EE2703 : Applied Programming Lab
Experiment 4

Ch Venkat Sai
EE18B042

February 27, 2020

Section - 1

The experiment is about:
Fitting functions into fourier series(periodic and non-periodic)
Finding the bestfit coefficients for limited n
Comparing them with actual coefficients
Plotting graphs to compare

Section - 2

A periodic function with a period of 2π can be expressed in terms of series of sin and cos which is also called fourier series. For a function f the fourier series is given by

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

where the coefficients are given by

$$\mathrm{a}_0 = \frac{1}{2\pi} \int\limits_0^{2\pi} f(x)\mathrm{d}x$$
$$\mathrm{a}_n = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos nx\mathrm{d}x$$
$$\mathrm{b}_n = \frac{1}{\pi} \int\limits_0^{2\pi} f(x) \sin nx\mathrm{d}x$$

Section - 3

We now write here a two functions in the code defining the coscos and exponential functions which can either take an array as an input and or just a float number. Here the instance compares the type of variable and gives a boolean output. The functions are plotted in Figure-1 and Figure-2. Along with the two plots the expected fourier plots are also shown in dashed blue. Since the exponential is not periodic they don't match except for the region in 0 to 2π. Whereas since coscos is periodic we have a match between the two plots.

```
def exponen(a):            # defining exponential function
    b = []
    if isinstance(a, float):        # for float values
        return math.exp(a)
    else:
        for i in range(len(a)):         # for vectors
            b.append(math.exp(a[i]))
        return b


def coscos(a):          # defining coscos
    b = []
    if isinstance(a, float):
        return math.cos(math.cos(a))        # for float values
    else:
        for i in range(len(a)):             # for vectors
            b.append(math.cos(math.cos(a[i])))
        return b
```

Section - 4

We now obtain the first 51 coefficients for both the functions using above formulae. To integrate we used a quad function and lambda notation. The lambda notation creates the function to be integrated with proper ordering of the parameters. The quad function integrates with the limits 0 and 2π and the for loop changes the parameter k.

```
    if m == 0:
        u = lambda x, k : exponen(float(x))*cos(k*x)
        v = lambda x, k : exponen(float(x))*sin(k*x)
    elif m ==1:
        u = lambda x, k : coscos(float(x))*cos(k*x)
        v = lambda x, k : coscos(float(x))*sin(k*x)

    for k in range(26):
        a = teg.quad(u, 0, 2*math.pi, args=(k))[0]

    for k in range(26):
        a = teg.quad(u, 0, 2*math.pi, args=(k))[0]      # inntegrating
        if k == 0:
            a = a/(2*math.pi)
        else:
            a = a/(math.pi)
        N += [[k]]
        V += [[a]]                    # appending corresponding value
        if k != 0:
            b = teg.quad(v, 0, 2*math.pi, args=(k))[0]
            b = b/(math.pi)
            N += [[k]]
            V += [[b]]
    V_1 = array(V)
    N_1 = array(N)
```

Section - 5

Now after storing the corresponding values in an array we plot these values with the value of n in a semilogy plot as well as loglog plot. The plots for exponential are in Figure-3 and Figure-4. And the plots for coscos are in Figure-5 and Figure-6.

```
if m  == 0:                 # plotting bestfit vs coeff
    w = semilogy(N_1, abs(V_1), 'or', label='exponen coeff')
    semilogy(N_1, abs(c1), 'og',markersize=3, label='exponen bestfit')
elif m == 1:
    semilogy(N_1, abs(V_1), 'or', label='coscos coeff')
    semilogy(N_1, abs(c2), 'og',markersize=3, label='coscos bestfit')
legend(loc = 'upper right')
grid(True)
show()
if m == 0:                # using loglog
    loglog(N_1, abs(V_1), 'or', label='exponen coeff')
    loglog(N_1, abs(c1), 'og',markersize=3, label='exponen bestfit')
elif m == 1:
    loglog(N_1, abs(V_1), 'or', label='coscos coeff')
    loglog(N_1, abs(c2), 'og',markersize=3, label='coscos bestfit')
legend(loc = 'upper right')
grid(True)
show()
m += 1
```

a. The b coefficients are almost zero because coscos is an even function.
b.The first case coefficients do not decay as fast as the coefficients as in the second function as in the second function is not periodic so the coefficients go on and on. Whereas the first one is periodic and starts attaining the value of the periodic function at a sufficiently large number such as 25.
c.The semilog plot in Figure 4 looks linear as the coefficients are not spread

much and thus the x-axis isn't log axis and about the y-axis we need less coefficients to actually match the value(as seen in this example) so there is an exponential decay. Whereas the loglog plot in Figure 5 looks linear as coefficients are spread over a large range in the x-axis and the value in log axis thus decreases linearly for a corresponding decrease in the log axis since it is exponential.

Section - 6

Now we here do a least squares approach and with exponential and coscos function values at 400 places between 0 and 2π we evaluate the best fit coefficients. We first create a matrix with all zeroes of suitable size and pour the first columns with ones. And the rest with $\sin px$ and $\cos px$ alternatively with p ranging from 1 to 25. And the b vector being the corresponding value of function for the corresponding x. And we use leastsquares to get c.

```
 x = linspace(0, 2*pi, 401)
x = x[:-1]              # drop last term
if m == 0:
    b = exponen(x)
elif m == 1:
    b = coscos(x)
A = zeros((400, 51))            # creating A matrix
A[:, 0] = 1                     # 1st column 1
for k in range(1, 26):
    A[:, 2*k-1] = cos(k*x)
    A[:, 2*k] = sin(k*x)
if m == 0:
    c1 = lstsq(A, b, rcond=None)[0]         # best fits
if m == 1:
    c2 = lstsq(A, b, rcond=None)[0]
```

Section - 7

We correspondingly plot the c1 and c2 obtained in Figures 1,2,3,4 with green dots.

```
 b1 = dot(A, c1)                # fourier fn with bestfit coeff.
b2 = dot(A, c2)
x_scale = linspace(-2*math.pi, 4*math.pi, 100)
x_scale = array(x_scale)        # plotting actual
semilogy(x_scale, (exponen(x_scale)), '-r', label='Exponential')
x_scale_1 = linspace(0, 2*pi)
x_scale_1 = array(x_scale_1)            # plotting expected
semilogy(x_scale_1+2*pi, exponen(x_scale_1), '--b')
semilogy(x_scale_1-2*pi, exponen(x_scale_1), '--b')
semilogy(x_scale_1, exponen(x_scale_1), '--b', label='Exponen expect')
semilogy(x, b1, 'og', label='Exponen series')
legend(loc = 'upper right')
grid(True)
show()
plot(x_scale, coscos(x_scale), '-g',label='CosCos')
plot(x_scale, coscos(x_scale), '--b', label='CosCos expect')
plot(x, b2, 'or', label='Coscos series')
plot()
legend(loc = 'upper right')
grid(True)
show()
```

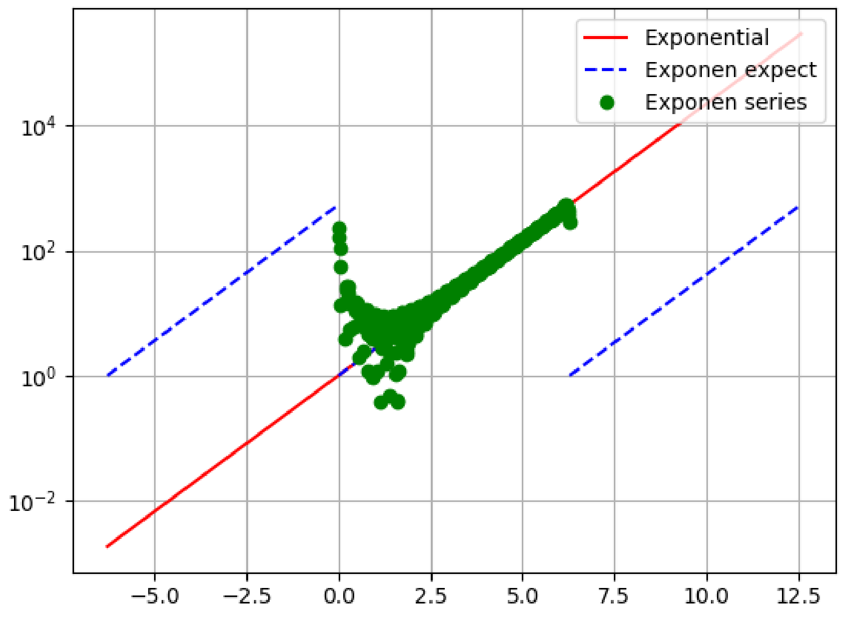

Figure 1: Exponen coeff, bestfit- semilog

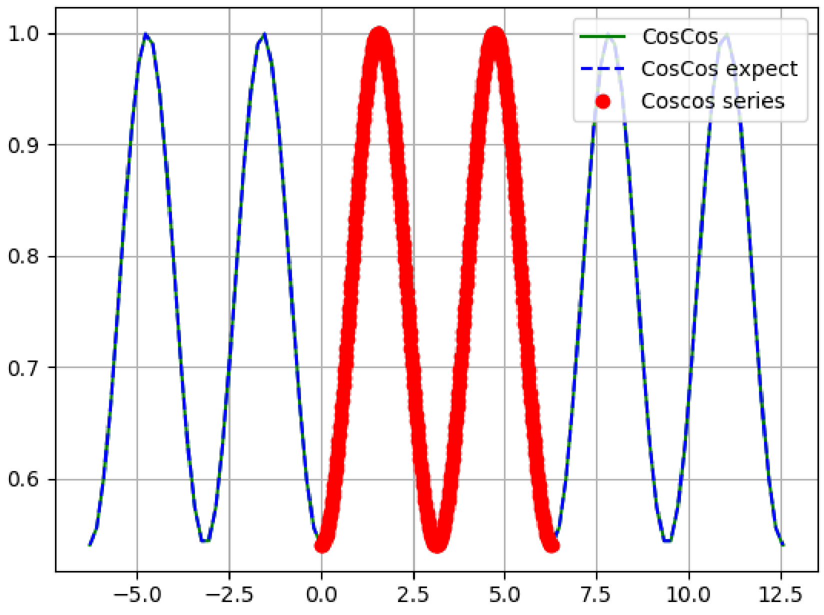

Figure 2: Exponen coeff, bestfit- loglog

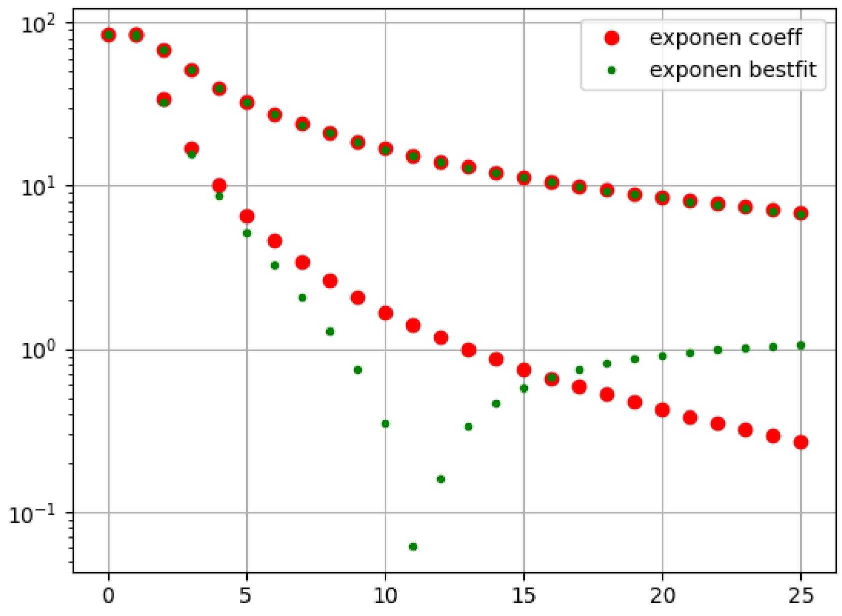

Figure 3: Coscos coeff, bestfit- semilog

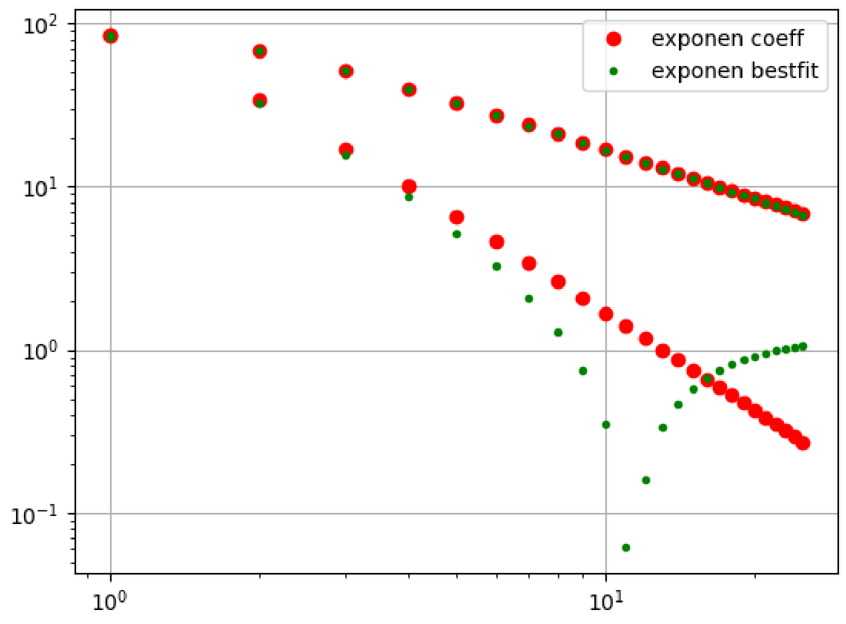

Figure 4: Coscos coeff, bestfit- loglog

Section - 8

We now compare both the points and get the maximum deviation for both the functions. The values as expected are not supposed to agree much but in the case of coscos it should be almost zero which is true.

```
     if m == 0:                     #       calculating deviation
        dev1 = max(abs(c_[c1]-V_1))
        print(f'The maximum deviation in the case of exponential is {dev1}')
    if m == 1:
        dev2 = max(abs(c_[c2]-V_1))
        print(f'The maximum deviation in the case of coscos is {dev2}')
```

The deviation in the case of coscos is 2.67467704e-15
The deviation in the case of exponential is 1.33273087

Section - 9

As seen from Section-7 code we calculated the function values from the best fit coefficients and correspondingly plotted them in Figure-1 and Figure-2 with green dots.

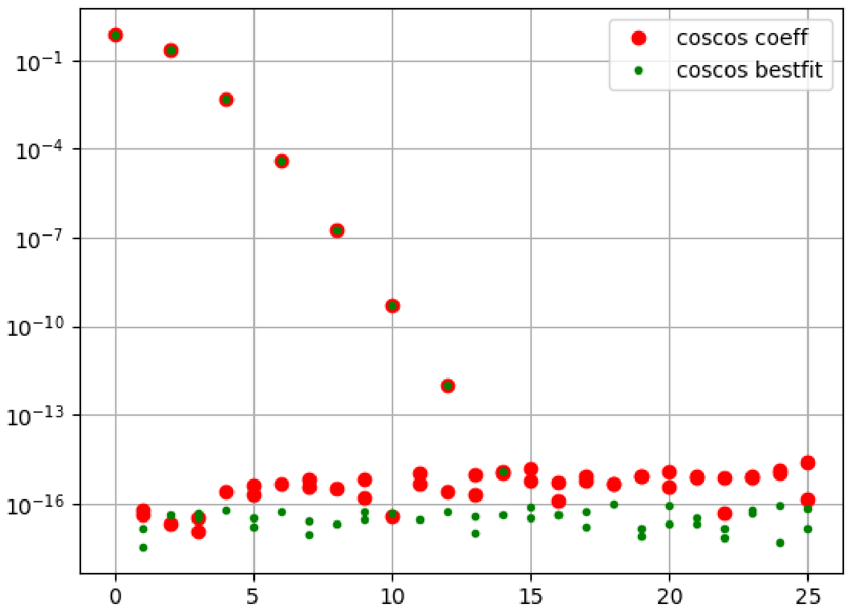

Figure 5: Exponen bestfit, coeff, expected

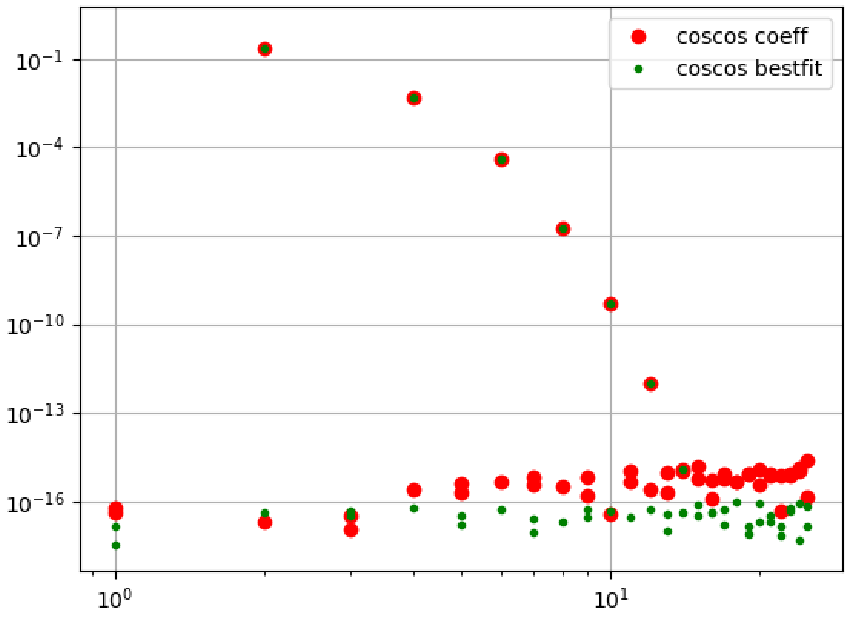

Figure 6: Coscos bestfit, coeff, expected

The deviation in figure 1 can be related to the non-decaying coefficients. As the coefficients decay fast in coscos case so we almost have other coefficients to be 0. And hence the bestfit gives coefficients such that the two graphs almost become one, which is not the case with exponential. Or we can say that since as the exponential function is discontinuous at the boundaries there will be a ringing in the output. If we consider more and more coefficients this ringing will be reduced.

Conclusion

In conclusion we learnt how to approximately or in some cases plot the fourier series coefficients in the graphs. It also has given us some insight into the fourier series and it's analysis.