Fourier Approximations
Assignment 3

Rohithram R, EE16B031
B.Tech Electrical Engineering, IIT Madras

February 18, 2018
First created on February 13,2018

Abstract

This report presents a study of different methods of creating a $tan^{-1}(x)$ from its integral definition of $\int_0^x dx/(1+t^2)$ using scipy's quad function to integrate and other method is of numerical integration using trapezoidal rule,which can be used even for non-integrable functions to integrate it. And it also discusses the advantage of Vectorization of code compared to for loops and also on finding estimate errors when actual error is unknown i.e function is non-integrable by halving the stepsize till it reaches certain tolerance!.

1 Introduction

This report discusses 7 tasks in python to find Fourier Approximations of two function e^x and $\cos(\cos(x))$ from its integral definition and using Least Squares method.
We will fit two functions, e^x and $\cos(\cos(x))$ over the interval [0;2π) using the fourier series

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

The equations used here to find the Fourier coefficients are as follows:

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

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(nx)dx \tag{3}$$

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(nx)dx \tag{4}$$

2 Python code

```
In [1]: # load libraries and set plot parameters
        from pylab import *
        from scipy.integrate import quad
        %matplotlib inline

        from IPython.display import set_matplotlib_formats
        set_matplotlib_formats('pdf', 'png')
        plt.rcParams['savefig.dpi'] = 75
        plt.rcParams['figure.autolayout'] = False
        plt.rcParams['figure.figsize'] = 10, 6
        plt.rcParams['axes.labelsize'] = 18
        plt.rcParams['axes.titlesize'] = 20
        plt.rcParams['font.size'] = 16
        plt.rcParams['lines.linewidth'] = 2.0
        plt.rcParams['lines.markersize'] = 4
        plt.rcParams['legend.fontsize'] = 14
        plt.rcParams['legend.numpoints'] = 2
        plt.rcParams['legend.loc'] = 'best'
        plt.rcParams['legend.fancybox'] = True
        plt.rcParams['legend.shadow'] = True
        plt.rcParams['text.usetex'] = True
        plt.rcParams['font.family'] = "serif"
        plt.rcParams['font.serif'] = "cm"
        plt.rcParams['text.latex.preamble'] = r"\usepackage{subdepth}, \usepackage{type1cm}"
```

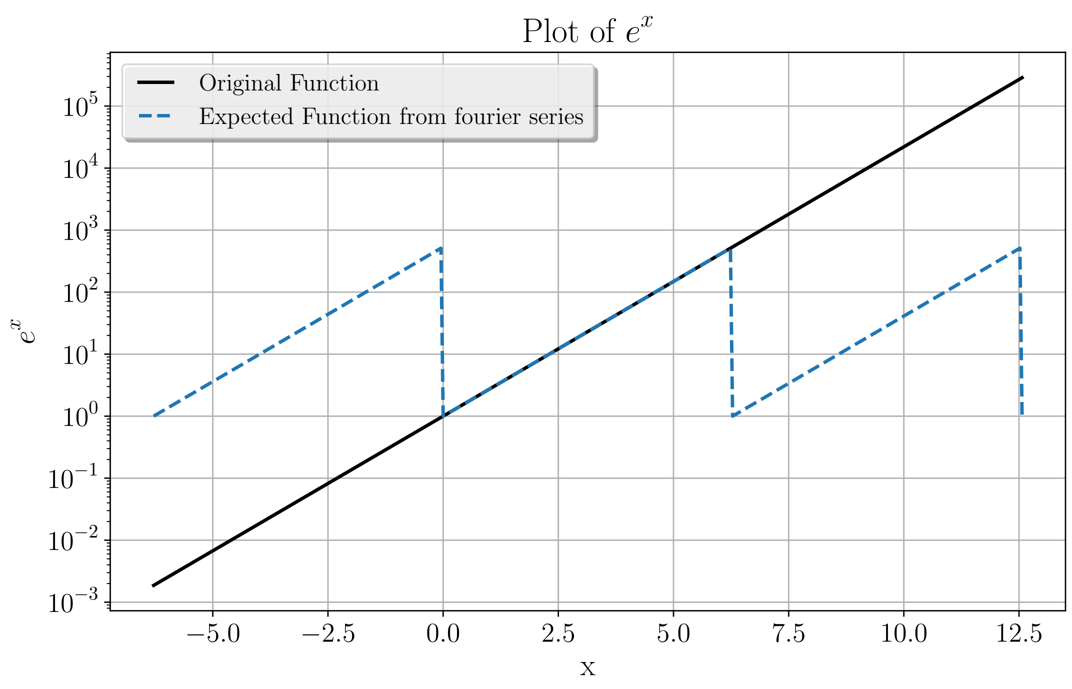
2.1 Question 1

- 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

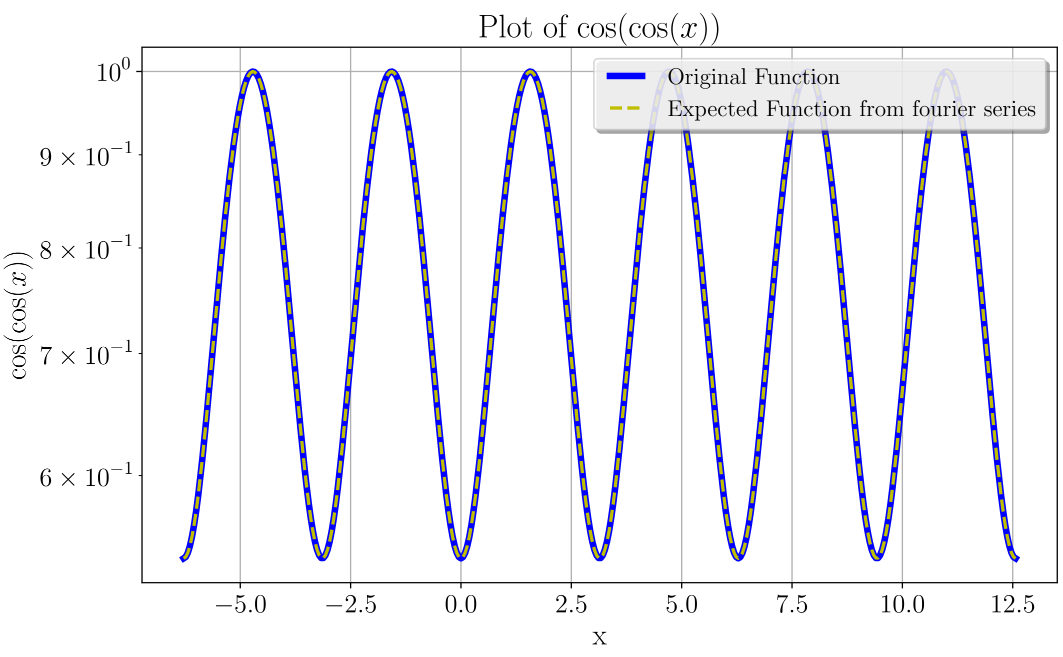
```
In [2]: #Functions for $e^{x}$ and $\cos(\cos(x))$ is defined
        def fexp(x):
            return exp(x)

        def fcoscos(x):
            return cos(cos(x))
```

```
In [3]: x = linspace(-2*pi, 4*pi,400)
        #Period of function created using fourier coefficients will be 2pi
        period = 2*pi
        exp_fn = fexp(x)                #finding exp(x) for all values in x vector
        cos_fn = fcoscos(x)             #finding cos(cos(x)) for all values in x vector
```

```
In [4]: #Plotting original function vs expected function for exp(x)
        fig1 = figure()
        ax1 = fig1.add_subplot(111)
        ax1.semilogy(x,exp_fn,'k',label="Original Function")
        #plotting expected function by dividing the x by period and giving remainder as
        #input to the function, so that x values repeat after given period.
        ax1.semilogy(x,fexp(x%period),'--',label="Expected Function from fourier series")
        ax1.legend()
        title("Plot of $e^{x}$")
        xlabel("x")
        ylabel("$e^{x}$")
        grid()
        savefig("Figure1.jpg")
```



```
In [5]: #Plotting original function vs expected function for cos(cos((x)))
        fig2 = figure()
        ax2 = fig2.add_subplot(111)
        ax2.plot(x,cos_fn,'b',linewidth=4,label="Original Function")
        #plotting expected function by dividing the x by period and giving remainder as
        #input to the function, so that x values repeat after given period.
        ax2.semilogy(x,fcoscos(x%period),'y--',label="Expected Function from fourier series")
        ax2.legend(loc='upper right')
        title("Plot of $\cos(\cos(x))$")
        xlabel("x")
        ylabel("$\cos(\cos(x))$")
        grid()
        savefig("Figure2.jpg")
        show()
```



2.1.1 Results and Discussion :

- We observe that e^x is not periodic, whereas $\cos(\cos(x))$ is periodic as the expected and original function matched for the latter but not for e^x.
- Period of $\cos(\cos(x))$ is 2π as we observe from graph and e^x monotously increasing hence not periodic.
- We get expected function by:
  - plotting expected function by dividing the x by period and giving remainder as input to the function, so that x values repeat after given period.
  - That is f(x%period) is now the expected periodic function from fourier series.

2.2 Question 2

- Obtain the first 51 coefficients i.e $a_0, a_1, b_1, ...$ for e^x and $\cos(\cos(x))$ using scipy quad function
- And to calculate the function using those coefficients and comparing with original funcitons graphically.

```
In [6]: #function to calculate
        def fourier_an(x,k,f):
            return f(x)*cos(k*x)

        def fourier_bn(x,k,f):
            return f(x)*sin(k*x)
```

```
In [7]: #function to find the fourier coefficients taking function 'f' as argument.
        def find_coeff(f):

            coeff = []
            coeff.append((quad(f,0,2*pi)[0])/(2*pi))
            for i in range(1,26):
                coeff.append((quad(fourier_an,0,2*pi,args=(i,f))[0])/pi)
                coeff.append((quad(fourier_bn,0,2*pi,args=(i,f))[0])/pi)

            return coeff
```

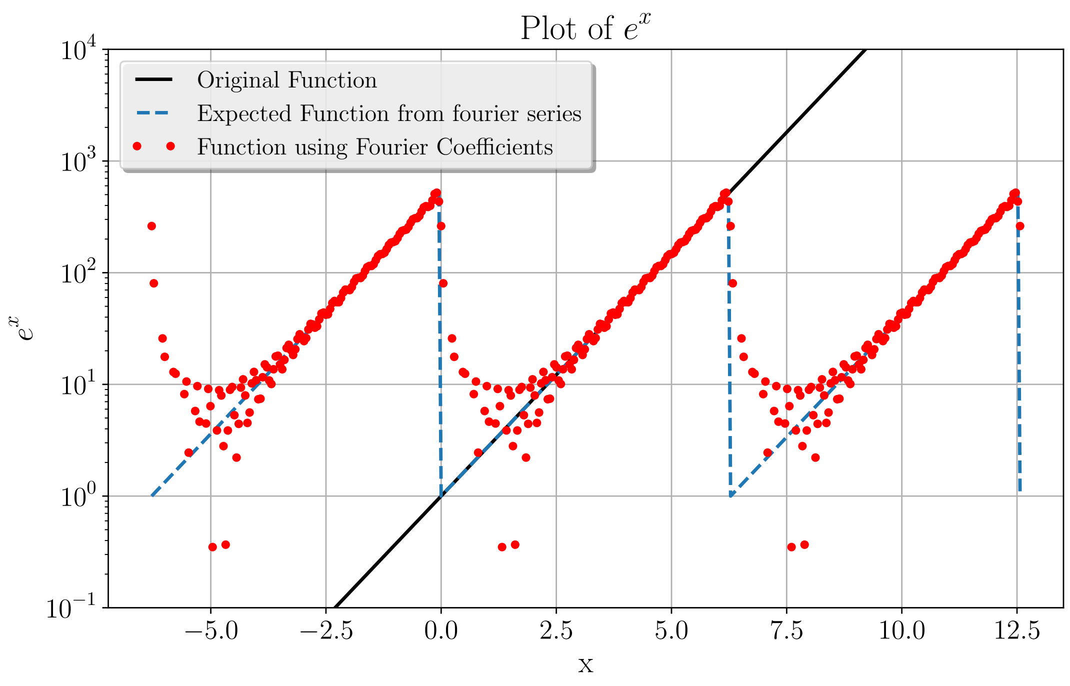
```
In [8]: #function to create 'A' matrix for calculating function back from coefficients
        # with no_of rows, columns and vector x as arguments
        def createAmatrix(nrow,ncol,x):
            A = zeros((nrow,ncol)) # allocate space for A
            A[:,0]=1 # col 1 is all ones
            for k in range(1,26):
                A[:,2*k-1]=cos(k*x) # cos(kx) column
                A[:,2*k]=sin(k*x) # sin(kx) column
            #endfor
            return A
```

```
In [9]: #Function to compute function from coefficients with argument as coefficient vector 'c'
        def computeFunctionfromCoeff(c):
            A = createAmatrix(400,51,x)
            f_fourier = A.dot(c)
            return f_fourier
```

```
In [10]: # Initialising empty lists to store coefficients for both functions
         exp_coeff = []
         coscos_coeff = []
         exp_coeff1 = []
         coscos_coeff1 = []

         exp_coeff1 = find_coeff(fexp)
         coscos_coeff1 = find_coeff(fcoscos)

         # to store absolute value of coefficients
         exp_coeff = np.abs(exp_coeff1)
         coscos_coeff = np.abs(coscos_coeff1)

         # Computing function using fourier coeff
         fexp_fourier = computeFunctionfromCoeff(exp_coeff1)
         fcoscos_fourier = computeFunctionfromCoeff(coscos_coeff1)
```

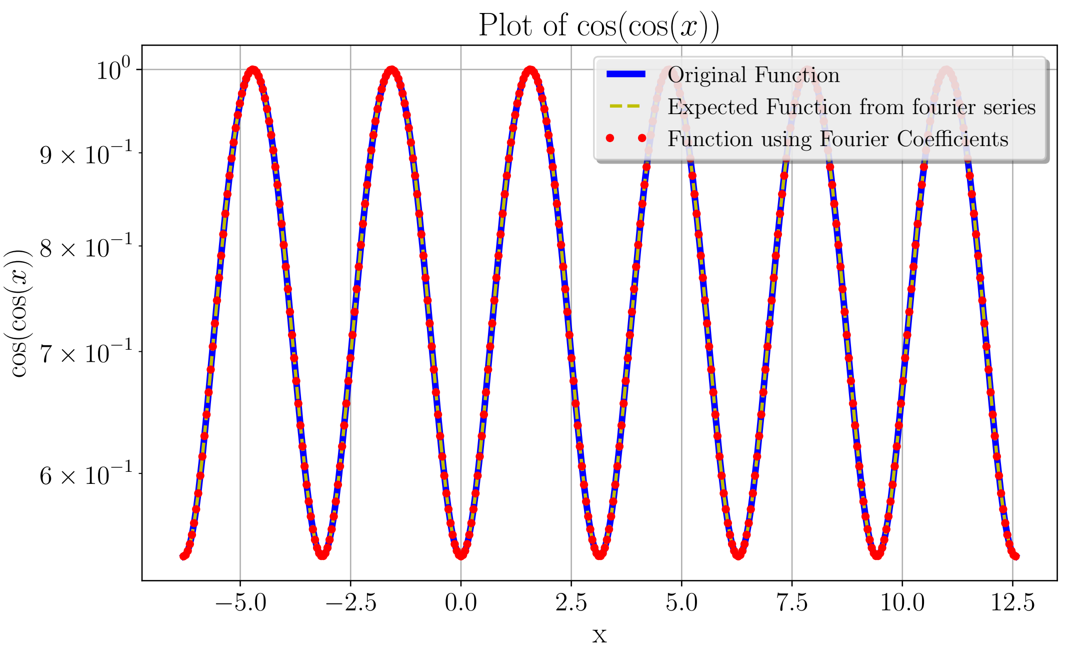
```
In [11]: # Plotting the Function computed using Fourier Coefficients
         ax1.semilogy(x,fexp_fourier,'ro',label = "Function using Fourier Coefficients")
         ax1.set_ylim([pow(10,-1),pow(10,4)])
         ax1.legend()
         fig1
```

```
Out[11]:
```



```
In [12]: ax2.plot(x,fcoscos_fourier,'ro',label = "Function using Fourier Coefficients")
         ax2.legend(loc='upper right')
         fig2
```

```
Out[12]:
```

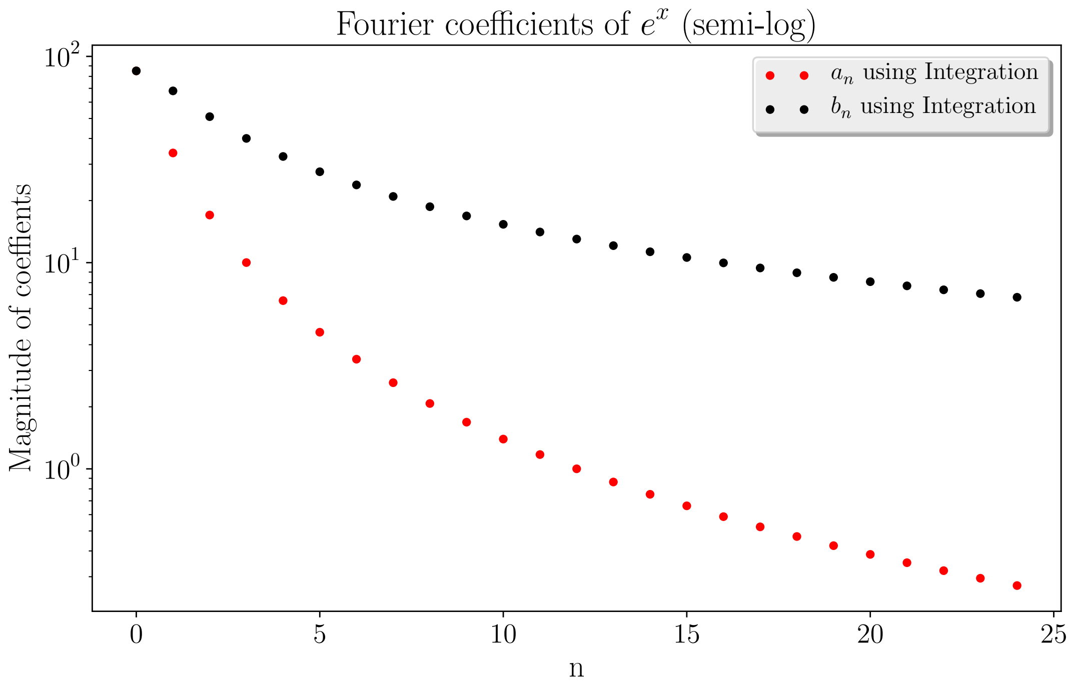


2.3 Question3

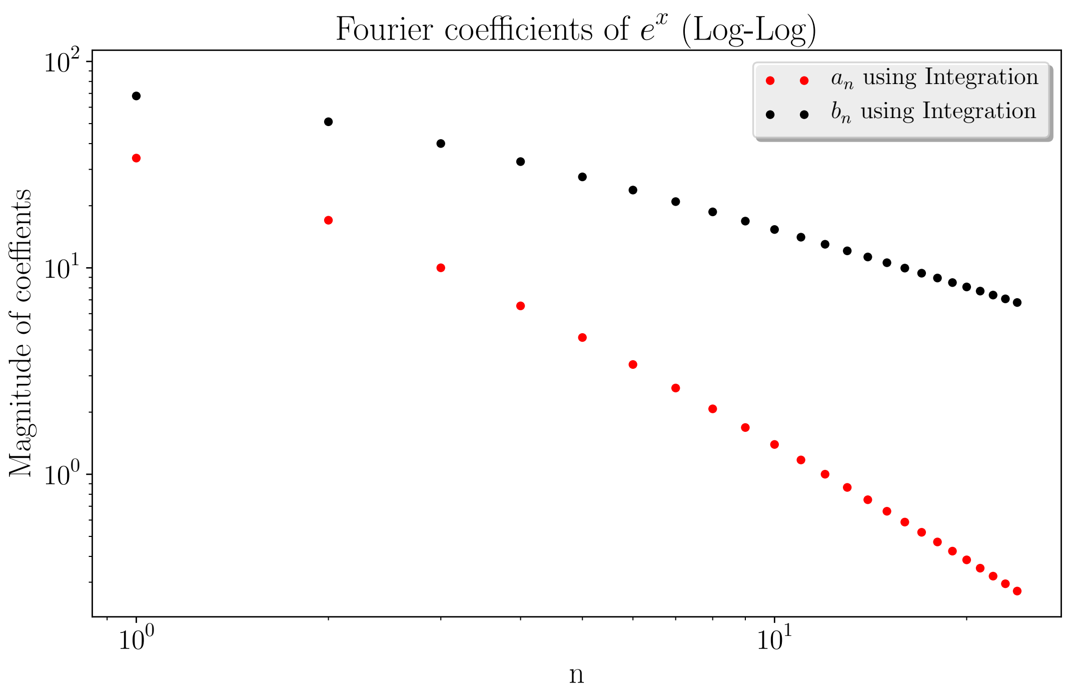
- Two different plots for each function using "semilogy" and "loglog" and plot the magnitude of the coefficients vs n

- And to analyse them and to discuss the observations. ## Plots:
- For each function magnitude of a_n and b_n coefficients which are computed using integration are plotted in same figure in semilog as well as loglog plot for simpler comparisons.

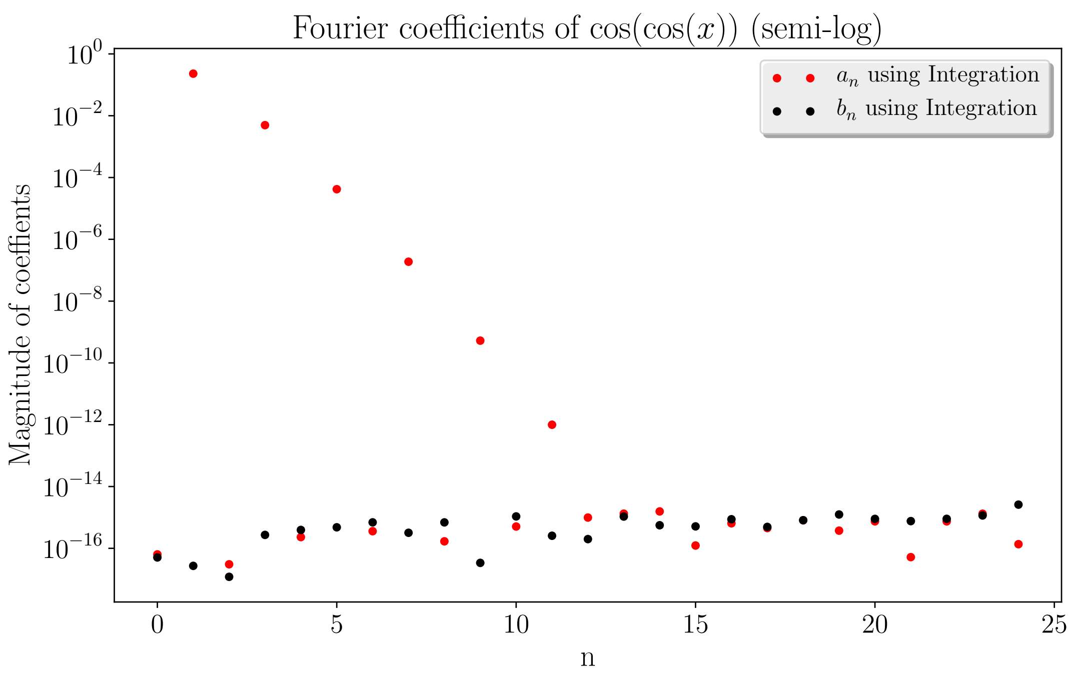
In [13]:
```
# Plotting
fig3 = figure()
ax3 = fig3.add_subplot(111)
# By using array indexing methods we separate all odd indexes starting from 1 -> an
# and all even indexes starting from 2 -> bn
ax3.semilogy((exp_coeff[1::2]),'ro',label = "$a_{n}$ using Integration")
ax3.semilogy((exp_coeff[2::2]),'ko',label = "$b_{n}$ using Integration")
ax3.legend()
title("Fourier coefficients of $e^{x}$ (semi-log)")
xlabel("n")
ylabel("Magnitude of coeffients")
show()
```



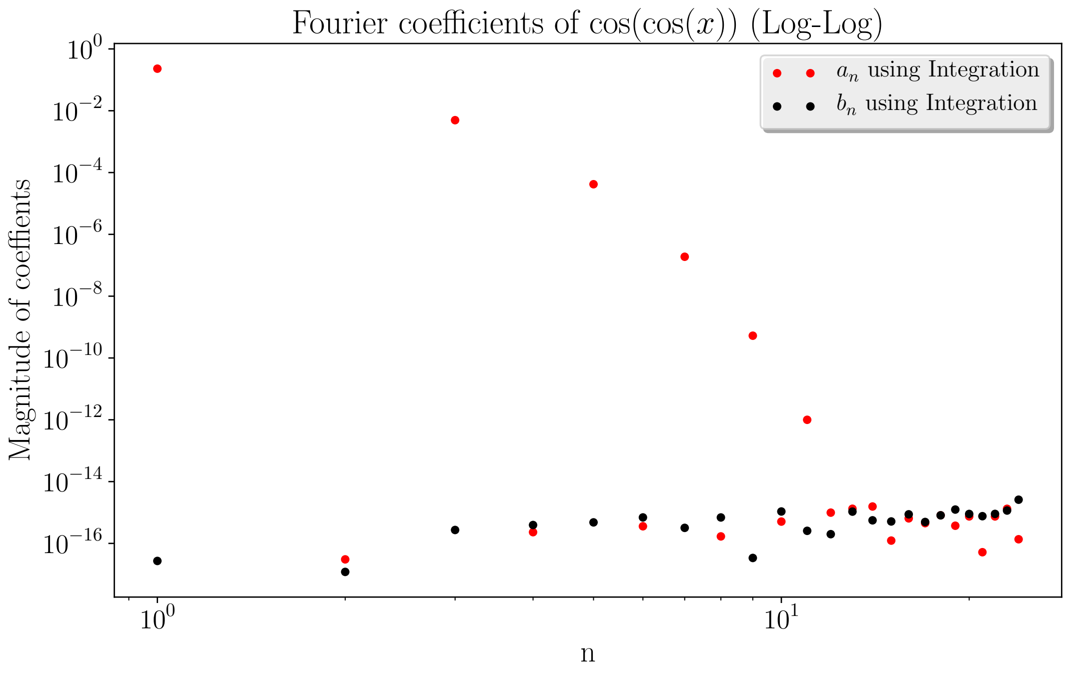
In [14]:
```
fig4 = figure()
ax4 = fig4.add_subplot(111)
# By using array indexing methods we separate all odd indexes starting from 1 -> an
# and all even indexes starting from 2 -> bn
ax4.loglog((exp_coeff[1::2]),'ro',label = "$a_{n}$ using Integration")
ax4.loglog((exp_coeff[2::2]),'ko',label = "$b_{n}$ using Integration")
ax4.legend(loc='upper right')
title("Fourier coefficients of $e^{x}$ (Log-Log)")
xlabel("n")
ylabel("Magnitude of coeffients")
show()
```



```
In [15]: fig5 = figure()
         ax5 = fig5.add_subplot(111)
         # By using array indexing methods we separate all odd indexes starting from 1 -> an
         # and all even indexes starting from 2 -> bn
         ax5.semilogy((coscos_coeff[1::2]),'ro',label = "$a_{n}$ using Integration")
         ax5.semilogy((coscos_coeff[2::2]),'ko',label = "$b_{n}$ using Integration")
         ax5.legend(loc='upper right')
         title("Fourier coefficients of $\cos(\cos(x))$ (semi-log)")
         xlabel("n")
         ylabel("Magnitude of coeffients")
         show()
```



```
In [16]: fig6 = figure()
         ax6 = fig6.add_subplot(111)
         # By using array indexing methods we separate all odd indexes starting from 1 -> an
         # and all even indexes starting from 2 -> bn
         ax6.loglog((coscos_coeff[1::2]),'ro',label = "$a_{n}$ using Integration")
         ax6.loglog((coscos_coeff[2::2]),'ko',label = "$b_{n}$ using Integration")
         ax6.legend(loc='upper right')
         title("Fourier coefficients of $\cos(\cos(x))$  (Log-Log)")
         xlabel("n")
         ylabel("Magnitude of coeffients")
         show()
```



2.3.1 Results and Observations :

(a) The b_n coefficients in the second case should be nearly zero. Why does this happen?
(b) In the first case, the coefficients do not decay as quickly as the coefficients for the second case. Why not?
(c) Why does loglog plot in Figure 4 look linear, wheras the semilog plot in Figure 5 looks linear?

2.4 Question 4 & 5

- Uses least squares method approach to find the fourier coefficients of e^x and $\cos(\cos(x))$
- Evaluate both the functions at each x values and call it b. Now this is approximated by $a_0 + \sum\limits_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx)$
- such that

$$a_0 + \sum_{n=1}^{\infty} a_n \cos(nx_i) + b_n \sin(nx_i) \approx f(x_i) \tag{5}$$

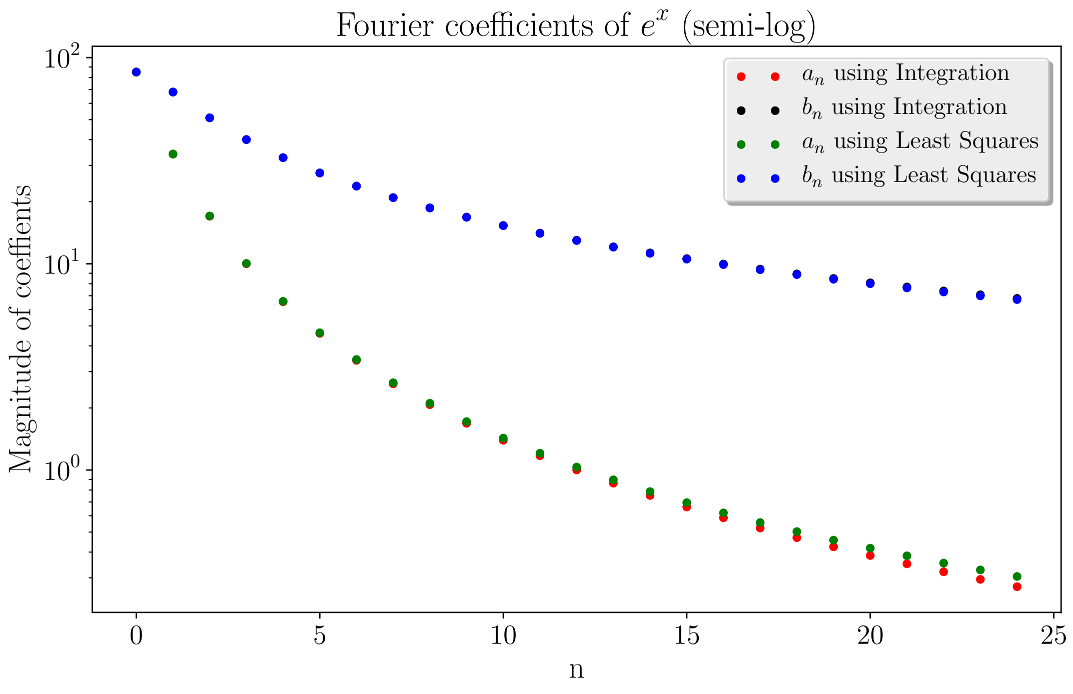
- To implement this we use matrices to find the coefficients using Least Squares method using inbuilt python function 'lstsq'

```
In [17]: #Function to calculate coefficients using lstsq and by calling
         # function 'createAmatrix' which was defined earlier in the code
         # to create 'A' matrix with arguments as function 'f' and lower
         # and upper limits of input x and no_of points needed

         def getCoeffByLeastSq(f,low_lim,upp_lim,no_points):
             x1 = linspace(low_lim,upp_lim,no_points)
             b = f(x1)
             A = createAmatrix(400,51,x1)
             c = []
             c=lstsq(A,b)[0] # the '[0]' is to pull out the
             # best fit vector. lstsq returns a list.
             return c
```

```
In [18]: # Calling function and storing them in respective vectors.
         coeff_exp = getCoeffByLeastSq(fexp,0,2*pi,400)
         coeff_coscos = getCoeffByLeastSq(fcoscos,0,2*pi,400)

         # To plot magnitude of coefficients this is used
         c1 = np.abs(coeff_exp)
         c2 = np.abs(coeff_coscos)
```

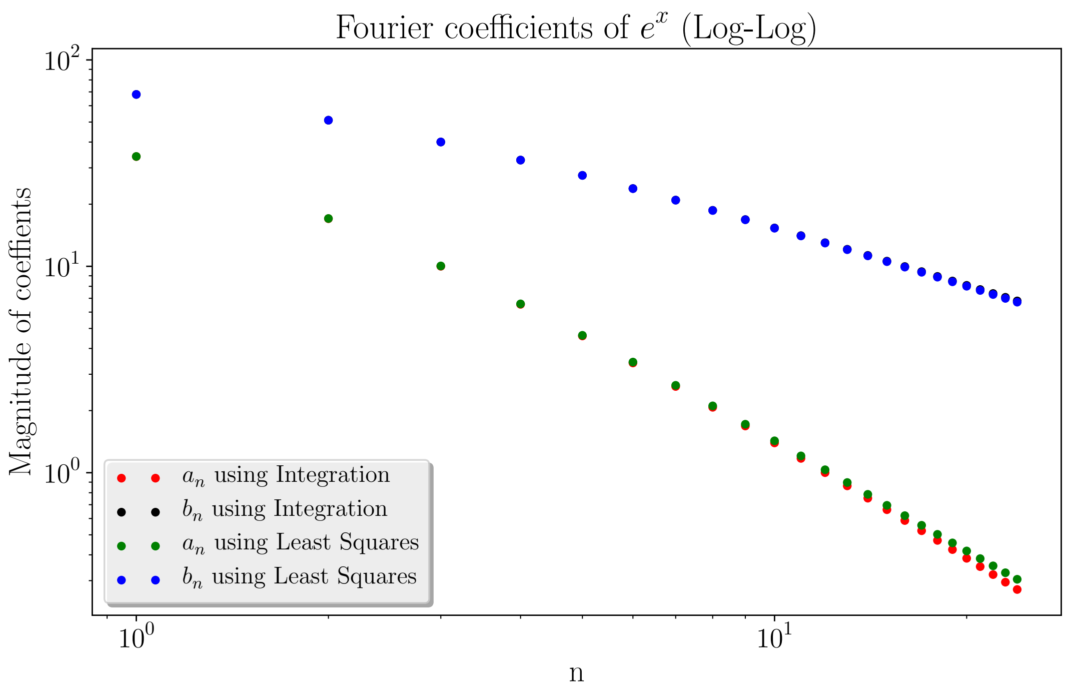
```
In [19]: # Plotting in coefficients got using Lstsq in corresponding figures
         # 3,4,5,6 using axes.
         ax3.semilogy((c1[1::2]),'go',label = "$a_{n}$ using Least Squares")
         ax3.semilogy((c1[2::2]),'bo',label = "$b_{n}$ using Least Squares")
         ax3.legend(loc='upper right')
         fig3
```

```
Out[19]:
```



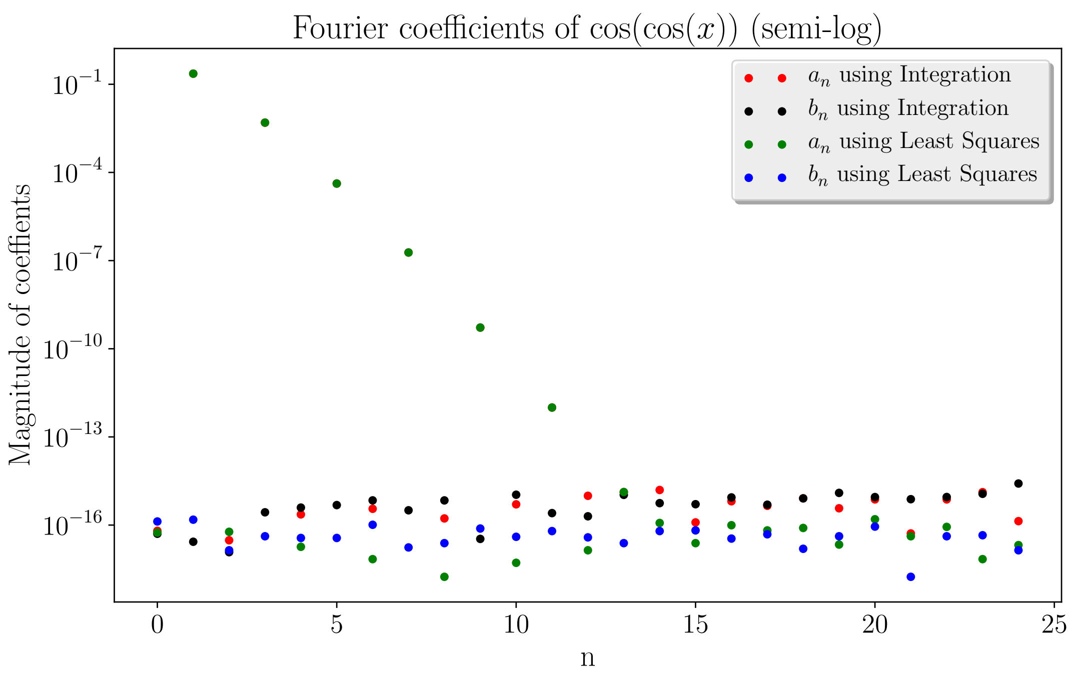
```
In [20]: ax4.loglog((c1[1::2]),'go',label = "$a_{n}$ using Least Squares ")
         ax4.loglog((c1[2::2]),'bo',label = "$b_{n}$ using Least Squares")
         ax4.legend(loc='lower left')
         fig4
```

```
Out[20]:
```

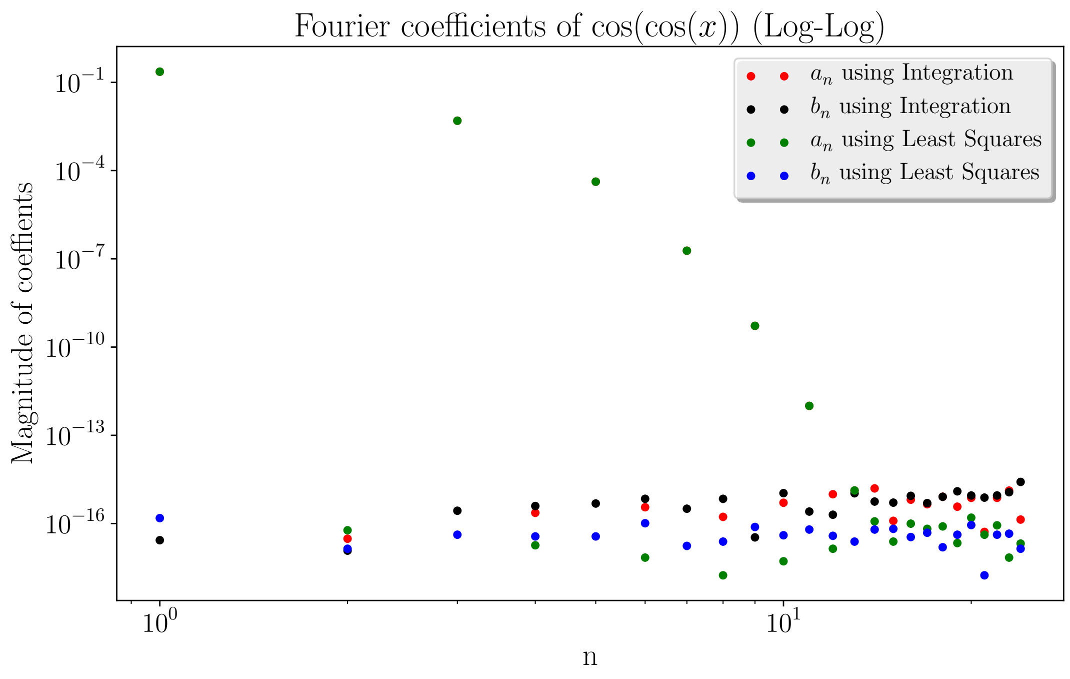


```
In [21]: ax5.semilogy((c2[1::2]),'go',label = "$a_{n}$ using Least Squares")
         ax5.semilogy((c2[2::2]),'bo',label = "$b_{n}$ using Least Squares")
         ax5.legend(loc='upper right')
         fig5
```

```
Out[21]:
```



```
In [22]: ax6.loglog((c2[1::2]),'go',label = "$a_{n}$ using Least Squares ")
         ax6.loglog((c2[2::2]),'bo',label = "$b_{n}$ using Least Squares")
         ax6.legend(loc=0)
         fig6

Out[22]:
```



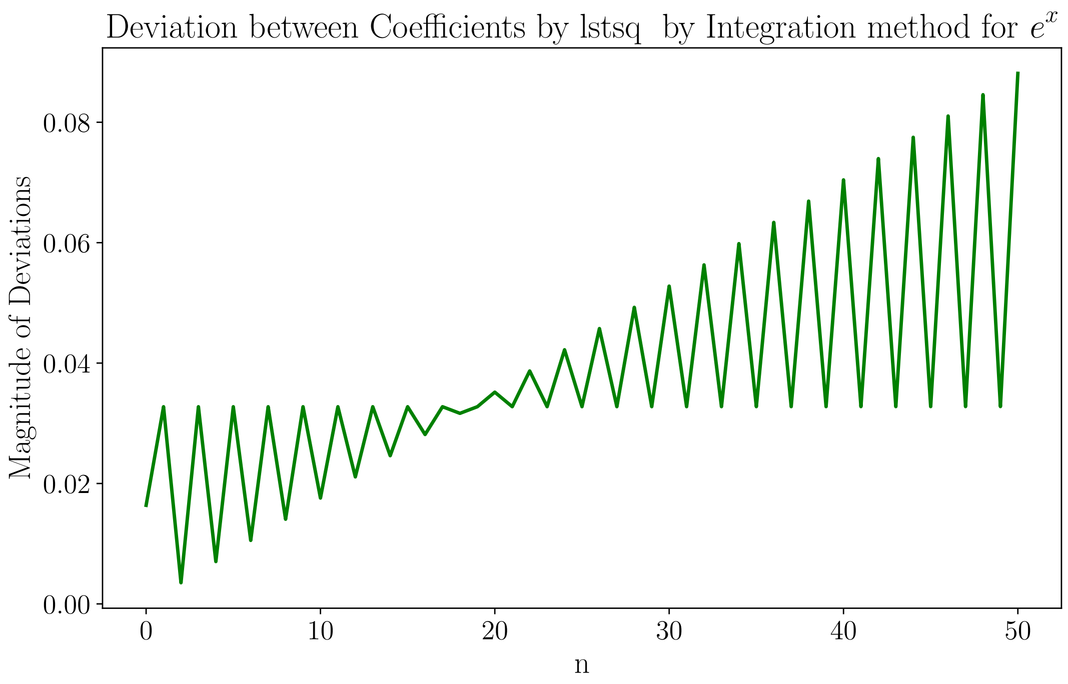
2.5 Question 6

- To compare the answers got by least squares and by the direct integration.
- And finding deviation between them and find the largest deviation using Vectors

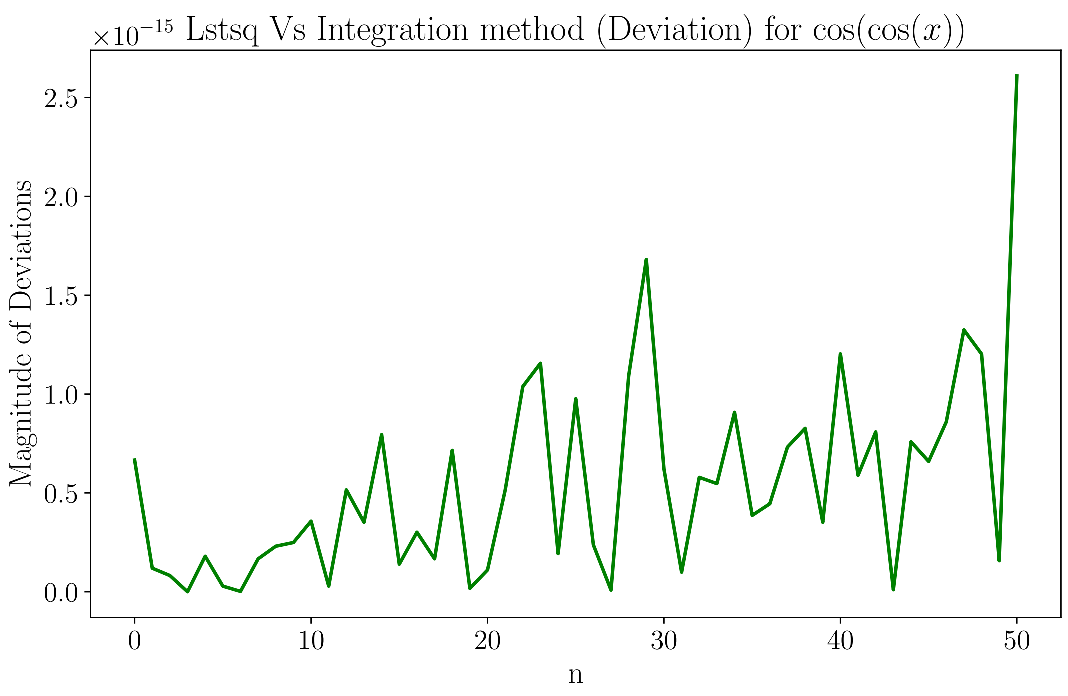
```
In [23]: # Function to compare the coefficients got by integration and
         # least squares and find largest deviation using Vectorized Technique
         # Argument : 'integer f which is either 1 or .
         # 1 -> exp(x)     2 -> cos(cos(x))
         def compareCoeff(f):
             deviations = []
             max_dev = 0
             if(f==1):
                 deviations = np.abs(exp_coeff1 - coeff_exp)
             elif(f==2):
                 deviations = np.abs(coscos_coeff1 - coeff_coscos)

             max_dev = np.amax(deviations)
             return deviations,max_dev

In [24]: dev1,maxdev1 = compareCoeff(1)
         dev2,maxdev2 = compareCoeff(2)

         print("The largest deviation for exp(x) : %g" %(maxdev1))
```

```
        print("The largest deviation for cos(cos(x)) : %g" %(maxdev2))

        # Plotting the deviation vs n
        plot(dev1,'g')
        title("Deviation between Coefficients by lstsq & by Integration method for $e^{x}$")
        xlabel("n")
        ylabel("Magnitude of Deviations")
        show()
```

```
The largest deviation for exp(x) : 0.0881217
The largest deviation for cos(cos(x)) : 2.60883e-15
```



```
In [25]: # Plotting the deviation vs n
         plot(dev2,'g')
         title("Lstsq Vs Integration method (Deviation) for $\cos(\cos(x))$")
         xlabel("n")
         ylabel("Magnitude of Deviations")
         show()
```



2.5.1 Results and Discussion :

- As we observe that there is a significant deviation for e^x as it has discontinuites at $2n\pi$ which can be observed in Figure 1 and hence there will be **Gibbs phenomenon** i.e there will be oscillations around the discontinuity points.
- Due to this the fourier coefficients using least squares will not fit the curve exactly
- Whereas for $\cos(\cos(x))$ the largest deviation is in order of 10^{-15} because the function itself is a periodic function and it is a continous function in entire x range so we get very negligible deviation.
- 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.
- Thats why there is significant deviations are found for e^x

2.6 Question 7

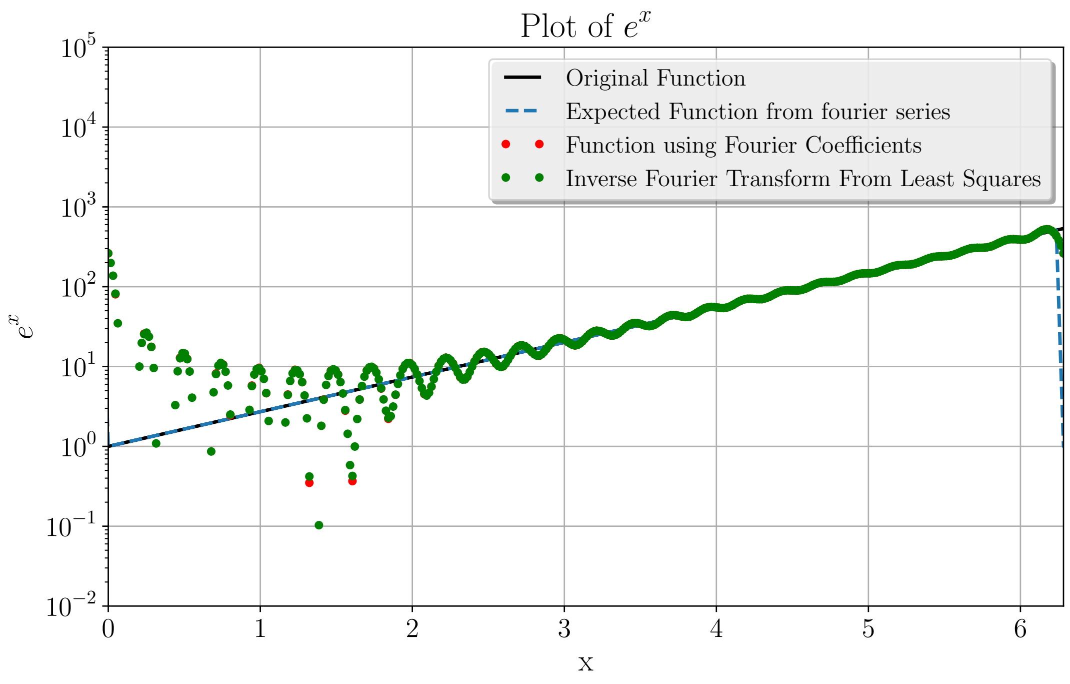
- Computing Ac i.e multiplying Matrix A and Vector C from the estimated values of Coeffient Vector C by Least Squares Method.
- To Plot them (with green circles) in Figures 1 and 2 respectively for the two functions.

```
In [26]: # Define vector x1 from 0 to 2pi
         x1 = linspace(0,2*pi,400)
```

```
In [27]: # Function to reconstruct the signalfrom coefficients
         # computed using Least Squares.
         # Takes coefficient vector : 'c' as argument
         # returns vector values of function at each x
         def computeFunctionbyLeastSq(c):
             f_lstsq = []
             A = createAmatrix(400,51,x1)
             f_lstsq = A.dot(c)
             return f_lstsq
```

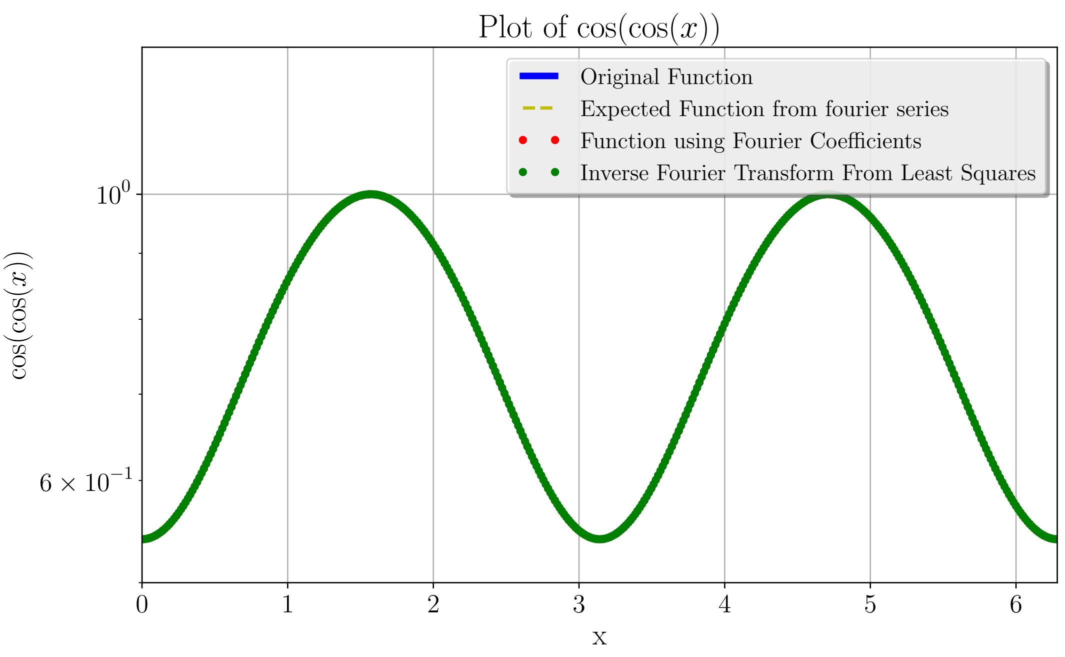
```
In [28]: fexp_lstsq = computeFunctionbyLeastSq(coeff_exp)
         fcoscos_lstsq = computeFunctionbyLeastSq(coeff_coscos)

         # Plotting in Figure1 to compare the original function
         # and Reconstructed one using Least Squares method
         ax1.semilogy(x1,fexp_lstsq,'go',
                      label = "Inverse Fourier Transform From Least Squares")
         ax1.legend()
         ax1.set_ylim([pow(10,-2),pow(10,5)])
         ax1.set_xlim([0,2*pi])
         fig1
```

Out[28]:



```
In [29]: ax2.plot(x1,fcoscos_lstsq,'go',markersize=4,
                  label = "Inverse Fourier Transform From Least Squares")
         ax2.set_ylim([0.5,1.3])
         ax2.set_xlim([0,2*pi])
         ax2.legend()
         fig2
```

Out[29]:



2.6.1 Results and Discussion :

- As we observe that there is a significant deviation for e^x as it has discontinuites at $2n\pi$ which can be observed in Figure 1 and Because 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 rimples are high in starting and reduces and oscillate with more frequency as we go towards 2π. This phenomenon is called **Gibbs Phenomenon**
- Due to this. the orginal 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.
- Thats why there are significant deviations for e^x from original function.
- Whereas for $\cos(\cos(x))$ the curves fit almost perfectly because the function itself is a periodic function and it is a continous function in entire x range so we get very negligible deviation and able to reconstruct the signal with just the fourier coefficients.