EE2703: Assignment 4

Tanay Dixit
EE19B123

March 10, 2021

Introduction

In this assignment we study the Fourier series approximations of e^x and $cos(cos(x))$.We first compute the Fourier series coefficients using the expression of sum of scaled harmonics. The coefficients of the harmonics are computed using numerical integration provided by $quad()$ function in scipy. We then study another method for calculating Fourier coefficients using least square estimation.We further study the outcomes of the two mentioned methods and analyse the results and cause of discrepancy.

Functions

```
def exponential(x):
    return np.exp(x)
```

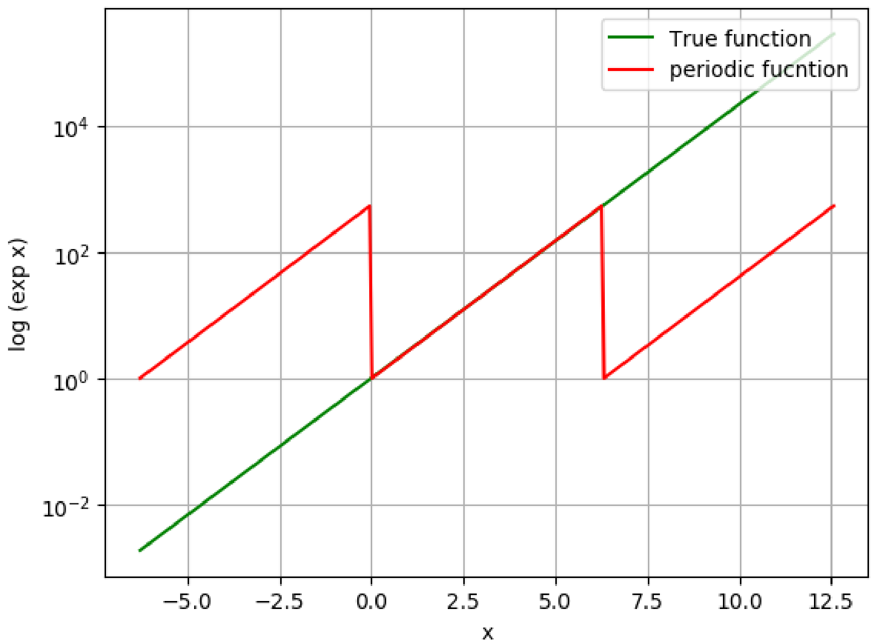


Figure 1: e^x

Please note that e^x is a non periodic function and Fourier series exists only for periodic functions. Hence we have considered a variation of e^x with period 2π that has the actual value of e^x only in the range $[0,2\pi)$.

```python
def coscos(x):
    return np.cos(np.cos(x))
```

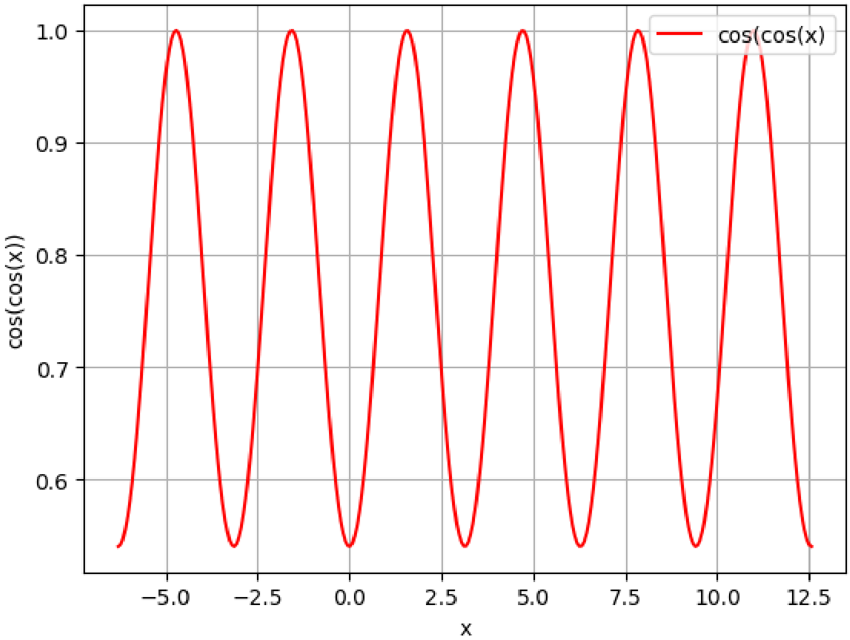


Figure 2: cos(cos(x))

Clearly one can see cos(cos(x)) is periodic while e^x is not

Fourier series coefficients

The fourier series of a function is calculated using the following

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

where

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(x)dx$$
$$a_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)\cos(nx)dx$$
$$b_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)\sin(nx)dx$$

Generating Fourier Coefficients

The first 51 coefficients of the fourier series are generated using the *scipy.integrate.quad*() using the equations mentioned in Equation 1.

```
def fourier_coeff(n,func):
    '''
    n : is the number of fourier series coefficients wanted
    func : is the function whose fourier coefficients are to be found

    return vector (a0,a1,b1, ....a25,b25)
    '''
    coef = np.empty(n)
    u = lambda x,k: func(x)*np.cos(k*x)
    v = lambda x,k: func(x)*np.sin(k*x)
    #a_0
    coef[0]= quad(func,0,2*np.pi)[0]/(2*np.pi)
    #a_n
    for i in range(1,n,2):
        coef[i] = quad(u,0,2*np.pi,args=((i+1)/2))[0]/np.pi
    #b_n
    for i in range(2,n,2):
        coef[i] = quad(v,0,2*np.pi,args=(i/2))[0]/np.pi
    return coef
```

The coefficients returned is a vector $\left(a_0, a_1, b_1, ...a_{25}, b_{25}\right)$

Plotting Fourier Coefficients

We plot the first 51 fourier coefficients on different scales mainly semilog and loglog in order to analyse decay of coefficients.

We use a custom defined function for plotting all plots as all have same theme , but different features(*Note: This same function is used in all plots each having different arguments*)

```

def plotdata(x,y1,y2 =[] ,xname= None, yname = None ,icon = 'ro',label = 'plot',
   path  =\ None, plottype = None, clear  = True):
    '''
    Params:
    x,y1,y2 are the data columns to be plotted ,y2 (optional)
    xname, yname are name to be given to axis, (optional)
    icon is marker icon (default is red dots)
    label : name given to graph (default : plot)
    path : place to save the image (required)
    plottype: The type pf plot to be use ie: semilogy, loglog, plot (required)
    '''
```

```
    if len(y2) >0:
        for i,y in enumerate([y1,y2]):
            assert len(x) ==len(y), "fucntions not same length"
            if plottype =='semilogy':
                plt.semilogy(x,y,icon[i] ,label =label[i], ms =4)
            elif plottype =='loglog':
                plt.loglog(x,y,icon[i] ,label =label[i],ms =4)
            else:
                plt.plot(x,y,'r-', label = label[i])
    else:
        assert len(x) ==len(y1), "fucntions not same length"
        if plottype =='semilogy':
            plt.semilogy(x,y1,icon ,label =label, ms =4)
        elif plottype =='loglog':
            plt.loglog(x,y1,icon ,label =label,ms =4)
        else:
            plt.plot(x,y1,'r-', label = label)

    plt.grid(True)
    plt.legend(loc ='upper right')
    plt.xlabel(xname)
    plt.ylabel(yname)
    if path !=None:
        plt.savefig(path+'.png',bbox_inches='tight')
        print("file saved at {}".format(path+'.png'))
    if clear:
        plt.clf()
```

The plots for coefficients of e^x and cos(cos(x)) are as follows
Note: We plot absolute values of coefficients as we are taking log

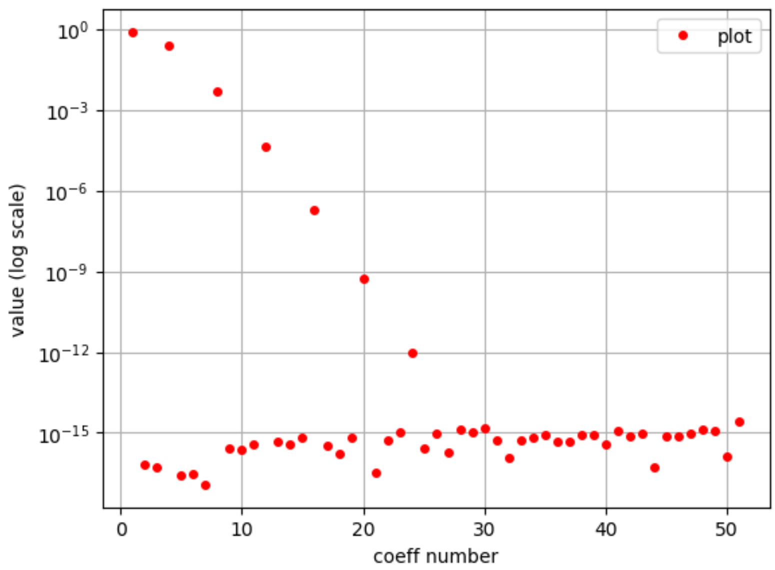


Figure 3: Fourier Coefficients for cos(cos(x)) in semilogy scale

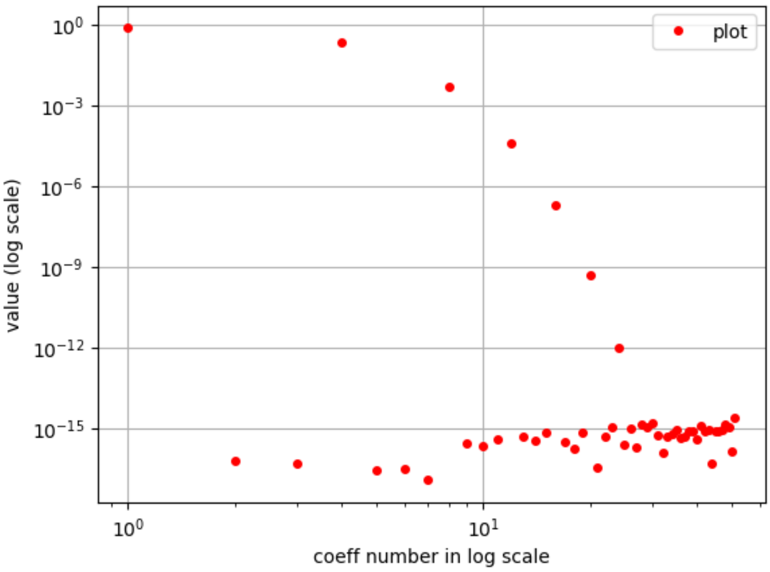


Figure 4: Fourier Coefficients for cos(cos(x)) in loglog scale

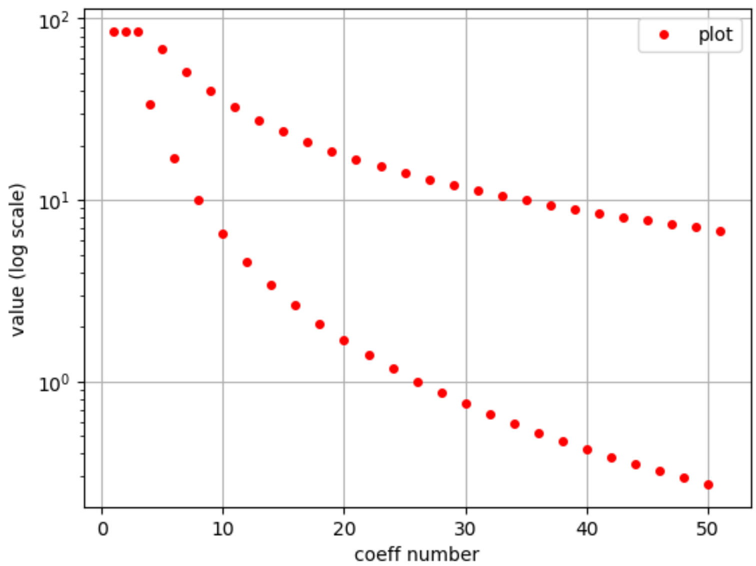


Figure 5: Fourier Coefficients for e^x in semilogy scale

Analysis of coefficients

- The b_n coefficients correspond to the odd harmonics and as $\cos\cos(x)$ is a even function,it does not contain odd harmonics and hence one can see b_n coefficients are almost 0 in Figure 3 and Figure 4 *(the small discrepancy is cause of numerical accuracy issues and approximation of* π*)*

- We can observe that $cos(cos(x))$ is a function which whose period is π and is continuous/smooth hence most of the contribution to the harmonics will be from lower order terms , hence the quick decay of coefficients in Figure 3 and Figure 4. While for e^x the decay of coefficients is slow because there is an exponential increase in gradient and has discontinuities which causes a wide range of harmonics to be present Figure 5.

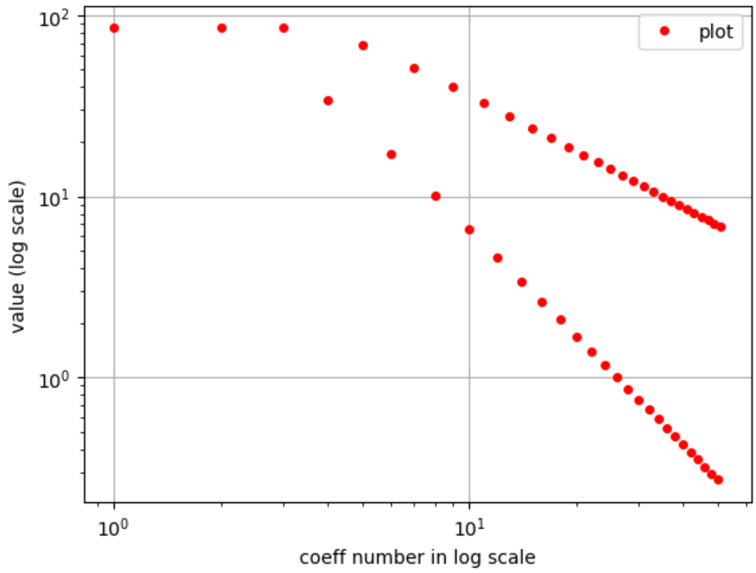


Figure 6: Fourier Coefficients for e^x in loglog scale

- One can also see form Equation 1 that the coefficients of e^x depend on n as

$$a_n \propto \frac{1}{n^2} \quad b_n \propto \frac{1}{n}$$

hence..

$$log(a_n),\ log(b_n) \propto log(n)$$

hence in **loglog** plot of e^x,one can see linear relationship in Figure 6

While for $cos(cos(x)$ the coefficients depend exponentially on n

$$a_n \propto e^{-n}$$

hence in semilog scale

$$a_n \propto log(e^{-n}) \approx n$$

hence in **semilog** plot of $\cos\cos x$ one can see a linear relationship in Figure 3

Least Squares Approach

We now obtain the fourier coefficients using another method ie: Least square approach. In this method we solve the following equation by using least square estimation

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

Where A is the left most matrix and b is the RHS. We want to solve c in $Ac = b$ where c are the Fourier coefficients.

The code below resembles the above equation

```python
def leastSquareCoef(func):
    '''
    fucntion used to compute the A , b matrixes for Least Sqaure Estimate case
    '''
    A = np.empty((400,51))
    x = np.linspace(0,2*np.pi, 401)
    x = x[:-1]
    A[:,0] =1
    for i in range(1,26):
        A[:,2*i-1] = np.cos(i*x)
        A[:,2*i] = np.sin(i*x)
    b = func(x)
    return A, b
```

Visualizing output of the Least Squares Approach

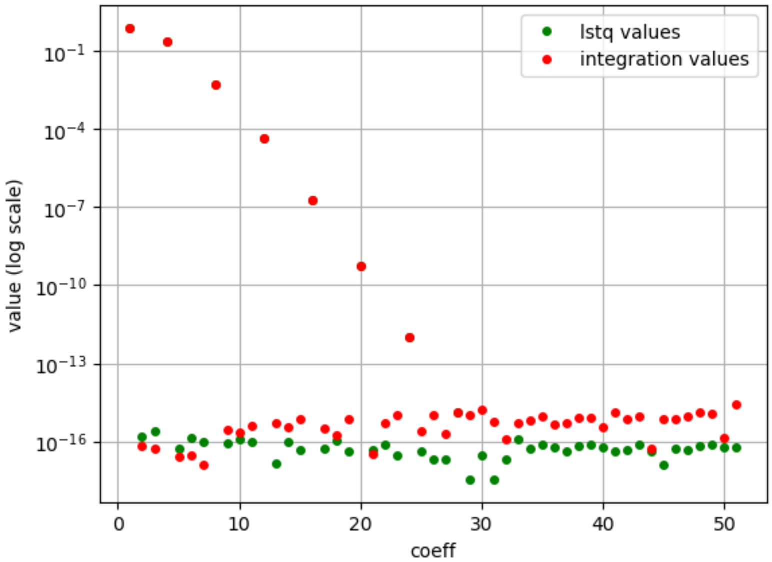


Figure 7: Fourier Coefficients for cos(cos(x)) in semilogy scale

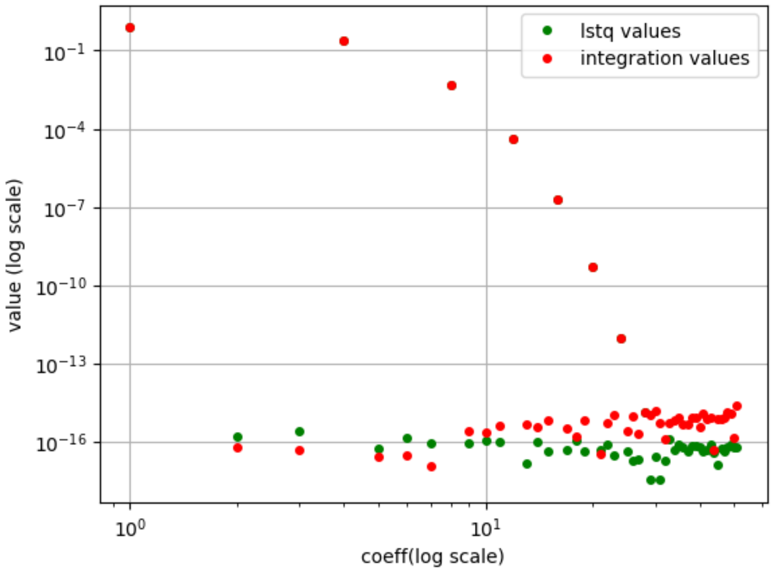


Figure 8: Fourier Coefficients for cos(cos(x)) in loglog scale

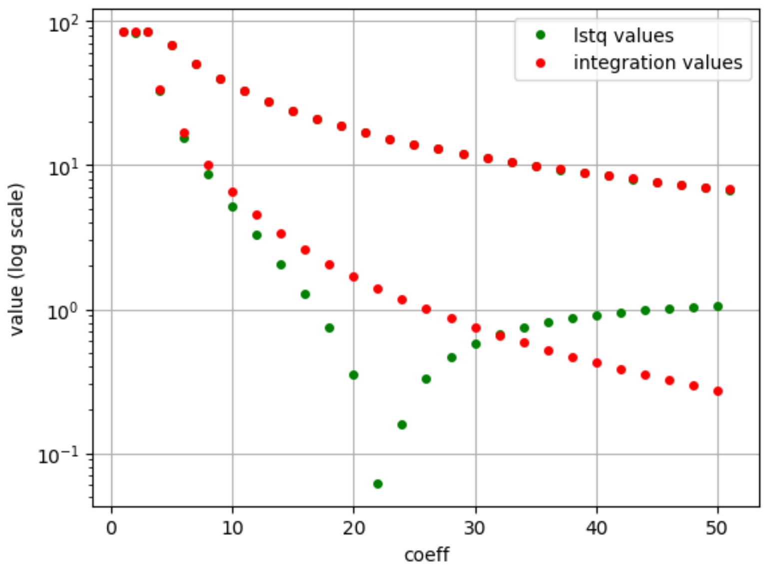


Figure 9: Fourier Coefficients for e^x in semilogy scale

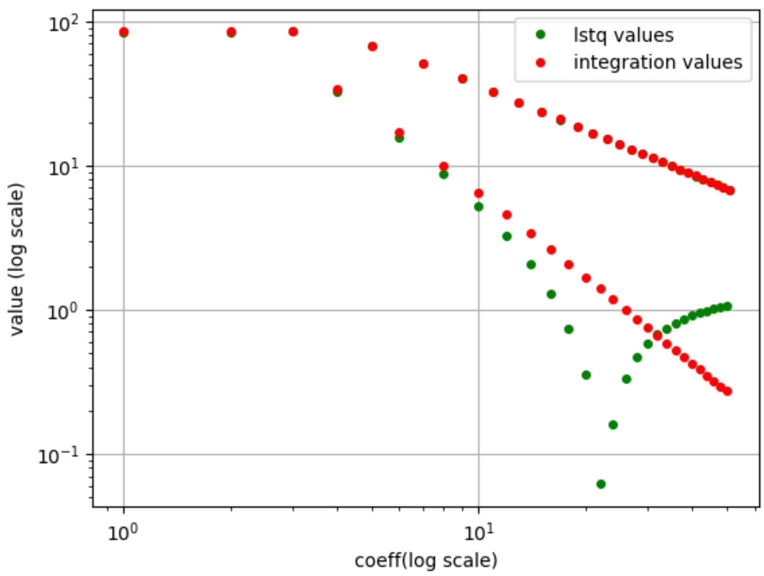


Figure 10: Fourier Coefficients for e^x in loglog scale

Comparing the two approaches

From the above plots we can se that the coefficients more or less agree in the case of cos(cos(x)) while in the case of e^x they significantly disagree.

This mainly occurs because in numerical integration method it estimates the discontinuity points better, while least square estimate has only 400 points of information.Since e^x contains several higher order frequencies we will need many more data points to get a good estimate from least squares method.
We support our claim with the following numbers given below

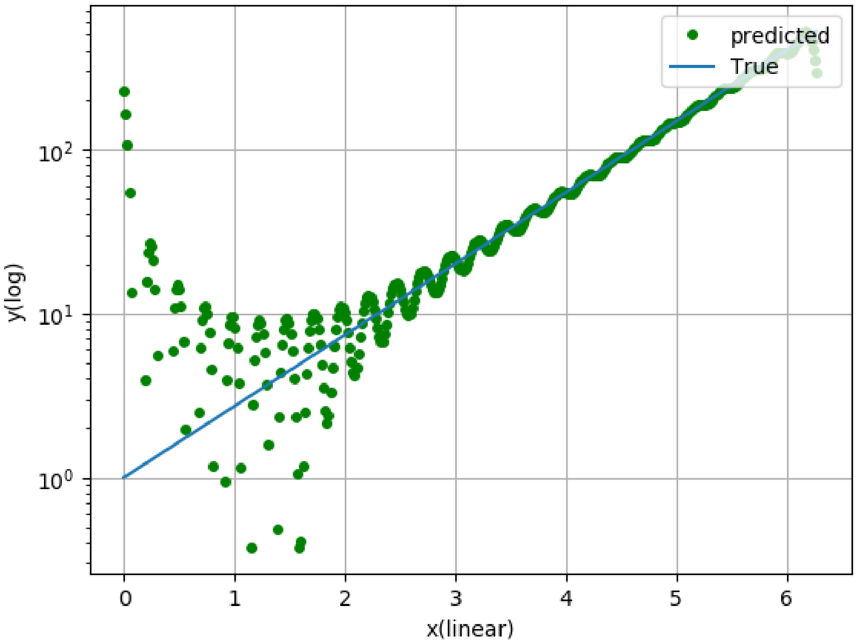
```
def compare(coef1, coef2):
    '''
    computes the maximum deviation between the 2 given coefficients vectors
    '''
    dev = np.abs(coef1 - coef2)
    max_dev = np.max(dev)
    return max_dev
```

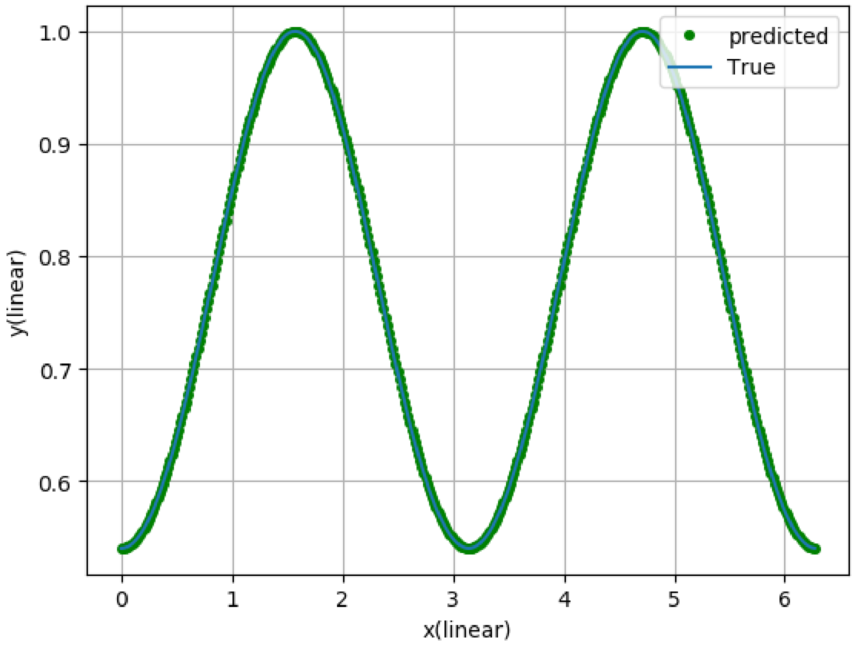
The maximum absolute error in estimation of coefficients for e^x =1.33
Whereas the maximum absolute error in estimation of coefficients for cos(cos(x)) = 2.65e-15
Our Predictions for e^x are poor compared to that of cos(cos(x)) as e^x has more frequency terms and it requires a higher sampling rate than 400.
For example if we sample around 1e4 points then the errors are :
The maximum absolute error for e^x =0.00674
Whereas for cos(cos(x)) = 2.65e-15
This decrease in error is at a very large computational cost as complexity is O(n^2), as a result its not a good method.

Plotting Results

```
predicted_cos =  np.matmul(Acos,Lcoef_cos)
predicted_exp  = np.matmul(Aexp, Lcoef_exp)

xcoord = np.linspace(0,2*np.pi, 401)[:-1]

plt.plot(xcoord, predicted_cos,'go',ms =4, label = 'predicted', )
plt.plot(xcoord, coscos(xcoord), label = 'True')
plt.grid(True)
plt.xlabel('x(linear)')
plt.ylabel('y(linear)')
plt.legend(loc = 'upper right')
plt.savefig('imgs/Figure1.png',bbox_inches='tight')
plt.clf()

plt.semilogy(xcoord, predicted_exp,'go',ms =4, label = 'predicted', )
```

```
ycoord = np.tile(exponential(xcoord), 3)
plt.semilogy(xcoord,exponential(xcoord), label = 'True')
plt.xlabel('x(linear)')
plt.ylabel('y(log)')
plt.grid(True)
plt.legend(loc = 'upper right')
plt.savefig('imgs/Figure0.png',bbox_inches='tight')
plt.clf()
```





Conclusion

We observe that the fourier estimation of e^x deviates significantly with the function value at regions close to 0, but agrees almost perfectly in the case of $\cos(\cos(x))$. This happens because of 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, which means that the partial sums obtained using the fourier coefficients converge at different rates for different values of x.

This difference in the rates of convergence leads to the property of Gibb's phenomenon, which is the observed at discontinuities in the fourier estimation of a discontinuous function. This oscillation is present for any finite N, but as $N \to \infty$ the series begins to converge. This explains the discrepancy in the fourier approximation for e^x.