EE2703 ASSIGNMENT 4

Nandha Varman
EE19B043

1 Outline:

In this assignment, we will use the concept of Fourier series to approximate functions: $exp(x)$ and $cos(cos(x))$.

We will employ two methods to find the coefficients, namely **Direct Integration** and **Least Squares Estimation**. We will also compare the results of the two methods and then see how close the Fourier approximation is to the actual functions.

2 Theory:

Any periodic function can be written as a sum of shifted sinusoids We will apply this idea of Fourier series in this assignment. Namely a function can be approximated in the domain $[0, 2\pi]$ by repeating its values in $[0, 2\pi]$, to make it a periodic function with periodic 2π.

The Fourier series of a function $f(x)$ is given by

$$f(x) = a_0 + \sum_{k=1}^{\infty} a_k cos(kx) + b_k sin(kx)$$

where,

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

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

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

3 Tasks:

3.1 Part 1:

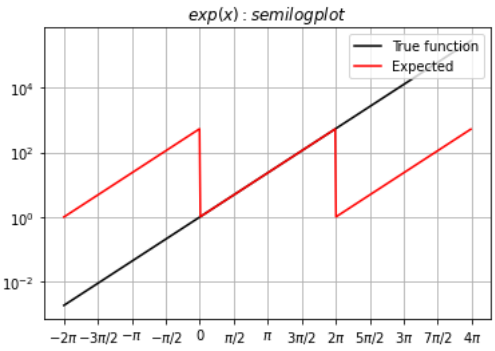
- Define Python functions for the two functions e^x and $cos(cos(x))$ which return a vector (or scalar) value.
- Plot the functions over the interval $[-2\pi, 4\pi)$.
- Discuss periodicity of both functions.
- Plot the expected functions from Fourier series.

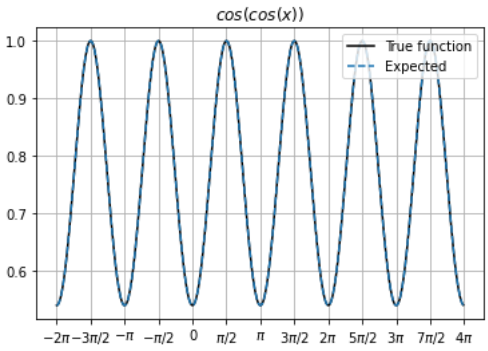
3.1.1 Code:

```
#importing necessary modules
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import quad

PI = np.pi

# functions for exp(x) and cos(cos(x))
def f1(x):
    return np.exp(x)
def f2(x):
    return np.cos(np.cos(x))

t = np.linspace(-2*PI, 4*PI, 500 )
t = t[0:-1]

# lists for locations and labels for ticks
locs = [-2*PI, -3/2*PI, -PI,-PI/2, 0, PI/2, PI, 3*PI/2, 2*PI, 5*PI/2, 3*PI, 7*PI/2, 4*PI]
labels = ['$-2\pi$', '$-3\pi/2$', '$-\pi$','$-\pi/2$', 0, '$\pi/2$', '$\pi$', '$3\pi/2$', '$2\

# plotting exp(x) as a semilog plot
plt.figure(1)
plt.xticks(locs, labels)
plt.semilogy(t,f1(t),'k', label = 'True function')
plt.semilogy(t,f1(t%(2*PI)),'r', label = 'Expected')
plt.legend(loc='upper right')
```

```
plt.title('$exp(x): semilog plot$')
plt.grid()
plt.show()

# plotting cos(cos(x))
plt.figure(2)
plt.xticks(locs, labels)
plt.plot(t,f2(t),'k', label = 'True function')
plt.plot(t,f2(t%(2*PI)),'--', label = 'Expected')
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$')
plt.grid()
plt.show()
```

3.1.2 Plots:





3.1.3 Observations:

1. From the above plots, you can easily see that $cos(cos(x))$ is periodic with period 2π,and e^x isn't periodic and rises monotonically.

2. From the Fourier series, we expect that the Fourier approximation of the function would be repetitions of the function between $[0, 2\pi)$.

3.2 Part 2:

- Obtain the first 51 coefficients, i.e., a_0, a_1, b_1,... for e^x and $cos(cos(x))$ using the quad function.

- Calculate the function using those coefficients and compare with original functions graphically.

3.2.1 Code:

```
def cos_coeff(x, k, h):
    return h(x)*np.cos(k*x)
def sin_coeff(x, k, h):
    return h(x)*np.sin(k*x)

# function to find the first 51 fourier series coefficients
def fourier_coeff_51(h):
    coeff = np.zeros(51)
    coeff[0] = quad(cos_coeff, 0, 2*PI, args =(0, h))[0]/(2*PI)
    for i in range(1,26):
        coeff[2*i-1] = quad(cos_coeff, 0, 2*PI, args =(i, h))[0]/(PI)
        coeff[2*i] = quad(sin_coeff, 0, 2*PI, args =(i, h))[0]/(PI)
    return  coeff

# arrays holding the first 51 FS coefficients in the form [ a0, a1, b1, a2, b2,..]
exp_coeff = fourier_coeff_51(f1)
coscos_coeff = fourier_coeff_51(f2)
```

3.2.2 Observations:

The function `fourier_coeff_51(h)` takes in the name of a function as argument and returns the first 51 coefficients of the function `h`, in order, as an array.

3.3 Part 3:

Make two different plots for each function using `semilogy` and `loglog` and plot the magnitude of the coefficients versus n.

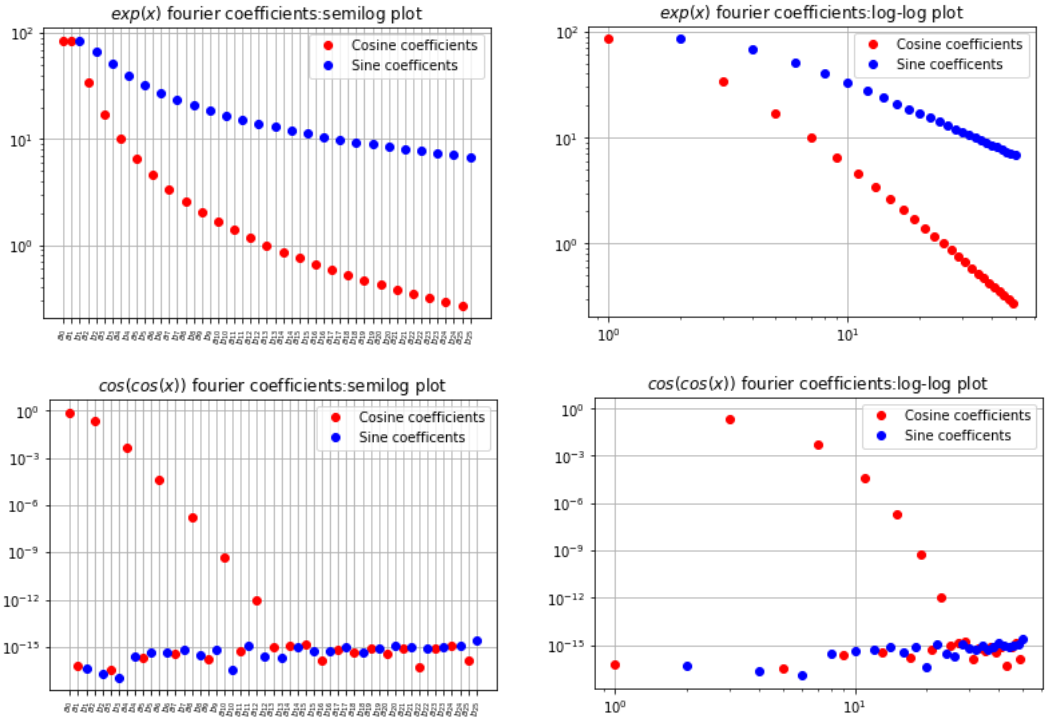
3.3.1 Code:

```
# generating lists for location and labels for the coefficents plot
n_locs = range(51)
```

```
n_labels = ['$a_0$']
for i in range(1,26):
    n_labels.append('$a_{'+str(i)+'}$')
    n_labels.append('$b_{'+str(i)+'}$')

# semi-log plot for FS coefficients of exp(x)
plt.figure(3)
plt.xticks(n_locs,n_labels , rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(0, np.abs(exp_coeff[0]), 'ro')
plt.semilogy(np.arange(1,51,2), np.abs(exp_coeff[1::2]),'ro', label = 'Cosine coefficients')
plt.semilogy(np.arange(2,51,2), np.abs(exp_coeff[2::2]),'bo', label = 'Sine coefficents')
plt.legend(loc='upper right')
plt.title('$exp(x)$ fourier coefficients:semilog plot')
plt.grid()
plt.show()

# log-log plot for FS coefficients of exp(x)
plt.figure(4)
plt.loglog(0, np.abs(exp_coeff[0]), 'ro')
plt.loglog(np.arange(1,51,2), np.abs(exp_coeff[1::2]),'ro', label = 'Cosine coefficients')
plt.loglog(np.arange(2,51,2), np.abs(exp_coeff[2::2]),'bo', label = 'Sine coefficents')
plt.legend(loc='upper right')
plt.title('$exp(x)$ fourier coefficients:log-log plot')
plt.grid()
plt.show()

# semi-log plot for FS coefficients of cos(cos(x))
plt.figure(5)
plt.xticks(n_locs,n_labels , rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(0, np.abs(coscos_coeff[0]), 'ro')
plt.semilogy(np.arange(1,51,2), np.abs(coscos_coeff[1::2]),'ro', label = 'Cosine coefficients'
plt.semilogy(np.arange(2,51,2), np.abs(coscos_coeff[2::2]),'bo', label = 'Sine coefficents')
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$ fourier coefficients:semilog plot')
plt.grid()
plt.show()

# log-log plot for FS coefficients of cos(cos(x))
```

```
plt.figure(6)
plt.loglog(0, np.abs(coscos_coeff[0]), 'ro')
plt.loglog(np.arange(1,51,2), np.abs(coscos_coeff[1::2]),'ro', label = 'Cosine coefficients')
plt.loglog(np.arange(2,51,2), np.abs(coscos_coeff[2::2]),'bo', label = 'Sine coefficents')
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$ fourier coefficients:log-log plot')
plt.grid()
plt.show()
```

3.3.2 Plots:



3.3.3 Observations:

1. From the above plots we observe that almost all b_n coefficients are very close to 0 for $cos(cos(x))$. This is expected due to the odd nature of the integrand between $[-\pi, \pi)$.

   It does not become exactly zero due to approximations in the implementation of the `quad` function.

2. Rate of decay of Fourier coefficients is determined by how smooth the function is. If a function is infinitely differentiable then its Fourier coefficients decay very fast. But if the k^{th} derivative of the function f, denoted by $f^{(k)}$, is discontinuous, then the rate of decay of the Fourier coefficients is only $\frac{1}{n^k}$.

   $cos(cos(x))$ is infinitely differentiable, hence it's Fourier coefficients decay very fast, while that of e^x decay very slowly due to the discontinuity in the Fourier approximation of the function at $2n\pi$.

3.4 Part 4:

- Use *Least Squares estimation* to find the Fourier coefficients of the functions, using `scipy.linalg.lstsq` .
- Build the coefficient matrix A and the constant matrix b.

3.4.1 Code:

```
# function to find the first 51 coefficients by least squares estimation
def Lstsq_FS_coeff(f):
    x=np.linspace(0,2*PI,401)
    x=x[:-1] # drop last term to have a proper periodic integral
    b=f(x) # f has been written to take a vector
    A=np.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]=np.cos(k*x) # cos(kx) column
        A[:,2*k]=np.sin(k*x) # sin(kx) column
    #endfor
    ls_coeffs=np.linalg.lstsq(A,b,rcond = -1)[0] # the '[0]' is to pull out the
    # best fit vector. lstsq returns a list.
    return ls_coeffs

# arrays holding the first 51 FS coefficients obtained by least squares estimation
exp_coeff_lstsq = Lstsq_FS_coeff(f1)
coscos_coeff_lstsq = Lstsq_FS_coeff(f2)
```

3.5 Part 5:

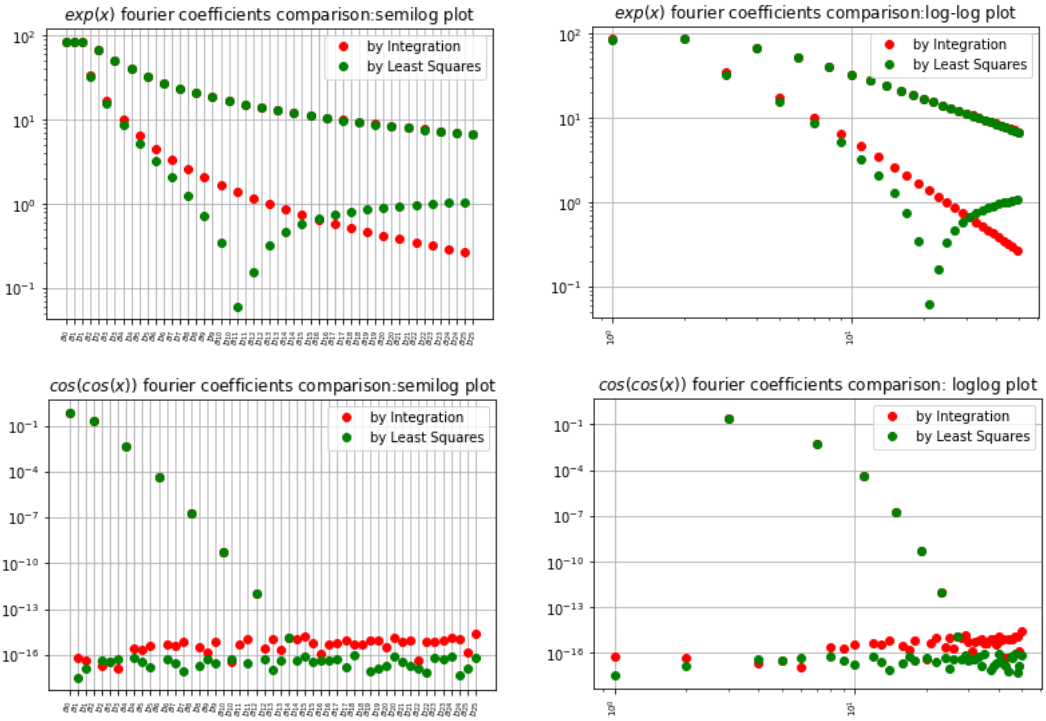
- Compare the coefficients obtained through the *least squares method* and the *direct integration* method.
- Find the maximum deviation between the coefficients obtained in the two methods.

3.5.1 Code:

```
# semi-log comparison plot of the FS coefficents obtained by integrationa and
# Least squares estimation for exp(x)
plt.figure(7)
plt.xticks(n_locs, n_labels, rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(range(51), np.abs(exp_coeff),'ro', label = 'by Integration')
plt.semilogy(range(51), np.abs(exp_coeff_lstsq),'go', label = 'by Least Squares')
plt.legend(loc='upper right')
plt.title('$exp(x)$ fourier coefficients comparison:semilog plot')
plt.grid()
plt.show()

# semi-log comparison plot of the FS coefficents obtained by integrationa and
# Least squares estimation for exp(x)
plt.figure(8)
plt.xticks(n_locs, n_labels, rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.loglog(range(51), np.abs(exp_coeff),'ro', label = 'by Integration')
plt.loglog(range(51), np.abs(exp_coeff_lstsq),'go', label = 'by Least Squares')
plt.legend(loc='upper right')
plt.title('$exp(x)$ fourier coefficients comparison:log-log plot')
plt.grid()
plt.show()

# semi-log comparison plot of the FS coefficents obtained by integrationa and
# Least squares estimation for cos(cos(x))
plt.figure(9)
plt.xticks(n_locs, n_labels, rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(range(51), np.abs(coscos_coeff),'ro', label = 'by Integration')
plt.semilogy(range(51), np.abs(coscos_coeff_lstsq),'go', label = 'by Least Squares')
```

```
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$ fourier coefficients comparison:semilog plot')
plt.grid()
plt.show()

# log-log comparison plot of the FS coefficents obtained by integrationa and
# Least squares estimation for cos(cos(x))
plt.figure(10)
plt.xticks(n_locs, n_labels, rotation = '90')
plt.tick_params(axis='x', labelsize=7)
plt.loglog(range(51), np.abs(coscos_coeff),'ro', label = 'by Integration')
plt.loglog(range(51), np.abs(coscos_coeff_lstsq),'go', label = 'by Least Squares')
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$ fourier coefficients comparison: loglog plot')
plt.grid()
plt.show()

# finding the maximum absolute deviation between the FS coefficients obtained by the two metho
abs_error_exp = [abs(exp_coeff[i]-exp_coeff_lstsq[i]) for i in range(len(exp_coeff))]
abs_error_coscos = [abs(coscos_coeff[i]-coscos_coeff_lstsq[i]) for i in range(len(coscos_coeff


print('Largest deviation for exp(x): '+str(max(abs_error_exp)))
print('Largest deviation for cos(cos(x)): '+str(max(abs_error_coscos)))
```

3.5.2 Plots:



3.6 Part 6:

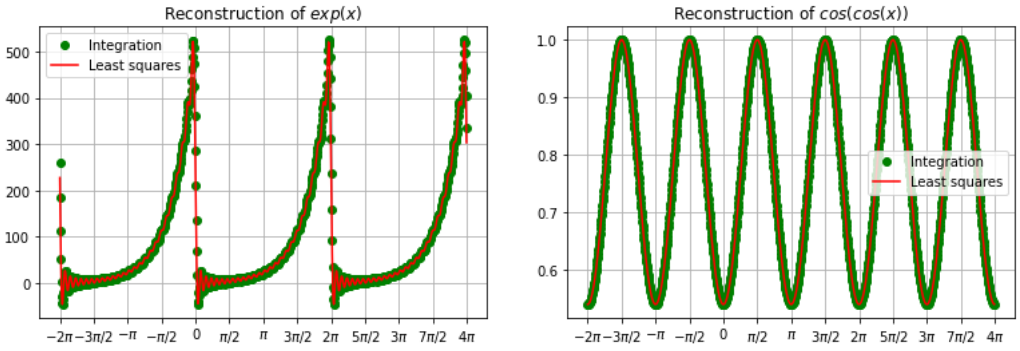
- Computing $A \cdot c$ from the estimated values of c by *Least Squares Method* and plotting them.

3.6.1 Code:

```
x = np.linspace(-2*PI, 4*PI, 1001)
x = x[:-1]

A = np.zeros((1000,51))
A[:,0] = 1
for k in range(1,26):
    A[:,2*k-1]=np.cos(k*x)
    A[:,2*k]=np.sin(k*x)

# plot showing reconstruction of exp(x) using the obtained FS coefficients
```

```
plt.figure(11)
plt.xticks(locs, labels)
plt.plot(x, np.matmul(A, exp_coeff ), 'go', label='Integration')
plt.plot(x, np.matmul(A, exp_coeff_lstsq), 'r', label='Least squares')
plt.title('Reconstruction of $exp(x)$')
plt.legend()
plt.grid()

# plot showing reconstruction of cos(cos(x)) using the obtained FS coefficients
plt.figure(12)
plt.xticks(locs, labels)
plt.plot(x, np.matmul(A, coscos_coeff ), 'go', label='Integration')
plt.plot(x, np.matmul(A, coscos_coeff_lstsq), 'r', label='Least squares')
plt.title('Reconstruction of $cos(cos(x))$')
plt.legend()
plt.grid()
```

3.6.2 Plots:



3.6.3 Observations:

- We observe that there is a significant deviation for e^x at its discontinuities at $2n\pi$. There are oscillations around the discontinuity points and their ripple amplitude decrease as we go close to discontinuity.This is called **Gibbs phenomenon**.
- Due to this, the original function and one which is reconstructed using least squares will not fit exactly.

- And as we know that Fourier series is used to define periodic signals in frequency domain. e^x is a aperiodic signal and we can't define an aperiodic signal on an interval of finite length.
- For $\cos(\cos(x))$ the curves fit almost perfectly because the function itself is a periodic function and it is continuous everywhere, so we get very negligible deviation and can reconstruct the signal with just the Fourier coefficients.

4 Conclusions:

The Fourier estimation of e^x does not match accurately with the function close to 0, but matches almost exactly in the case of $\cos(\cos(x))$. This is due to the presence of a discontinuity at $x = 0$ for the periodic extension of e^x. This discontinuity leads to non-uniform convergence of the Fourier series, with different rates for both the functions.

The mismatch in the Fourier approximation of e^x is explained by Gibbs Phenomenon.

We can hence conclude that the Fourier Series Approximation Method works extremely well for smooth periodic functions, but gives bad results for discontinuous periodic functions.