EE2703: Assignment 4

Akilesh Kannan(EE18B122)

February 19, 2020

1 Abstract

In this assignment, I look at approximating a few functions using the Fourier series. I will employ two methods to find the Fourier approximation of e^x and $cos(cos(x))$:

(a) Direct Integration

(b) Least Squares Method

I shall also compare the Fourier coefficients I have got in the two methods and see how close the Fourier approximation is to the true function.

2 Theory

Any periodic function can be expressed as a linear combination of various sinusoids. This is the underlying principle of the Fourier Approximation. We can also approximate any function in the domain [0,2π) by slightly 'tweaking' the mathematics of the Fourier series by extending copies of the function between [0,2π) till infinity, thus creating a periodic function.

For a function $f(x)$, it's Fourier series is given as:

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

where,

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

$$\begin{aligned} a_k &= \frac{1}{\pi}\int_0^{2\pi} f(x)cos(kx)dx \\ b_k &= \frac{1}{\pi}\int_0^{2\pi} f(x)sin(kx)dx \end{aligned} \tag{3}$$

The above equations describe the *Direct Integration* method of finding the Fourier approximation. We shall also use the *Least Squares* method we learnt in the last assignment to find the Fourier approximation.

3 Tasks

I shall now walk through the tasks that were asked to be performed and also include the necessary portions of the code and any associated plots.

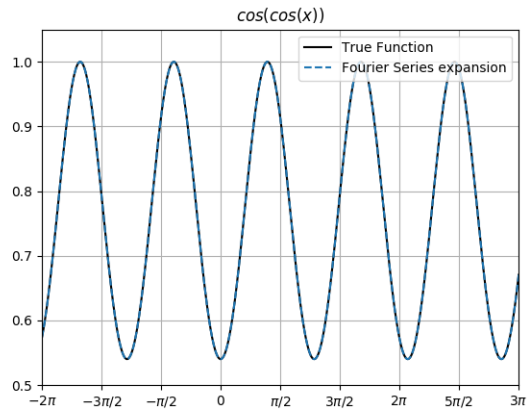
3.1 Question 1

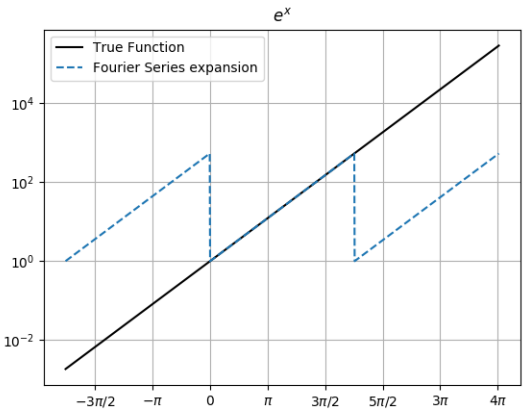
- Define Python functions for the two functions e^x and $cos(cos(x))$ which return a vector (or scalar) value.
- Plot the functions over the interval $[-2\pi, 4\pi)$.
- Discuss periodicity of both functions.
- Plot the expected functions from Fourier series.

3.1.1 Code

```
def coscosxFunc(x):
    return np.cos(np.cos(x))
def expFunc(x):
    return np.exp(x)

PI = np.pi
x = np.linspace(-2*PI, 4*PI, 1201)
x = x[:-1] # drop last entry to get a nice period of integration

plt.figure('Figure 1')
ax1 = plt.axes()
ax1.xaxis.set_major_formatter(plt.FuncFormatter(pi_tick)) # x-axis to tick in multiples of pi/2
plt.plot(x, coscosxFunc(x), 'k', label='True Function')
plt.plot(x, coscosxFunc(x%(2*PI)), '--', label='Fourier Series expansion')
plt.axis([-6, 10, 0.5, 1.05])
plt.legend(loc='upper right')
plt.title('$cos(cos(x))$')
plt.grid()
plt.show()

plt.figure('Figure 2')
ax2 = plt.axes()
ax2.xaxis.set_major_formatter(plt.FuncFormatter(pi_tick)) # x-axis to tick in multiples of pi/2
plt.semilogy(x, expFunc(x), 'k', label='True Function')
plt.semilogy(x, expFunc(x%(2*PI)), '--', label='Fourier Series expansion')
plt.legend(loc='upper left')
plt.title('$e^x$')
plt.grid()
plt.show()
```

3.1.2 Plots





3.1.3 Discussions

1. From the above plots, you can easily see that while $cos(cos(x))$ is periodic with period 2π, the latter, 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 $[0, 2\pi)$. We can thus say that the Fourier approximation of the function $f(x)$ is $f(x \mod 2\pi)$[1].

3.2 Question 2

- Obtain the first 51 coefficients i.e., a_0, a_1, b_1,... for e^x and $cos(cos(x))$ using `quad` function.
- Calculate the function using those coefficients and compare with original functions graphically.

3.2.1 Code

```
def cosCoeff(x, k, f):
    return f(x)*np.cos(k*x)
def sinCoeff(x, k, f):
    return f(x)*np.sin(k*x)

def calc51FourierCoeffs (f):
    aCoeff = np.zeros(26)
    bCoeff = np.zeros(26)
    aCoeff[0] = quad(cosCoeff, 0, 2*PI, args=(0, f))[0]/(2*PI)
    for i in range(1, 26):
        aCoeff[i] = quad(cosCoeff, 0, 2*PI, args=(i, f))[0]/(PI)
        bCoeff[i] = quad(sinCoeff, 0, 2*PI, args=(i, f))[0]/(PI)
    coeffs = np.zeros(51)
    coeffs[0] = aCoeff[0]
    coeffs[1::2] = aCoeff[1:]
    coeffs[2::2] = bCoeff[1:]
    return coeffs

coeffCosCos = calc51FourierCoeffs(coscosxFunc)
coeffExp = calc51FourierCoeffs(expFunc)
```

3.2.2 Discussions

The function `calc51FourierCoeffs(f)` takes in the name of a function as argument and returns the first 51 coefficients of the function `f`, in order, as an array. One advantage of hard-coding the number 51 is that it becomes faster, while the hard-coding makes it difficult for the function to be generalised.

3.3 Question 3

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

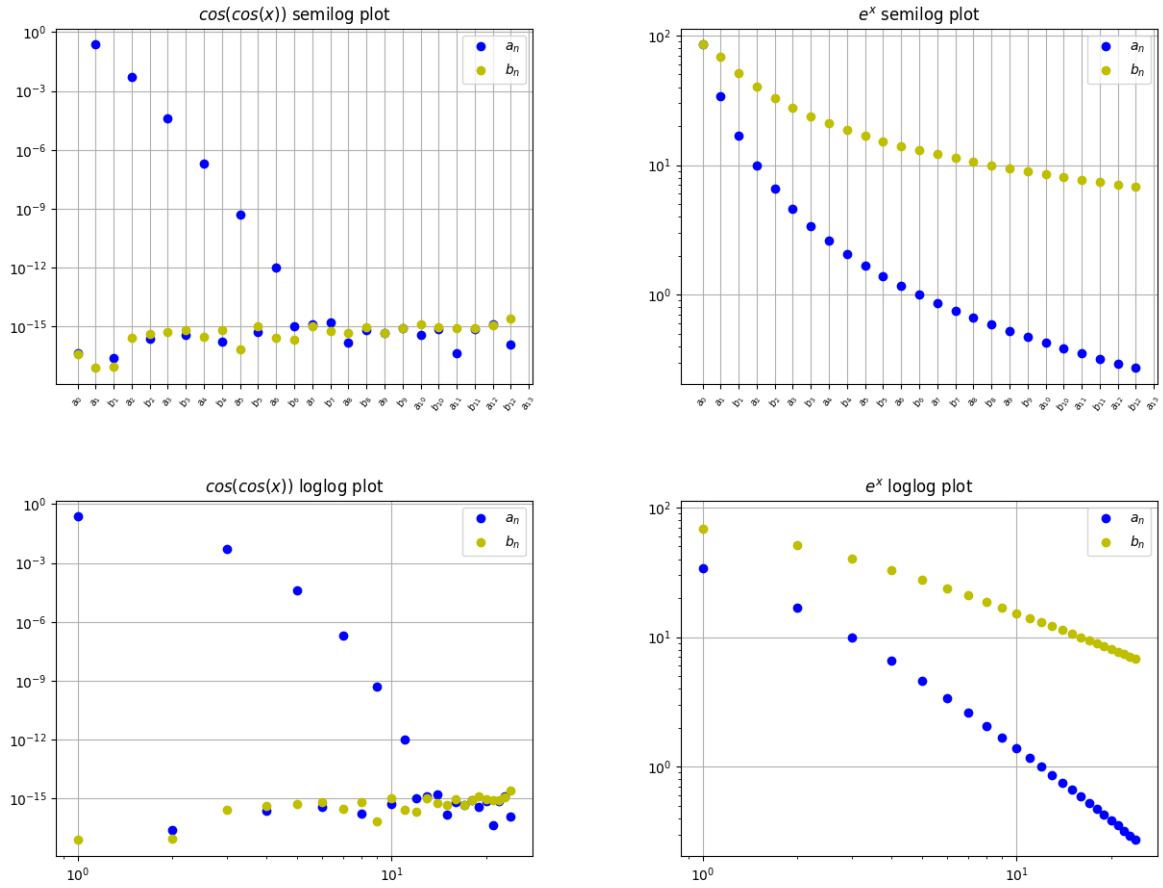
3.3.1 Code

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

plt.figure('Figure 3')
plt.xticks(np.arange(51), xTicksForCoeffsSemilog, rotation=60)
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(abs(coeffCosCos[1::2]), 'bo', label='$a_n$')
```

[1] It is actually $f(x \mod p)$, where p the minimum among the period of the periodic function and 2π.

```
plt.semilogy(abs(coeffCosCos[2::2]), 'yo', label='$b_n$')
plt.legend()
plt.title('$cos(cos(x))$ semilog plot')
plt.grid()
plt.savefig(plotsDir+'Figure 3.png')

plt.figure('Figure 4')
plt.xticks(np.arange(51), xTicksForCoeffsSemilog, rotation=60)
plt.tick_params(axis='x', labelsize=7)
plt.semilogy(abs(coeffExp[1::2]), 'bo', label='$a_n$')
plt.semilogy(abs(coeffExp[2::2]), 'yo', label='$b_n$')
plt.legend()
plt.title('$e^x$ semilog plot')
plt.grid()
plt.savefig(plotsDir+'Figure 4.png')

plt.figure('Figure 5')
plt.loglog(abs(coeffCosCos[1::2]), 'bo', label='$a_n$')
plt.loglog(abs(coeffCosCos[2::2]), 'yo', label='$b_n$')
plt.legend()
plt.title('$cos(cos(x))$ loglog plot')
plt.grid()
plt.savefig(plotsDir+'Figure 5.png')

plt.figure('Figure 6')
plt.loglog(abs(coeffExp[1::2]), 'bo', label='$a_n$')
plt.loglog(abs(coeffExp[2::2]), 'yo', label='$b_n$')
plt.legend()
plt.title('$e^x$ loglog plot')
plt.grid()
plt.savefig(plotsDir+'Figure 6.png')
```

3.3.2 Plots



3.3.3 Discussions

(a) As we can see from the above plots, almost all b_n coefficients are very close to 0 for $cos(cos(x))$. This is expected - we find by hand integration that the b_n coefficients are 0 (due to the odd nature of the integrand between $[-\pi, \pi)$.

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

(b) 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 $\frac{1}{n^k}$.

Since $cos(cos(x))$ is an infinitely differentiable function, 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 $2n\pi$.

(c)

3.4 Questions 4 & 5

- Use a *Least Squares* approach 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

```
1 def findLSTSQCoeff(f):
2     x = np.linspace(0, 2*PI, 401)
```

```
    x = x[:-1]
    b = f(x)
    M = np.zeros((400, 51))
    M[:,0] = 1
    for k in range(1,26):
        M[:,(2*k)-1]=np.cos(k*x)
        M[:,2*k]=np.sin(k*x)
    return np.linalg.lstsq(M, b, rcond=None)[0]

lstsqCosCos = findLSTSQCoeff(coscosxFunc)
lstsqExp = findLSTSQCoeff(expFunc)

plt.figure('Figure 3.1')
plt.semilogy(abs(coeffCosCos[1::2]), 'ro', label='$a_n$ by Integration')
plt.semilogy(abs(coeffCosCos[2::2]), 'bo', label='$b_n$ by Integration')
plt.semilogy(abs(lstsqCosCos[1::2]), 'go', label='$a_n$ by lstsq')
plt.semilogy(abs(lstsqCosCos[2::2]), 'yo', label='$b_n$ by lstsq')
plt.title('Comparing $cos(cos(x))$ FS coefficients - semilogy plot')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 3.1.png')

plt.figure('Figure 4.1')
plt.semilogy(abs(coeffExp[1::2]), 'ro', label='$a_n$ by Integration')
plt.semilogy(abs(coeffExp[2::2]), 'bo', label='$b_n$ by Integration')
plt.semilogy(abs(lstsqExp[1::2]), 'go', label='$a_n$ by lstsq')
plt.semilogy(abs(lstsqExp[2::2]), 'yo', label='$b_n$ by lstsq')
plt.title('Comparing $e^x$ FS coefficients - semilogy plot')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 4.1.png')

plt.figure('Figure 5.1')
plt.loglog(abs(coeffCosCos[1::2]), 'ro', label='$a_n$ by Integration')
plt.loglog(abs(coeffCosCos[2::2]), 'bo', label='$b_n$ by Integration')
plt.loglog(abs(lstsqCosCos[1::2]), 'go', label='$a_n$ by lstsq')
plt.loglog(abs(lstsqCosCos[2::2]), 'yo', label='$b_n$ by lstsq')
plt.title('Comparing $cos(cos(x))$ FS coefficients - loglog plot')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 5.1.png')

plt.figure('Figure 6.1')
plt.loglog(abs(coeffExp[1::2]), 'ro', label='$a_n$ by Integration')
plt.loglog(abs(coeffExp[2::2]), 'bo', label='$b_n$ by Integration')
plt.loglog(abs(lstsqExp[1::2]), 'go', label='$a_n$ by lstsq')
plt.loglog(abs(lstsqExp[2::2]), 'yo', label='$b_n$ by lstsq')
plt.title('Comparing $e^x$ FS coefficients - loglog plot')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 6.1.png')
```

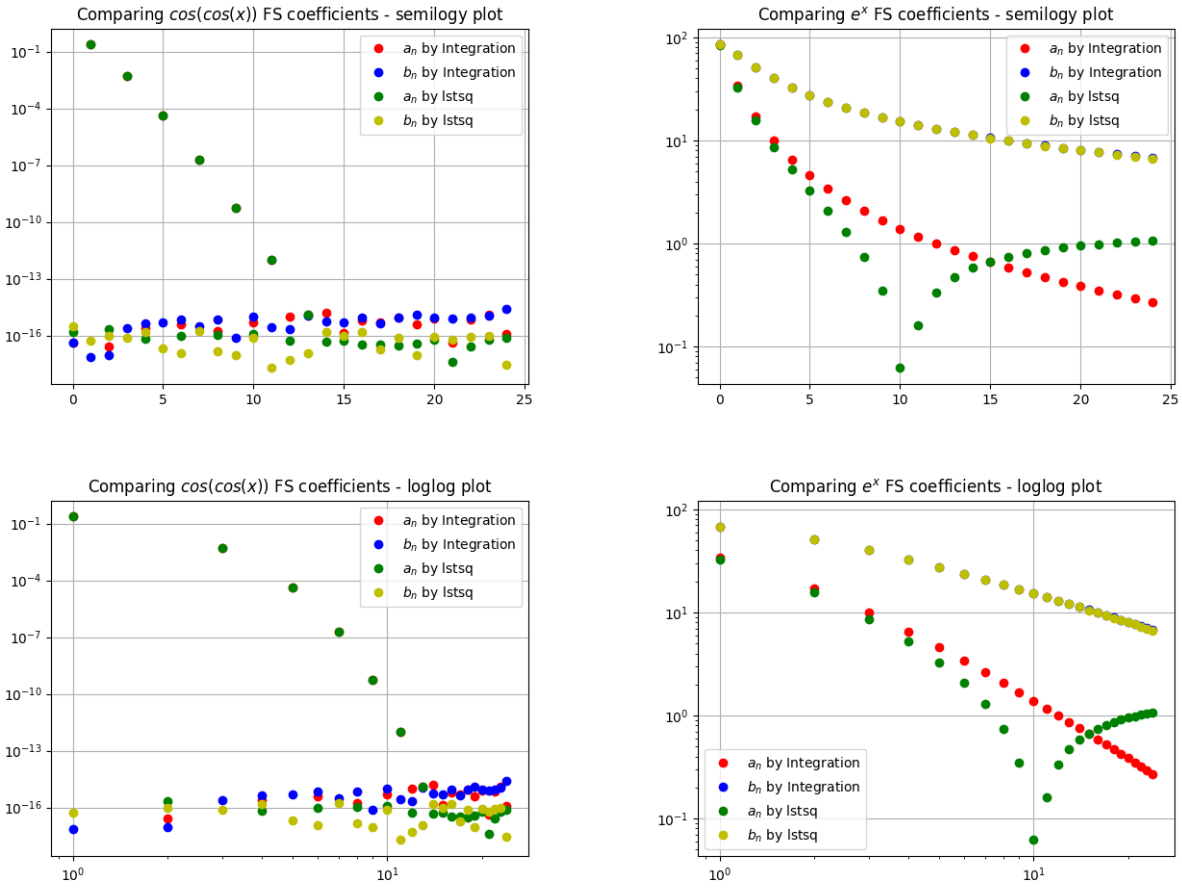
3.5 Question 6

- 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

```
absErrorsCosCos = [abs(coeffCosCos[i]-lstsqCosCos[i]) for i in
↪  range(len(coeffCosCos))]
absErrorsExp = [abs(coeffExp[i]-lstsqExp[i]) for i in range(len(coeffExp))]

print('Max. deviation for cos(cos(x)): '+str(max(absErrorsCosCos)))
print('Max. deviation for exp(x): '+str(max(absErrorsExp)))
```

3.5.2 Plots



3.5.3 Discussion

3.6 Question 7

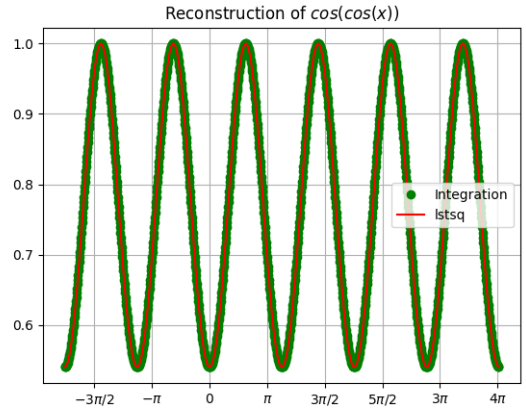
- Computing $A \cdot c$ from the estimated values of c by *Least Squares Method* and plotting them.

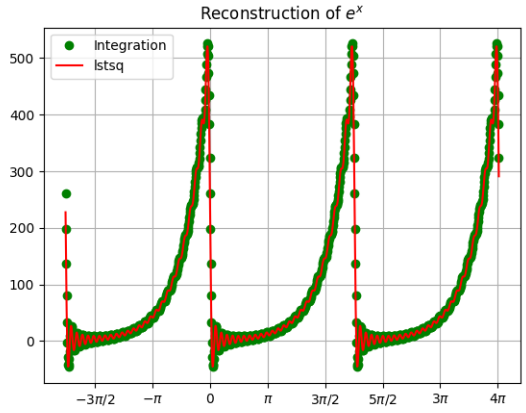
3.6.1 Code

```
xNew = np.linspace(-2*PI, 4*PI, 1201)
xNew = xNew[:-1]

matrixA = np.zeros((1200,51))
matrixA[:,0] = 1
```

```
for k in range(1,26):
    matrixA[:,(2*k)-1]=np.cos(k*xNew)
    matrixA[:,2*k]=np.sin(k*xNew)

plt.figure('Figure 7')
ax3 = plt.axes()
ax3.xaxis.set_major_formatter(plt.FuncFormatter(pi_tick))
plt.plot(xNew, matrixA@coeffCosCos, 'go', label='Integration')
plt.plot(xNew, matrixA@lstsqCosCos, 'r', label='lstsq')
plt.title('Reconstruction of $cos(cos(x))$')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 7.png')

plt.figure('Figure 8')
ax4 = plt.axes()
ax4.xaxis.set_major_formatter(plt.FuncFormatter(pi_tick))
plt.plot(xNew, matrixA@coeffExp, 'go', label='Integration')
plt.plot(xNew, matrixA@lstsqExp, 'r', label='lstsq')
plt.title('Reconstruction of $e^x$')
plt.legend()
plt.grid()
plt.savefig(plotsDir+'Figure 8.png')
```

3.6.2 Plots





3.6.3 Discussion

- As we observe that there is a significant deviation for e^x as it has discontinuities at $2n\pi$ which can be observed in the figure, and so there will be **Gibbs phenomenon** i.e, there will be oscillations around the discontinuity points and their ripple amplitude will decrease as we go close to discontinuity. In this case it is at 2π for e^x.
- As we observe that ripples are high initially and reduces and oscillate with more frequency as we go towards 2π. This phenomenon 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 and e^x is a aperiodic signal so you can't define an aperiodic signal on an interval of finite length (if you try, you'll lose information about the signal), so one must use the Fourier transform for such a signal.
- That is why there are significant deviations for e^x from original function.

- Forcos(cos(x)) the curves fit almost perfectly because the function itself is a periodic function and it is a continuous function everywhere, so we get very negligible deviation and able to reconstruct the signal with just the Fourier coefficients.

4 Conclusion

We see that the Fourier estimation of e^x does not match significantly with the function close to 0, but matches near perfectly 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 difference in the rates of convergence leads to the **Gibb's phenomenon**, which is the ringing observed at discontinuities in the Fourier estimation of a discontinuous function. This explains the mismatch in the Fourier approximation for e^x.

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