EE2703: Applied Programming Lab
Assignment 4
Fourier aproximations

V. Ruban Vishnu Pandian
EE19B138

March 13, 2021

Contents

1 Aim:

The aim of this assignment is to:

- Plot functions e^x and cos(cos(x)) for x ϵ [-2π, 4π).
- Find first 51 fourier coefficients of both the functions using direct integration and least squares approximation.
  (Training data for least squares approximation: Function values for
  x ϵ [0, 2π))
- Plot function values obtained using both the set of coefficients for
  x ϵ [-2π, 4π).

2 Theory:

2.1 Functions: e^x and cos(cos(x)):

These two real functions are to be fitted using fourier series coefficients over the interval [0, 2π) as follows:

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

where the coefficients are given by the formulae:

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

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

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

2.2 Fourier coefficients vector:

The first 51 coefficients of the given functions are calculated using the integration formulae given above. The column vector is of the form:

$$\begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_{25} \\ b_{25} \end{pmatrix}$$

We also know from Eqn.(1):

$$a_0 + \sum_{n=1}^{25} a_n cos(nx) + \sum_{n=1}^{25} b_n sin(x) \approx f(x)$$

Hence, in matrix form, the equation would be:

$$\begin{pmatrix} 1 & cos(x_1) & sin(x_1) & ... & cos(25x_1) & sin(25x_1) \\ 1 & cos(x_2) & sin(x_2) & ... & cos(25x_2) & sin(25x_2) \\ ... & ... & ... & ... & ... & ... \\ 1 & cos(x_{400}) & sin(x_{400}) & ... & cos(25x_{400}) & sin(25x_{400}) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_{25} \\ b_{25} \end{pmatrix} \approx \begin{pmatrix} f(x_1) \\ f(x_2) \\ ... \\ f(x_{400}) \end{pmatrix}$$

In this way, by generating the matrix and the coefficients vector, the function values can be computed for a number of values of *'x'*.

2.3 Least squares approximation:

We know the integration formulae for the fourier coefficients and we used it to find the approximate function values in the previous section.
What we could also do is we can find a least squares solution for the coefficients vector by forming the matrix and by having the function values beforehand.

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

Here, the matrix is called *'A'*, the coefficients vector *'c'* and the values vector *'b'*. Once, we have matrix *'A'* and vector *'b'*, we can find the least squares fit for vector *'c'*. The values we get for the coefficients in this manner won't be the exact same values we obtained by the integration formulae. But these coefficient values minimise the least square error.

3 Procedure:

According to the assignment, the following objectives must be completed:

1. Functions e^{x}and cos(cos(x)) should be plotted with *'x'* ranging from the interval [-2π,4π). The expected fourier plots should also be plotted on the respective figures. The code, by default, considers 400 equally spaced points in this interval as the discrete values of *'x'*. However, it could also be passed on by the user as a command line argument. (First argument after the python file name).

   Two user-defined functions must also be created to return the value of e^x and cos(cos(x)) for an array *'x'*.

2. The first 51 fourier coefficients:

$$\begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_{25} \\ b_{25} \end{pmatrix}$$

   must be obtained by default in this vector form. However, the number of fourier coefficients needed to be computed can also be passed as a command line argument.
   (Second argument after the python file name. If the user wants the coefficients, say till a$_{50}$, b$_{50}$, he should enter 50 as the command line argument).

   These fourier coefficients are to be computed using the in-built integrator of python and with the help of two user functions:

$$u(x,k) = f(x)cos(kx)$$

$$v(x,k) = f(x)sin(kx)$$

   Python inbuilt function:

```
import scipy.integrate as sp
# Library that contains the inbuilt integrator

integ = sp.quad(u,0,2*np.pi,args=(k,func))
# 'Quad' function is the integrator function
```

3. The fourier coefficients obtained from integration formulae should be plotted using *semilog* and *loglog* plot functions of python with respect to their indices. Since, we need to plot the logarithm of the fourier coefficients with respect to their indices, logarithms of the absolute values of the fourier coefficients are plotted since the fourier coefficients can also be negative sometimes.

4. The fourier coefficients should again be calculated, but now using the least squares approach. The matrix equation is:

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

   with x ϵ [0, 2π) and the python function used to do this is the inbuilt least squares function. Again, the default number of samples for 'x' is 400 but it could also be passed as a command line argument (It is the same first argument after the python file name).

   Python inbuilt function:

```
import scipy
# Library that contains the inbuilt least squares function

coeff_vec_exp_lst = scipy.linalg.lstsq(mat_temp,expon(vectemp))[0]
# 'lstsq' function is the least squares function
```

5. The coefficients obtained by integration and by least squares approximation should be compared and the largest deviation should also be printed.

6. The function values should be found using fourier series equation:

$$a_0 + \sum_{n=1}^{25} a_n cos(nx) + \sum_{n=1}^{25} b_n sin(x) \approx f(x)$$

   The function values obtained by both the sets of coefficients (Integration and least squares approximation) are to be compared with the expected fourier plots.

Note: If you want to pass a non-default value to only one of the arguments in {Number of 'x' samples, Max. coefficient index}, pass the default value to the other argument as well. Passing only one different value will create ambuiguity and hence, the code will not work.

4 Code and functions:

The functions which are explained below are coded based on **“Modular programming”**. They have a level of heirarchy within them, that is, one function is called inside another function, which is called inside another and so on.

In this way, each operation is coded as a seperate module and they can be used individually as well (Instead of doing it all under one function call).

4.1 Functions *expon(x)* and *coscos(x)*:

```
def expon(a):
    return np.exp(a)

def coscos(a):
    return np.cos(np.cos(a))
```

Both these functions take an input array (or scalar) *’a’* and returns the corresponding function values as output array (or scalar).

For example, if the input array (a) is:

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

then the output array for *expon* function is (expon(a)):

$$\begin{pmatrix} e^1 \\ e^2 \\ e^3 \\ e^4 \end{pmatrix}$$

4.2 Functions *u(x,k,func)* and *v(x,k,func)*:

```
def u(x,k,func):
    val = func(x)*np.cos(k*x)
    return val

def v(x,k,func):
    val = func(x)*np.sin(k*x)
    return val
```

These two functions are the helper functions which are used to provide the integrand terms of the fourier integral. The arguments are:

- x = The scalar value of the dependent variable (For example, it is the x appearing in f(x)cos(5x))
- k = The coefficient of x in the sine/cosine term (For example, k=5 in the term f(x)cos(5x))
- func = The function expression 'f' (For example, if the integrand is e^xcos(5x) then func is the *expon* function)

4.3 Functions *f_coeff_cos(k,func)* and *f_coeff_sin(k,func)*:

```
def f_coeff_cos(k,func):
    integ = sp.quad(u,0,2*np.pi,args=(k,func))
    integ = integ[0]
    if k==0:
        return integ/(2*np.pi)
    else:
        return integ/np.pi

def f_coeff_sin(k,func):
    integ = sp.quad(v,0,2*np.pi,args=(k,func))
    integ = integ[0]
    return integ/np.pi
```

These functions do the integration process and return the fourier coefficients. The arguments are:

- k = The coefficient of x in sine/cosine term (For example, k=5 in f(x)cos(5x))
- func = The function whose fourier coefficient is calculated (For example, func is expon in e^xcos(5x))

4.4 Function *coeff_vector(num,func):*

```
def coeff_vector(num,func):
    vec = np.zeros((2*num+1,1))
    vec[0][0] = f_coeff_cos(0,func)
    for i in range(1,num+1):
        vec[2*i-1][0] = f_coeff_cos(i,func)
        vec[2*i][0] = f_coeff_sin(i,func)
    return vec
```

This function calculates the first (2n+1) fourier coefficients and returns it as a column vector. The arguments are:

- num = The index till which the fourier coefficients are to be computed (For example, if the fourier coefficients needed are a_0,a_1,b_1,...,a_{50},b_{50}then num=50)
- func = The function whose fourier coefficient is calculated (For example, func is expon in e^xcos(5x))

4.5 Function *matrix_ gen(num,x):*

```
def matrix_gen(num,x):
    n = x.shape[0]
    mat = np.ones((n,1))
    for i in range(1,num+1):
        mat = np.c_[mat,np.cos(x*i)]
        mat = np.c_[mat,np.sin(x*i)]
    return mat
```

This function takes a column vector *'x'* and returns this matrix:

$$\begin{pmatrix} 1 & cos(x_1) & sin(x_1) & ... & cos(px_1) & sin(px_1) \\ 1 & cos(x_2) & sin(x_2) & ... & cos(px_2) & sin(px_2) \\ ... & ... & ... & ... & ... & ... \\ 1 & cos(x_n) & sin(x_n) & ... & cos(px_n) & sin(px_n) \end{pmatrix}$$

where *'p'* is the maximum index of the coefficient the user desires for. The arguments are:

- num = p (or) the maximum index of the coefficient
- x = The column vector containing discrete values of independent variable *'x'*

4.6 Function *fourier_func(num,x,func):*

```
def fourier_func(num,x,func):
    vec = coeff_vector(num,func)
    mat = matrix_gen(num,x)
    func_vals = np.dot(mat,vec)
    return func_vals
```

This function takes a colum vector *'x'*, maximum index *'num'* and function name *'func'* as input arguments, generates the coefficient vector using the *coeff_ vector(num,func)* function call, matrix using the *matrix_ gen(num,x)* function call and multiplies the matrix with the coefficient vector to generate the function values vector. In short it carries out the matrix equation given below:

$$\begin{pmatrix} 1 & cos(x_1) & sin(x_1) & ... & cos(px_1) & sin(px_1) \\ 1 & cos(x_2) & sin(x_2) & ... & cos(px_2) & sin(px_2) \\ ... & ... & ... & ... & ... & ... \\ 1 & cos(x_n) & sin(x_n) & ... & cos(px_n) & sin(px_n) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_p \\ b_p \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ ... \\ f(x_n) \end{pmatrix}$$

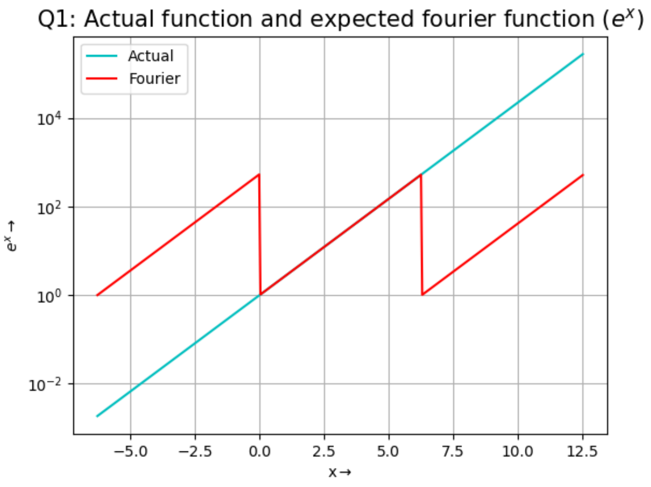
The input arguments are:

- num = p (or) the maximum index of the coefficient
- x = The column vector containing discrete values of independent variable *'x'*
- func = The function whose fourier coefficient is calculated (For example, func is expon in e^xcos(5x))

5 Observations and plots:

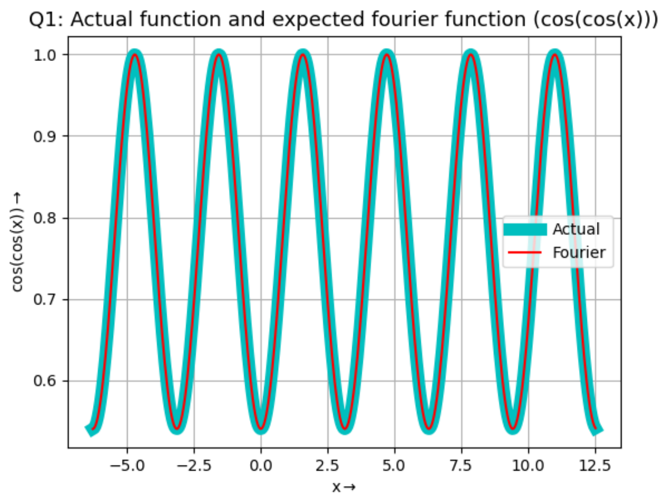
5.1 Exact vs Expected fourier plots:

5.1.1 Figure 1: For e^x function:



Since the fourier coefficients are calculated over an interval of $[0, 2\pi)$,the function generated by fourier coefficients is also periodic with period 2π

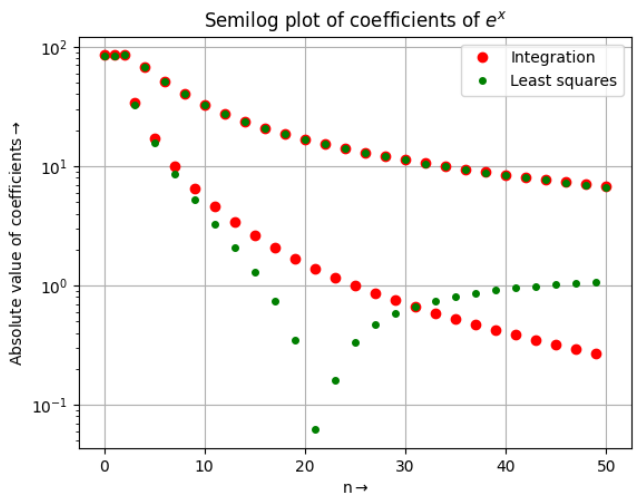
5.1.2 Figure 2: For cos(cos(x)) function:



cos(cos(x)) is already a periodic function with period of π. Hence, the expected fourier plot is exactly overlapping with the exact function plot

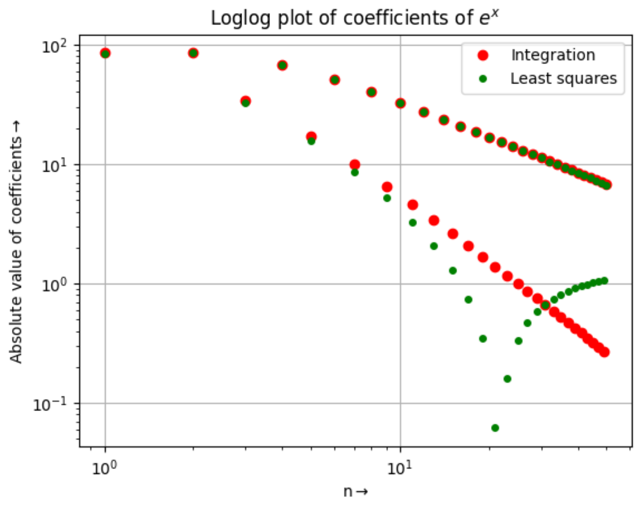
5.2 Fourier coefficients obtained by integration vs least squares approximation:

5.2.1 Figure 3: Semi-logy plot for coefficients of e^x:



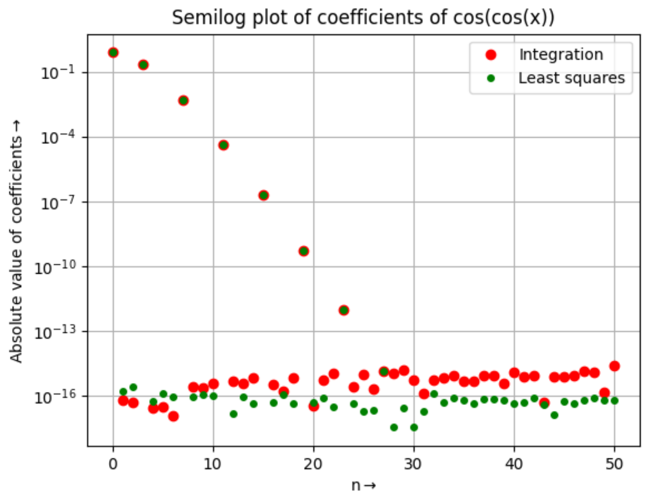
For function e^x, the coefficient magnitudes seem to decrease with respect to *'n'* in the semilog plot

5.2.2 Figure 4: Loglog plot for coefficients of e^x:



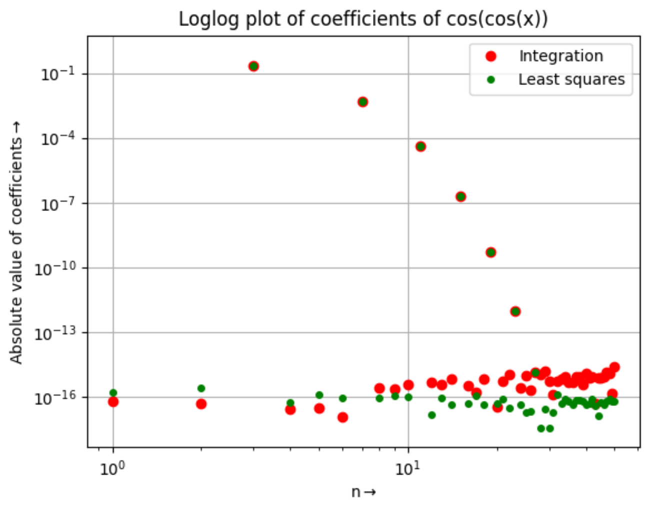
For function e^x, the coefficient magnitudes seem to decrease linearly with respect to *'n'* in the loglog plot

5.2.3 Figure 5: Semi-logy plot for coefficients of cos(cos(x)):



For function cos(cos(x)), the cosine coefficient magnitudes seem to decrease linearly with respect to *'n'* in the semilog plot

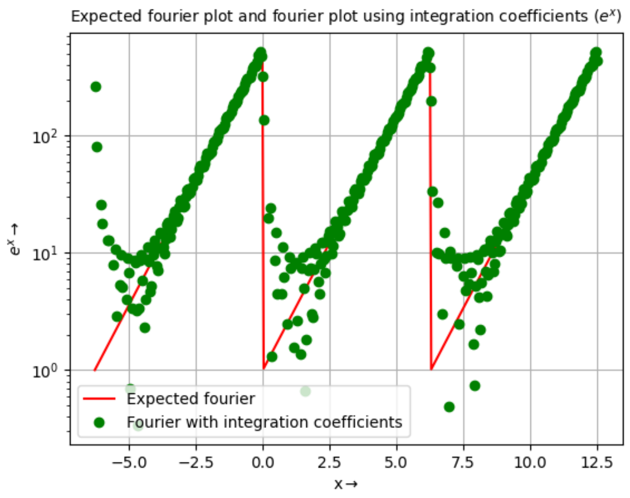
5.2.4 Figure 6: Loglog plot for coefficients of cos(cos(x)):



For function cos(cos(x)), the cosine coefficient magnitudes seem to decrease with respect to *'n'* in the loglog plot

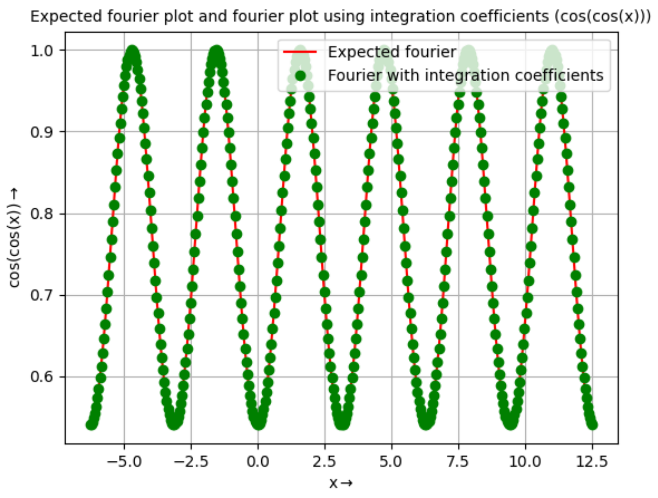
5.3 Expected fourier plot along with fourier plot obtained using integration coefficients:

5.3.1 Figure 7: For e^x function:



The function plot generated using fourier coefficients (Integration method) almost match the expected fourier plot, except at the discontinuous points (e^x)

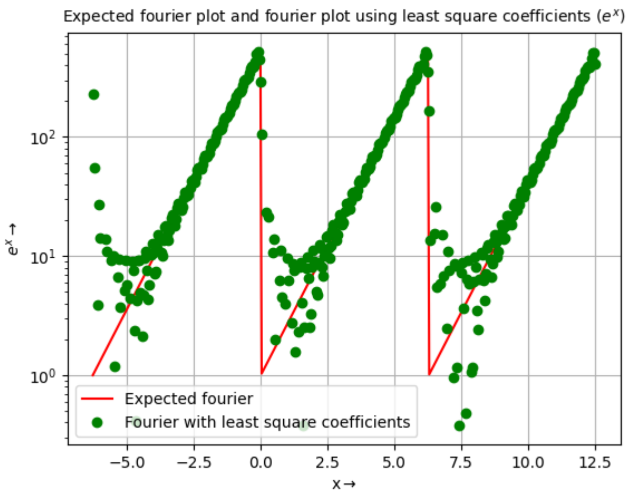
5.3.2 Figure 8: For cos(cos(x)) function:



The function plot generated using fourier coefficients (Integration method) matches the expected fourier plot seemingly flawlessly (cos(cos(x)))

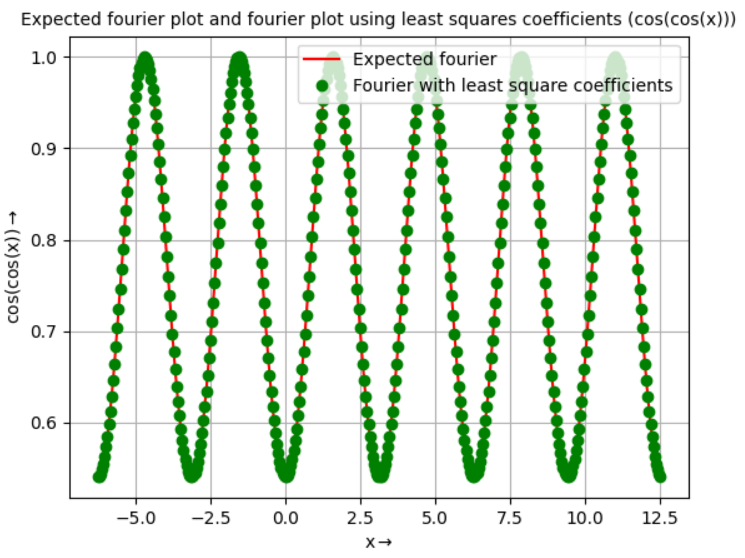
5.4 Expected fourier plot along with fourier plot obtained using least square coefficients coefficients:

5.4.1 Figure 9: For e^x function:



The function plot generated using fourier coefficients (Least squares method) almost match the expected fourier plot, except at the discontinuous points (e^x)

5.4.2 Figure 10: For cos(cos(x)) function:



The function plot generated using fourier coefficients (Least squares method) matches the expected fourier plot seemingly flawlessly (cos(cos(x)))

6 Results and conclusions:

6.1 Conclusions for question (3):

(a) We know

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

for function cos(cos(x)). Let us make the variable change $x = t - \pi$. Hence, we have:

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} cos(cos(t-\pi))sin(nt-n\pi)dt$$

$cos(cos(t-\pi)) = cos(-cos(t)) = cos(cos(t))$,
$sin(nt-n\pi) = sin(nt)cos(n\pi) - cos(nt)sin(n\pi) = (-1)^n sin(nt)$.
Clearly our new integral is:

$$b_n = \frac{(-1)^n}{\pi}\int_{-\pi}^{\pi} cos(cos(t))sin(nt)dt$$

Here, $cos(cos(t))$ is an even function whereas $sin(nt)$ is an odd function. The integral interval is also a symmetric interval $(-\pi, \pi)$. Hence, theoretically, the coefficient b_n is zero for all values of *'n'.* However, because of computer precision, the values are not exactly zero but very low (in the orders of 1e-17). Hence, they decay faster when compared to the $a_n$values.

(b) For e^x :

$$a_0 = \frac{e^{2\pi} - 1}{2\pi}$$

$$a_n = \frac{e^{2\pi} - 1}{\pi(1 + n^2)}$$

$$b_n = \frac{-n(e^{2\pi} - 1)}{\pi(1 + n^2)}$$

For cos(cos(x)), the exact formulae for the coefficients aren't known. So let us use the taylor series approximation:

$$cos(cos(x)) = 1 - \frac{cos^2 x}{2!} + \frac{cos^4 x}{4!} - \frac{cos^6 x}{6!} + ..., |cos(x)| < 1$$

Also we know that the only cosine coefficients will be present in a $cos^{2n}x$ term and it will be limited to a_{2n}. Hence, if we approximate cos(cos(x)) to a summation of $cos^{2n}x$ terms till, let's say n=10, the maximum index which the cosine coefficient can have will be 10. All other higher index terms will be zero.

Hence, we can see that the fourier coefficients of cos(cos(x)) decrease faster with respect to their index than the coefficients of e^x do.

(c) From part (b), we saw:

$$a_n = \frac{e^{2\pi} - 1}{\pi(1 + n^2)}$$

$$b_n = \frac{-n(e^{2\pi} - 1)}{\pi(1 + n^2)}$$

Hence, for large enough values of 'n', we can safely assume that the denominator is approximately n^2. Hence, we have:

$$log(|a_n|) = log(\frac{e^{2\pi} - 1}{\pi}) - 2log(n)$$

$$log(|b_n|) = log(\frac{e^{2\pi} - 1}{\pi}) - log(n)$$

That's why we have an almost linear plot in figure 4, which is the loglog plot of fourier coefficients of e^x.

The coefficients of cos(cos(x)) converge to an exponentially decaying function. That's why we have a linear plot in figure 5, which is the semi-logy plot of fourier coefficients of cos(cos(x)).

6.2 Conclusions for question (6):

The fourier coefficients obtained by integration formula are exact and the infinite terms summation:

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

gives the exact result (provided it converges). The least square solution, on the other hand, minimises the error between the norm of the difference of these two vectors:

$$\begin{pmatrix} 1 & cos(x_1) & sin(x_1) & ... & cos(px_1) & sin(px_1) \\ 1 & cos(x_2) & sin(x_2) & ... & cos(px_2) & sin(px_2) \\ ... & ... & ... & ... & ... & ... \\ 1 & cos(x_n) & sin(x_n) & ... & cos(px_n) & sin(px_n) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_p \\ b_p \end{pmatrix} and \begin{pmatrix} f(x_1) \\ f(x_2) \\ ... \\ f(x_n) \end{pmatrix}$$

Hence, naturally the integration coefficients and least square coefficients need not be exactly same and they won't be. Moreover, the least square solution depends on 'p' and 'n' as well but the fourier coefficients are fixed and depend only on the function. Hence, they won't be the same. However, they will be almost same with a few considerable deviations. And they'll also give the approximate function values with a really good accuracy which is evident from figures 7,8,9 and 10.

For p=400. n=25 (Default values given in the assignment), we have:

Maximum deviation in fourier coefficients (integration and least squares) of e^x = 1.332731
Maximum deviation in fourier coefficients (integration and least squares) of cos(cos(x)) = 2.656647e-15

Note: The maximum deviation in the absolute differences of the coefficients is found using this python code snippet:

```
vectemp = np.linspace(0,2*np.pi,N)
vectemp = vectemp[:-1]; vectemp.shape = (N-1,1)

mat_temp = matrix_gen(num,vectemp)
coeff_vec_exp_integ = coeff_vector(num,expon)
coeff_vec_coscos_integ = coeff_vector(num,coscos)
coeff_vec_exp_lst = scipy.linalg.lstsq(mat_temp,expon(vectemp))[0]
coeff_vec_coscos_lst = scipy.linalg.lstsq(mat_temp,coscos(vectemp))[0]

coeff_error_exp = np.abs(coeff_vec_exp_integ-coeff_vec_exp_lst)
coeff_error_coscos = np.abs(coeff_vec_coscos_integ-coeff_vec_coscos_lst)
```

6.3 Conclusions for question (7):

In the figures 8 and 10 (Plot of expected fourier for cos(cos(x)) vs plot obtained from fourier coefficients), there is no much deviation and the green circles almost lie on the expected fourier plot. However, we have lot of deviations from the expected fourier plot in figures 7 and 9 (Plot of expected fourier for e^x vs plot obtained from fourier coefficients). Especially, the green circles are randomly placed near the points where there is a jump discontinuity.

This happens because of Gibbs phenomenon:

In mathematics, the Gibbs phenomenon, discovered by Henry Wilbraham (1848) and rediscovered by J. Willard Gibbs (1899), is the peculiar manner in which the Fourier series of a piecewise continuously differentiable periodic function behaves at a jump discontinuity. The nth partial sum of the Fourier series has large oscillations near the jump, which might increase the maximum of the partial sum above that of the function itself. The overshoot does not die out as n increases, but approaches a finite limit. This sort of behavior was also observed by experimental physicists, but was believed to be due to imperfections in the measuring apparatus. (Excerpt from Wikipedia)