel(r'Fourier coefficients',size=20)
plt.title(r'Fourier coefficients of the first function')
plt.show()

plt.loglog(ab_x,ab_1,'ro')
plt.loglog(ab_x,np.abs(c_1),'go')
plt.grid(True)
plt.xlabel(r'Index',size=20)
plt.ylabel(r'Fourier coefficients',size=20)
plt.title(r'Fourier coefficients of the first function')
```

```
plt.show()

plt.semilogy(ab_x,ab_2,'ro')
plt.semilogy(ab_x,np.abs(c_2),'go')
plt.grid(True)
plt.xlabel(r'Index',size=20)
plt.ylabel(r'Fourier coefficients',size=20)
plt.title(r'Fourier coefficients of the second function')
plt.show()

plt.loglog(ab_x,ab_2,'ro')
plt.loglog(ab_x,np.abs(c_2),'go')
plt.grid(True)
plt.xlabel(r'Index',size=20)
plt.ylabel(r'Fourier coefficients',size=20)
plt.title(r'Fourier coefficients of the second function')
plt.show()

Ac1 = np.dot(A,c_1)
Ac2 = np.dot(A,c_2)

#Calculating the maximum deviation in the calculated values of
#fourier coefficients for both the functions:
md1 = np.max(np.abs(c_1 - ab_1))
md2 = np.max(np.abs(c_2 - ab_2))
print("The maximum deviation for the first function is",md1)
print("The maximum deviation for the first function is",md2)

## print(Ac1)
## print(Ac2)

## print(A.shape, c_1.shape, c_2.shape, Ac1.shape, Ac2.shape)

#Plots of the estimated function values using
#the fourier coefficients obtained:
plt.semilogy(x_ls,b_1,'ro',markersize=3)
plt.semilogy(x_ls,Ac1,'go',markersize=3)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$exp(x)$',size=20)
plt.title(r'The first function in semi log scale along y axis')
plt.show()

plt.plot(x_ls,b_1,'ro',markersize=3)
```

```
plt.plot(x_ls,Ac1,'go',markersize=3)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$exp(x)$',size=20)
plt.title(r'The first function')
plt.show()

plt.plot(x_ls,b_2,'ro',markersize=5)
plt.plot(x_ls,Ac2,'go',markersize=3)
plt.grid(True)
plt.xlabel(r'$x$',size=20)
plt.ylabel(r'$cos(cos(x))$',size=20)
plt.title(r'The second function')
plt.show()

# End Of File
```

Results and explanations

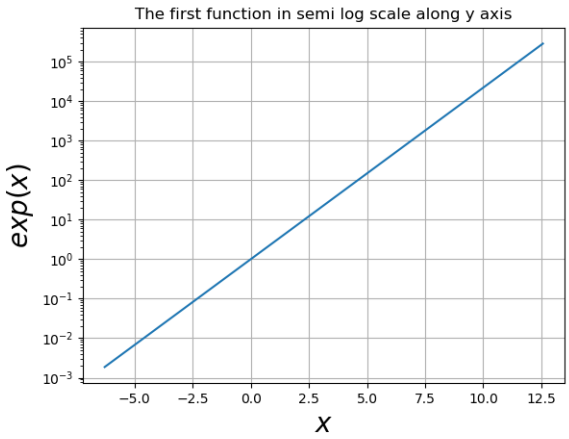

Figure 1: Plot of $f_1(x)$ against x in semi log scale

First of all, the functions $f_1(x)$ and $f_2(x)$ were defined. $f_1(x)$ is plotted in semilog scale in Figure 1 and in log log scale in Figure 2, both from -2π to 4π as mentioned in the question. $f_2(x)$ is plotted in Figure 3 from -2π to 4π.

Figures 4-7 are visualisations of absolute values of fourier coefficients ob-

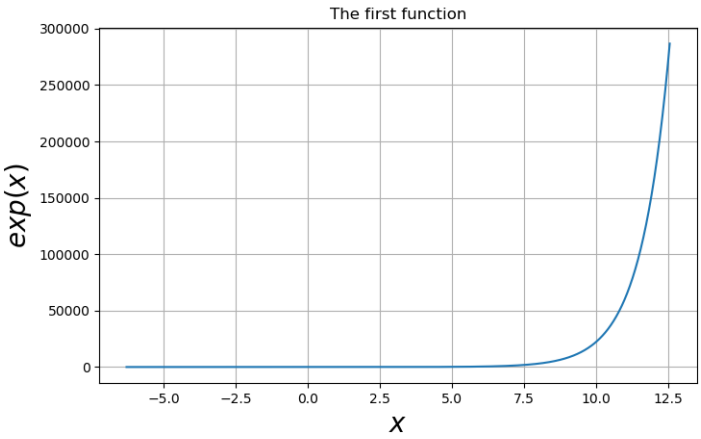

Figure 2: Plot of $f_1(x)$ against x

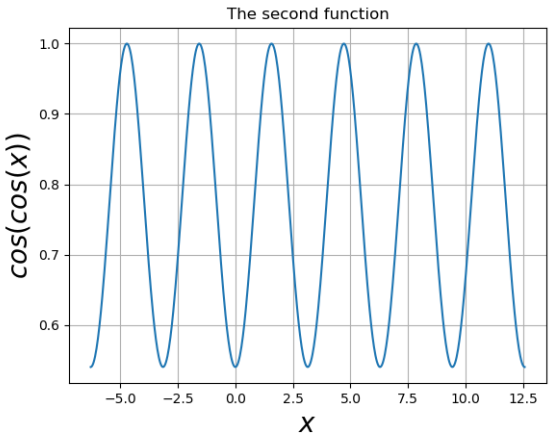

Figure 3: Plot of $f_2(x)$ against x

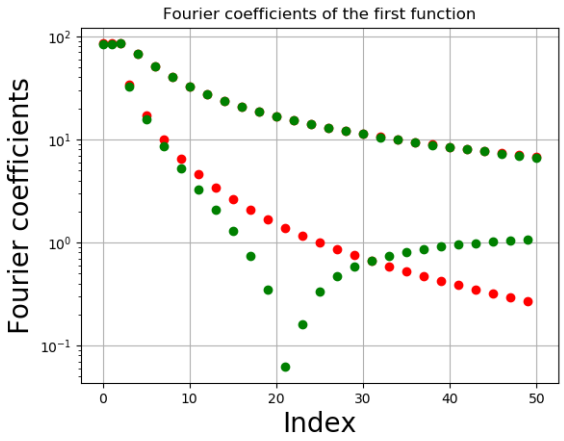

Figure 4: Plot of absolute values of fourier coefficients obtained using Method 1 (red dots) and Method 2 (green dots) for $f_1(x)$ in semi log scale

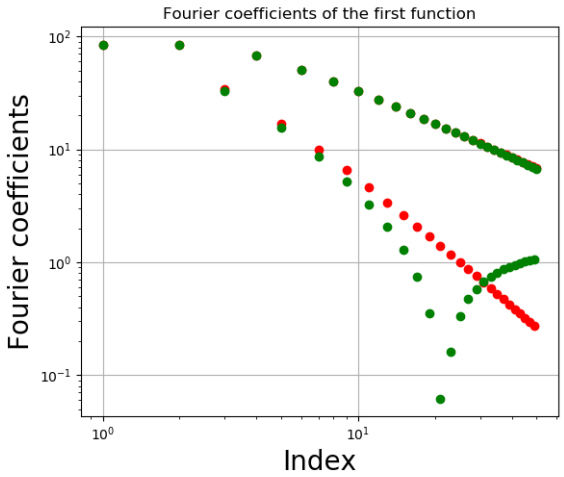

Figure 5: Plot of absolute values of fourier coefficients obtained using Method 1 (red dots) and Method 2 (green dots) for $f_1(x)$ in log log scale

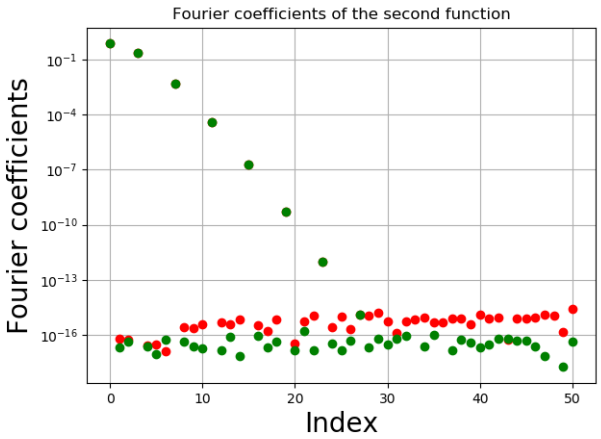

Figure 6: Plot of absolute values of fourier coefficients obtained using Method 1 (red dots) and Method 2 (green dots) for $f_2(x)$ in semi log scale

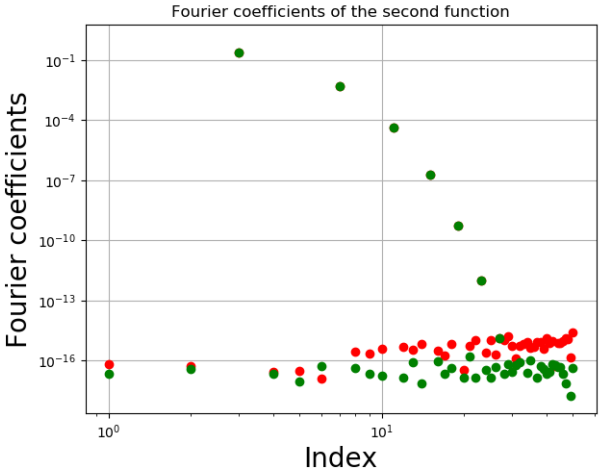

Figure 7: Plot of absolute values of fourier coefficients obtained using Method 1 (red dots) and Method 2 (green dots) for $f_2(x)$ in log log scale

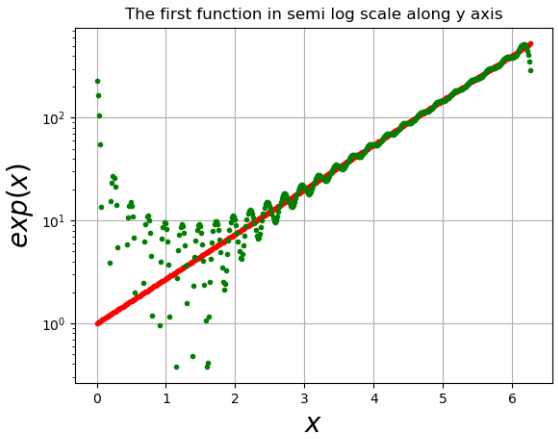

Figure 8: The estimate of $f_1(x)$ using the fourier coefficients obtained from methods 1 (red dots) and 2 (green dots) in semi log scale

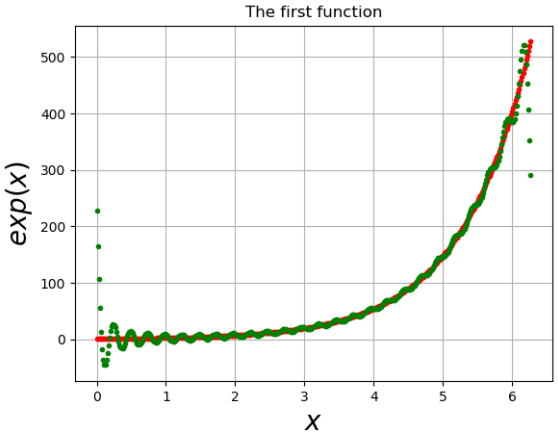

Figure 9: The estimate of $f_1(x)$ using the fourier coefficients obtained from methods 1 (red dots) and 2 (green dots)

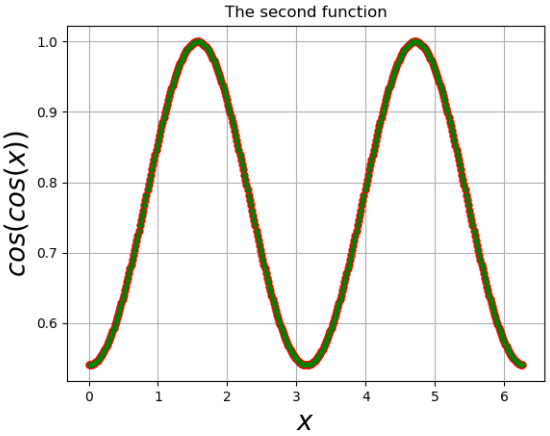

Figure 10: The estimate of $f_2(x)$ using the fourier coefficients obtained from methods 1 (red dots) and 2 (green dots)

tained by the two methods explained. The red dots represent the coefficients obtained using Method 1 and the green dots represent coefficients obtained using method 2. These are plotted in semilog scale as well as log log scale for both the functions. Since the $f_2(x)$ is an even function, we can observe that the coefficients b_i's are nearly zeroes.

Figures 8-10 show the estimates of the functions using the fourier coefficients obtained by the two methods. The red dots represent the function values obtained by the coefficients from method 1 and the green dots represent the function values obtained by the coefficients from method 2.

We can observe that the estimated functions by both sets of fourier coefficients are almost the same for $f_2(x)$. This is because of the periodicity of the function. Since the function is periodic, higher order coefficients are negligible.

We also know that the least squares error works well in the case of gaussian noise. Here, as we know there is no gaussian noise and hence least squares fitting is not expected to give very good results, hence the deviation in the coefficients obtained mainly in $f_1(x)$.

The maximum obtained deviation for the first function is 170.1304798253409 and for the second function is 0.4596139397276019. This is numerical proof for the explanation given above that $f_1(x)$ will have a high deviation, while $f_2(x)$ will not.