Vector Operations and Functions in Python

SANKALP SAOJI (EE16B063)

April 5, 2018

Abstract

In this assignment we learnt about vector operations and functions in python. We learned how to plot graphs in python using the built in modules like matplotlib and integrate using built in functions from scipy.

We also learnt how to implement self-made integration function using trapezoidal integration.

1 Introduction

One of the best part of python is scientific python which is used for scientific and technical computing. This contains many modules like SciPy , NumPy and Matplotlib. These modules together are put in a package called Pylab. These modules enables Python to do almost all the things that MATLAB can do and sometimes even much more. From handling vectors, handling array of elements to plotting graphs, Python can do all that MATLAB can do.
This assignment uses these modules for the calculation of definite integrals with the help of quad function, calculating inverse function of tan and also many vector operations.

1.1 The numpy module

NumPy module helps us to manipulate large multidimensional arrays and matrices. It also enables us to do lot of mathematical operations on these arrays.

1.2 The scipy module

SciPy contains modules for optimization, linear algebra, integration, interpolation, FFT, signal and image processing and other tasks common in science and engineering. In this assignment we have used Quad function which we imported from scipy.integrate module to do definite integration of functions.

1.3 Plotting using matplotlib

In this assignment we have used Pylab module which gives us the privilge to do plotting almost rivalling MATLAB. It gives us a lot of options to manipulate the graphs using labels, titles, legends etc. There is a separate module called matplotlib to plot different kind of graphs. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

2 Assignment Questions

First the following modules were imported.

```
from scipy.integrate
import quad from pylab import *
from matplotlib import pyplot as plt
import numpy
```

2.1 Function Definition

The function func(t) is defined so that it can take a vector argument as input and return a vector of the same size after operation.

```
# Defining function

def func(t):
        m = 1/(1+(float(t)**2))
        return m
```

2.2 Vector Generation

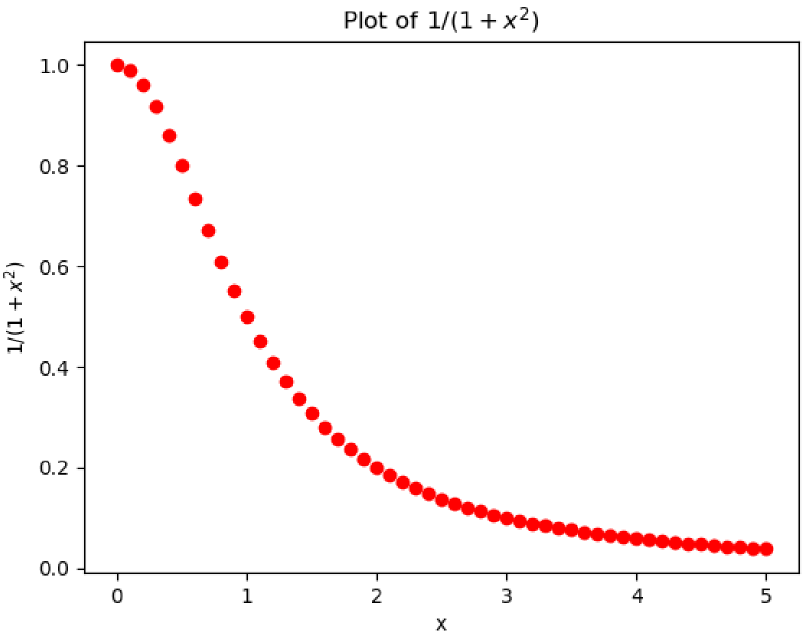
A vector 'x' is defined using the arange function, initiating from 0 and ending at 5 in steps of 0.1.

```
# Defining vector x

x = arange(0, 5.1, 0.1)
```

2.3 Plot of func(t) vs t

```
# Plotting with labels and title

for i in range(0, len(x)):
        plot(x[i], func(x[i]), 'ro')

plt.title('Plot of $1/(1+x^{2})$')
plt.xlabel('x')
plt.ylabel('$1/(1+x^{2})$')
plt.show()
```

```
xlabel() and ylabel() are used to label the axes.
The title() function is used to displays title at the top of the graph.
The '$' sign indicates that the enclosed string is a LATEX expression.
The show() command displays the graph.
```



2.4 Tan inverse function from its integral definition

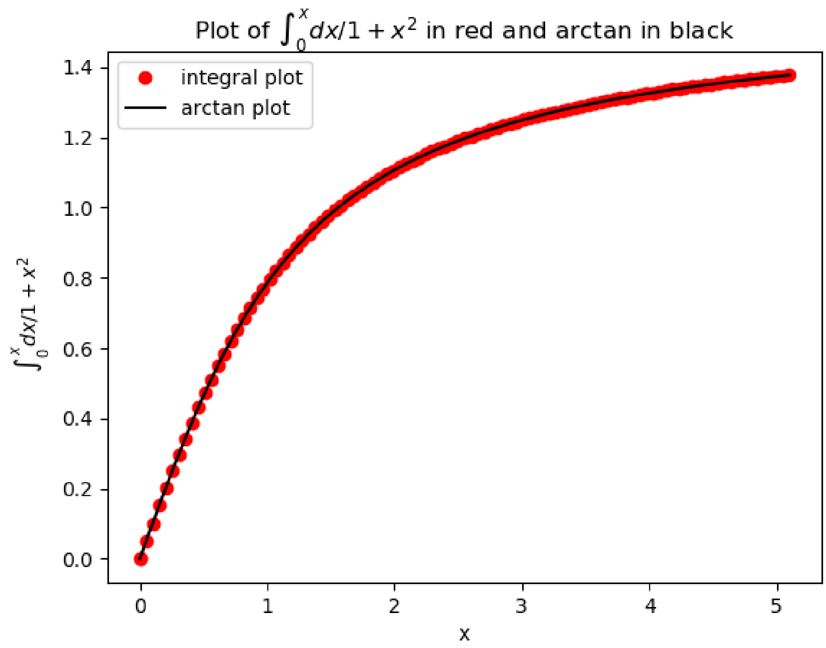
We used quad() function to integrate $\frac{1}{1+t^2}$ as per the required range.

```
#Scipy Integration using quad()

for i in range(0, len(x)):
        integration_ans, integration_error = quad(func, 0 , x[i])
        print(x[i], integration_ans)
plt.plot(x[i], integration_ans, 'ro')
```

The input vector x[i] is passed to the quad() function to integrate.
Since quad function returns two values, the value of the integral and associated error. We are only interested in the integral value, therefore only the 1st element of the array i.e. the value of the integral was stored in a variable integration_ans.
The variable stores the value of $\tan^{-1}$x .

```
#Plotting of Arctan and the Integral obtained with quad()

plt.plot(x[0], 0 , 'ro', label = 'integral plot')
plt.plot(x, j, '#000000', label = 'arctan plot')
plt.xlabel('x')
plt.ylabel(' $\int_{0}^{x} dx/{1+x^{2}}$')
plt.title('Plot of $\int_{0}^{x} dx/{1+x^{2}}$ in
red and arctan in black')
plt.legend()
plt.show()
```

The plot of $\tan^{-1}$x and x is shown.
In the 2nd plot, argument 'ro' allows us to plot the data to be plotted as red dots.
The legend() command adds a legend to the graph indicating the respective values shown.



Following are the tabulated values of the plots.

arctan quad	x
0	0
0.1	0.0996687
0.2	0.197396
0.3	0.291457
0.4	0.380506
0.5	0.463648
0.6	0.54042
0.7	0.610726
0.8	0.674741
0.9	0.732815
1	0.785398
1.1	0.832981
1.2	0.876058
1.3	0.915101
1.4	0.950547
1.5	0.982794
1.6	1.0122
1.7	1.03907
1.8	1.0637
1.9	1.08632
2	1.10715
2.1	1.12638
2.2	1.14417
2.3	1.16067
2.4	1.17601
2.5	1.19029
2.6	1.20362
2.7	1.21609
2.8	1.22777
2.9	1.23874
3	1.24905

3.1	1.25875
3.2	1.26791
3.3	1.27656
3.4	1.28474
3.5	1.2925
3.6	1.29985
3.7	1.30683
3.8	1.31347
3.9	1.31979
4	1.32582
4.1	1.33156
4.2	1.33705
4.3	1.3423
4.4	1.34732
4.5	1.35213
4.6	1.35674
4.7	1.36116
4.8	1.3654
4.9	1.36948

arctan quad	x
0	0
0.1	0.0996687
0.2	0.197396
0.3	0.291457
0.4	0.380506
0.5	0.463648
0.6	0.54042
0.7	0.610726
0.8	0.674741
0.9	0.732815
1	0.785398
1.1	0.832981
1.2	0.876058
1.3	0.915101
1.4	0.950547
1.5	0.982794
1.6	1.0122
1.7	1.03907
1.8	1.0637
1.9	1.08632
2	1.10715
2.1	1.12638
2.2	1.14417
2.3	1.16067
2.4	1.17601
2.5	1.19029
2.6	1.20362
2.7	1.21609
2.8	1.22777

2.9	1.23874
3	1.24905
3.1	1.25875
3.2	1.26791
3.3	1.27656
3.4	1.28474
3.5	1.2925
3.6	1.29985
3.7	1.30683
3.8	1.31347
3.9	1.31979
4	1.32582
4.1	1.33156
4.2	1.33705
4.3	1.3423
4.4	1.34732
4.5	1.35213
4.6	1.35674
4.7	1.36116
4.8	1.3654
4.9	1.36948

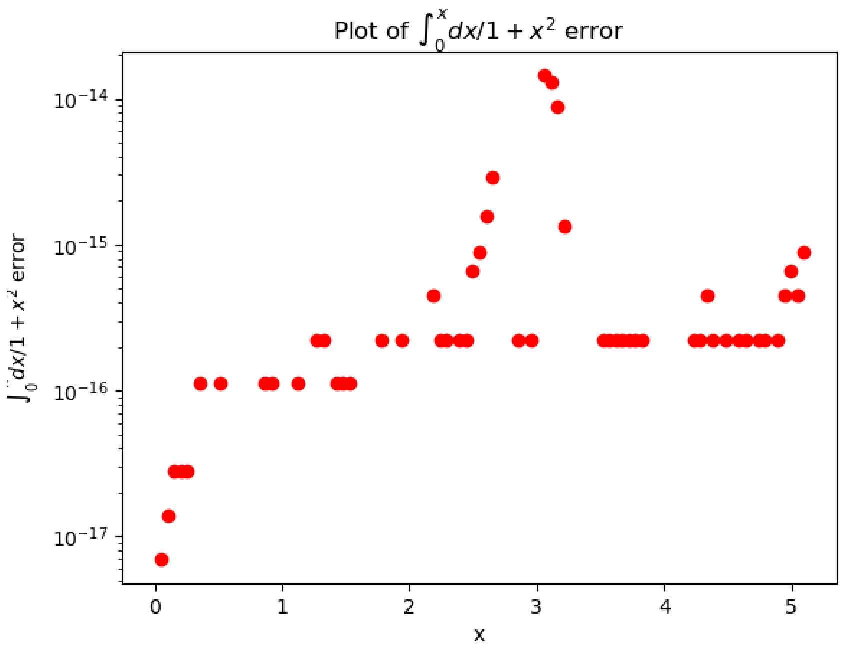
2.4.1 Error in the integral method

The error between the arctan function and the quad function is plotted.

```
#Calculating error

integration_ans = numpy.zeros(len(x))
for k,i in zip(x,range(0, len(x))):
        integration_ans[i] = quad(func, 0 , k)[0]
        print(k, integration_ans)

err = abs(integration_ans-j)
```

abs() function is used to get absolute value of the difference between integration_ans and j. i.e. the values obtained from quad() and
the arctan() function itself.
The graph is plotted in a semilogy plot.

```
#Plotting the error

plt.semilogy(x, err, 'ro')
plt.xlabel('x')
plt.ylabel('$\int_{0}^{x} dx/{1+x^{2}}$ error')
plt.title('Plot of $\int_{0}^{x} dx/{1+x^{2}}$ error')
plt.show()
```



2.5 Trapezoidal Integration

Here trapezoidal integration is used to find the integration of a function. According to the Trapezoidal Rule, if a function is known at points a , a + h , . . . , b , its integral is given by

$$I = \begin{cases} 0 & x=a \\ 0.5(f(a)+f(x_i))+\sum_{j=1}^{i-1} f(x_j) & x=a+ih \end{cases}$$

This can be rewritten as

$$I_i = h\left(\sum_{j=1}^{i} f(x_j) - \frac{1}{2}\left(f(x_1)+f(x_i)\right)\right)$$

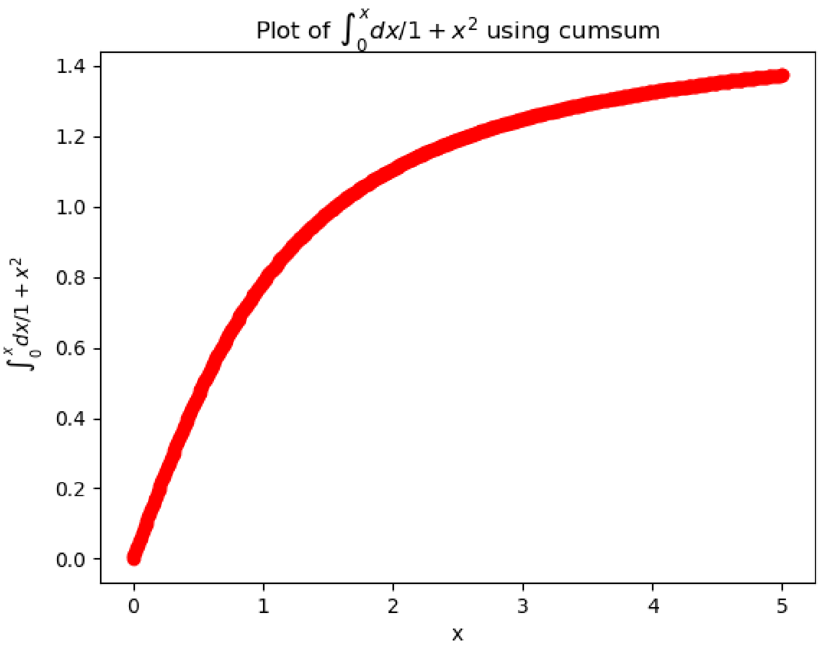
def integ(p): if (p == 0): I = 0 elif (p != 0): add = 0 i = int(p/h) for j in y[0:i]: add = add + j final = [] final.append(h*((add)-0.5*(y[0]) - 0.5*(y[i]))) I = 0 for l in final: I = I + l return I

Using a smaller value for steps ize 'h' in trapezoidal integration gives more accurate values.

The summation part can be calculated by using cumsum() function. A function to do trapezoidal integration is defined as integ(). It takes one argument whose integration is carried out as shown above.

```
def x(k,a,h):
        n = a + k*h
        return n
step = 0.01
vec = (numpy.linspace(0,5,(5/step) +1),object)
array = func(vec[0])
s = numpy.cumsum(array)
integration = step*(s-0.5*(func(x(1,0,step))+array))
ans = numpy.where(vec[0]==1)
print(integration[ans])
```

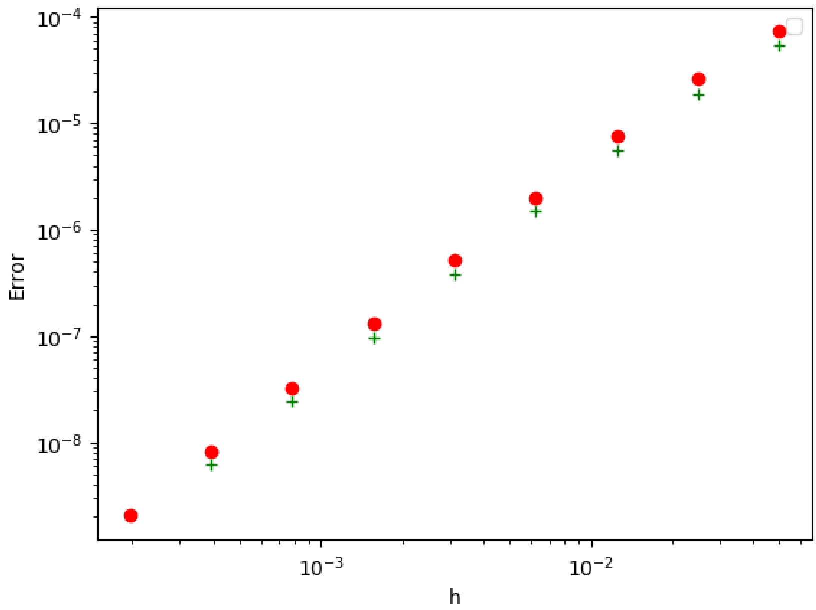
Above algorithm uses the cumsum() function to shorten the algorithm.



2.5.1 Error calculation of the trapezoidal method

In the above code, I have calculated the estimated error and the exac error. Following graph shos the obtained plot.

```
step=[0.05] vec = (numpy.linspace(0,5,(5/step[-1]) +1))
vec = (numpy.linspace(0,5,(5/step[-1]) +1))
array = func(vec)
s = numpy.cumsum(array)
exact_integ = numpy.arctan(vec)
integ_1 = step[-1]*(s-0.5*(func(x(1,0,step[-1]))+array))
max_error = numpy.amax(integ_1)
true_error = np.amax(abs(exact_integ-integ_1))

estimate=list()
exact=[true_error]
while max_error > 1e-8:
stepi=(step[-1])/2
vec_new = (numpy.linspace(0,5,(5/stepi) +1))
array_new = func(vec_new)
s=numpy.cumsum(array_new)
integ_2 = stepi*(s-0.5*(func(x(1,0,stepi))+array_new))
tan_def = numpy.arctan(vec_new)
comm = numpy.nonzero(numpy.in1d(vec_new,vec))[0]
integ_cmp = integ_2[comm]
max_error = numpy.amax(abs(integ_1-integ_cmp))
true_error = numpy.amax(abs(integ_2-tan_def))
estimate.append(max_error)
exact.append(true_error)
integ_1 = integ_2
vec = vec_new
array = array_new
step.append(stepi)
estimate.append(0)
```



This is the graph I obtained.