EE2703: Applied Programming Lab
Assignment 4
Fourier Approximations

Devaganthan S S
EE19B018

March 15, 2021

0.1 Abstract

This experiment aims to find the Fourier coefficients of coscos(x) and exp(x) using python by Integration and Least Squares Method and compare the results obtained by plotting them.

0.2 Introduction

We try to reconstruct two functions, $exp(x)$ and $cos(cos(x))$ over the the interval $[0, 2\pi)$ using the Fourier series,

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

The coefficients a_k and b_k are obtained by integration, using the function'quad().' They are also obtained by the least-squares method using the function 'lstsq()'

0.3 Implementation

The program prints a set of instructions. Based on the requirement, input has to be given. The program can accept multiple Keys, but they have to be separated by ','. For eg. "2,3,7".

0.4 Results

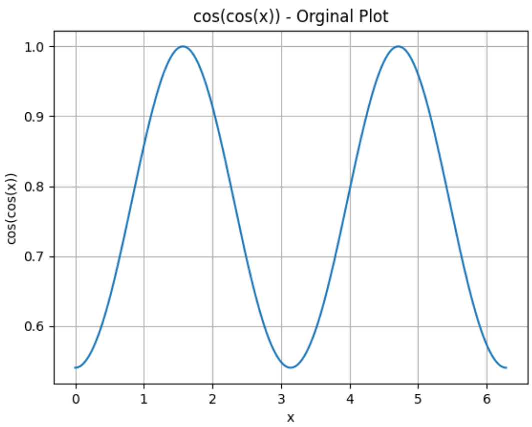
0.4.1 0.1 Plotting the Functions

The functions $coscos(x)$ and $exp(x)$ are plotted over the interval [0,2π). Since $exp(x)$ grows rapidly, 'semilogy()' is used to plot. The function $coscos(x)$ is periodic, whereas the function $exp(x)$ is not. Since we require periodic functions for the Fourier series, for exp(x), we take the interval [0,2π), and assume it repeats. The below code accomplishes the above.

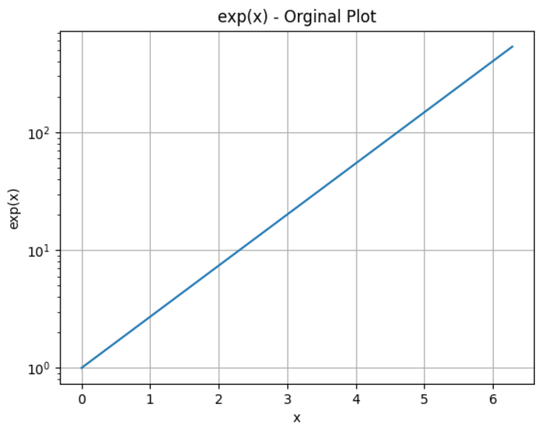
```
space = np.linspace(0, m.pi*2, 401)
# The fucntion takes in a vector and returns a coscos() of the
    vector
f1x = vecCoscos(space)
plot(space, f1x)  # Plot for coscos()
title('cos(cos(x)) - Orginal Plot')
xlabel('x')
ylabel('cos(cos(x))')
grid()
```

```
show()
# The fucntion takes in a vector and returns a exp() of the vector
f2x = vecExp(space)
semilogy(space, f2x)  # Semilogy plot for exp()
title('exp(x) - Orginal Plot')
xlabel('x')
ylabel('exp(x)')
grid()
show()
```

The plots obtained are,



(a) Figure 1



(b) Figure 2

0.4.2 Generating the Coefficients by Integration

The First 26 'a' coefficients and the 25 'b' coefficients are generated. The below code generates the coefficients for $exp(x)$,

```
a1 = []  # FOr Absolute 'a' coefficients
b1 = []  # FOr Absolute 'b' coefficients
a2 = []  # FOr 'a' coefficients
b2 = []  # FOr 'b' coefficients
def f1(x): return m.exp(x)
def f2(x, k): return m.exp(x)*m.cos(x*k)
def f3(x, k): return m.exp(x)*m.sin(x*k)


for i in range(26):
    if i == 0:
        temp = (quad(f1, 0, 2*m.pi)[0])/(m.pi*2)  # Computes the Coeffecients
        a1 = a1 + [abs(temp)]
```

```
        a2 = a2 + [temp]
    else:
        temp = (quad(f2, 0, 2*m.pi, args=i)[0])/(m.pi*2)
        temp2 = (quad(f3, 0, 2*m.pi, args=i)[0])/(m.pi*2)
        a1 = a1 + [abs(temp)]
        a2 = a2 + [temp]
        b1 = b1 + [abs(temp2)]
        b2 = b2 + [temp2]

return a1, b1, a2, b2
```

For $cos(cos(x))$

```
def genCoefcoscos():
    a1 = []  # FOr Absolute 'a' coefficients
    b1 = []  # FOr Absolute 'b' coefficients
    a2 = []  # FOr 'a' coefficients
    b2 = []  # FOr 'b' coefficientsos(m.cos(x))
    def f2(x, k): return m.cos(m.cos(x))*m.cos(x*k)
    def f3(x, k): return m.cos(m.cos(x))*m.sin(x*k)
    for i in range(26):
        if i == 0:
            temp = (quad(f1, 0, 2*m.pi)[0])/(m.pi*2)
            a1 = a1 + [abs(temp)]
            a2 = a2 + [temp]
        else:
            temp = (quad(f2, 0, 2*m.pi, args=i)[0])/(m.pi*2)
            temp2 = (quad(f3, 0, 2*m.pi, args=i)[0])/(m.pi*2)
            a1 = a1 + [abs(temp)]
            a2 = a2 + [temp]
            b1 = b1 + [abs(temp2)]
            b2 = b2 + [temp2]
    return a1, b1, a2, b2
```

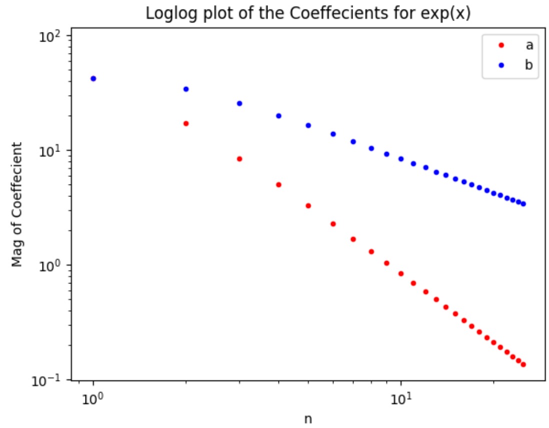
0.4.3 Plotting the magnitude of Coeffecients Obtained from Integ

For the obtained coefficients, the magnitude vs n plot is plotted using 'semilogy()' and 'loglog()' functions. The below code accomplishes the above,

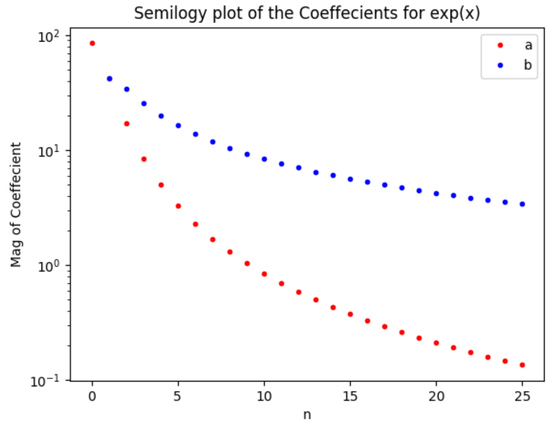
```
def loglogPlotter(x1, a2, label1, color1, title1, xlabel1, ylabel1):
    loglog(x1, a2, 'r.', label=label1, color=color1)
    title(title1)
    xlabel(xlabel1)
    ylabel(ylabel1)
    legend()

```

```python

def semilogyPlotter(x1, a2, label1, color1, title1, xlabel1, ylabel1):
    semilogy(x1, a2, 'r.', label=label1, color=color1)
    title(title1)
    xlabel(xlabel1)
    ylabel(ylabel1)
    legend()


# For exp()
x = np.arange(26)
loglogPlotter(x, abs_aexp, 'a', 'red',
              'Loglog plot of the Coeffecients for exp(x)', 'n', 'Mag of Coeffecient')
loglogPlotter(x[1:], abs_bexp, 'b', 'blue',
              'Loglog plot of the Coeffecients for exp(x)', 'n', 'Mag of Coeffecient')
show()
semilogyPlotter(x, abs_aexp, 'a', 'red',
                'Semilogy plot of the Coeffecients for exp(x)', 'n', 'Mag of Coeffecient')
semilogyPlotter(x[1:], abs_bexp, 'b', 'blue',
                'Semilogy plot of the Coeffecients for exp(x)', 'n', 'Mag of Coeffecient')
show()
# For coscos(x)
loglogPlotter(x, abs_acos, 'a', 'red',
              'Loglog plot of the Coeffecients for coscos(x)', 'n', 'Mag of Coeffecient')
loglogPlotter(x[1:], abs_bcos, 'b', 'blue',
              'Loglog plot of the Coeffecients for coscos(x)', 'n', 'Mag of Coeffecient')
show()
semilogyPlotter(x, abs_acos, 'a', 'red',
                'Semilogy plot of the Coeffecients for coscos(x)', 'n', 'Mag of Coeffecient')
semilogyPlotter(x[1:], abs_bcos, 'b', 'blue',
                'Semilogy plot of the Coeffecients for coscos(x)', 'n', 'Mag of Coeffecient')
show()
```

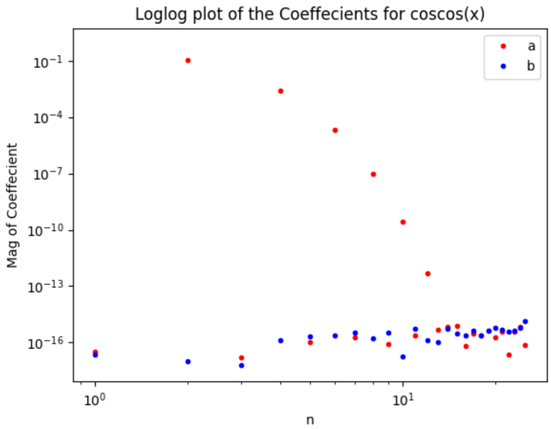
The Plots Obtained are,



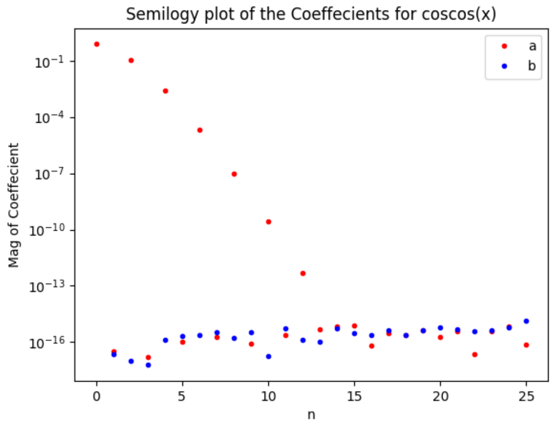
(a) Figure 3



(b) Figure 4



(a) Figure 5

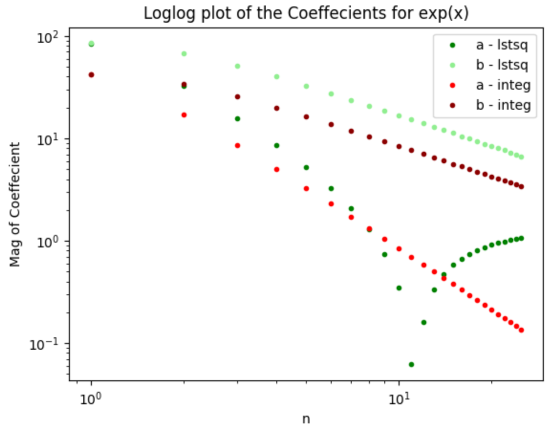


(b) Figure 6

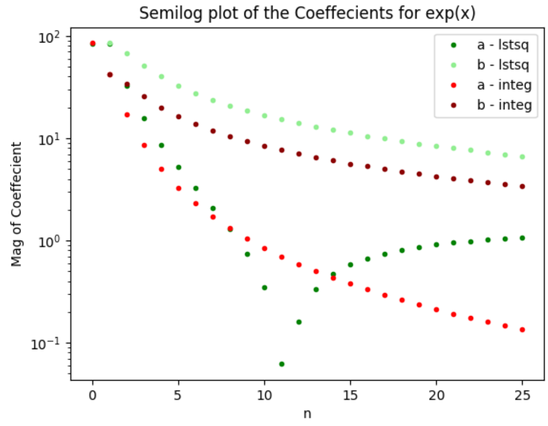
From the fig 5 and fig 6 we can see that b_n coefficients are nearly zero. This happens because $cos(cos(x))$ is an odd function. In the first case, the coefficients do not decay as quickly as the coefficients for the second case because, $exp(x)$ requires infinite number of cos and sin terms, whereas $cos(cos(x))$ does not.

0.4.4 Plotting the magnitude of Coefficients Obtained from lstsq()

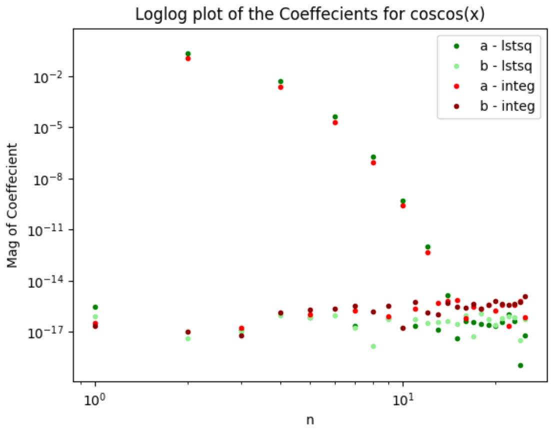
For the obtained coefficients, the magnitude vs n plot is plotted using 'semilogy()' and 'loglog()' functions. The code is similar to as given in section 0.4.3, with only the second argument for the plot functions are the coefficients obtained from lstsq() .The Plots Obtained are,



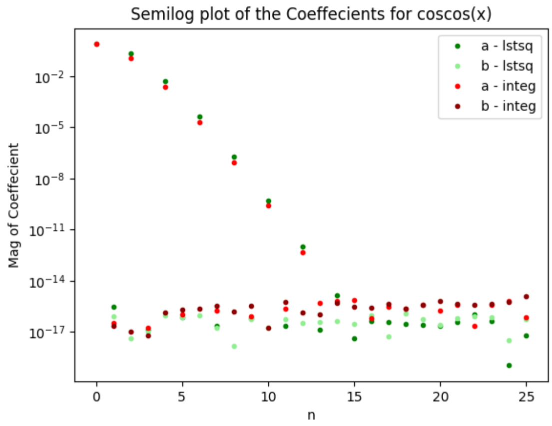
(a) Figure 7



(b) Figure 8



(a) Figure 9



(b) Figure 10

0.4.5 Comparing the Coefficients

The coefficients obtained from the 2 methods do not agree. The least-square method gives 51 coefficients that best approximate the function. Whereas the coefficients obtained from Integration, do not give a best approximate, because they need other coefficients (coefficients other than the 51 computed), to best approximate the function. The below code computes the deviation in each coefficient and also prints the largest deviation.

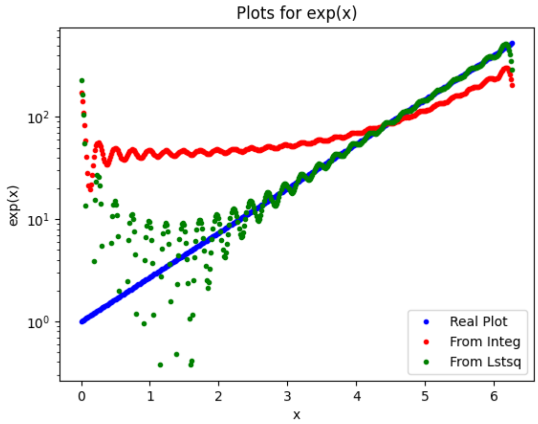
```
deviaA = []
deviaB = []
print('Deviation in "a" Coeffecients,')
for i in range(26):
    print('a{} = {}'.format(i, abs(abs_aexp[i]-lst_abs_aexp[i])))
    deviaA = deviaA + [abs(abs_aexp[i]-lst_abs_aexp[i])]
print('Deviation in "b" Coeffecients,')
for i in range(25):
    print('b{} = {}'.format(i+1, abs(abs_bexp[i]-lst_abs_bexp[i]))
    )
    deviaB = deviaB + [abs(abs_bexp[i]-lst_abs_bexp[i])]
print('Maximum Deviation = {}'.format(max([max(deviaA), max(deviaB
    )])))
```

0.4.6 Reconstructing the Function and Plotting using the Coefficients

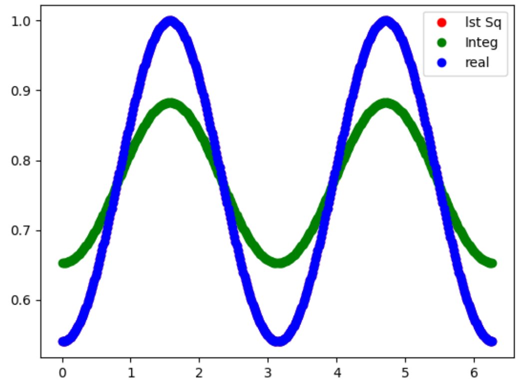
With the Coefficients obtained from both the methods, the function is reconstructed. The reconstructed functions are plotted along with the original function, for the interval, $[0,2\pi)$. The below code accomplishes the above for $exp(x)$. For $cos(cos(x))$ the code is the same, with a different set of coefficients.

```
x = linspace(0, 2*m.pi, 401)
x = x[:-1]
f = []
freal = vecExp(x)
for i in A:
    f = f + [sum(i*realexpCoeff)]
f2 = []
for i in A:
    f2 = f2 + [sum(i*cMatrixExp)]

# Plots the real Function
semilogyPlotter(x, freal, 'Real Plot', 'blue',
                'Plots for exp(x)', 'x', 'exp(x)')
# Plots the reconstructed function using coeff from Integ Method
semilogyPlotter(x, f, 'From Integ', 'red',
                'Plots for exp(x)', 'x', 'exp(x)')
# Plots the reconstructed function using coeff from Lst Sw Method
semilogyPlotter(x, f2, 'From Lstsq', 'green',
                'Plots for exp(x)', 'x', 'exp(x)')
legend()
show()
```

The plots obtained are,



(a) Figure 11



(b) Figure 12

The reconstruction of cos(cos(x)) is nearly perfect while exp(x) is not. The reason is that cos(cos(x)) can be written as a sum of a finite number of scaled harmonics while exp(x) cannot.

0.5 Conclusion

51 Coefficients for both the function are found using the Integration method. The best approximate for the 2 functions, constraining to 51 coefficients are found using the least square method.