Vector Operations and Functions in Python
Assignment 2

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

February 4, 2018
First created on January 31,2018

Report

1.1 Introduction

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!.

1.2 Python code

1.2.1 Code to create a tan inverse function from its integral definition

Integral definition of $tan^{-1}(x)$ and plotting it

```
In [34]: #Importing libraries needed
         from pylab import *
         from scipy.integrate import quad
         from math import pi
         from  tabulate import tabulate
         #Function which takes vector x as argument used in calculation of tan inverse(x)

         def f(x):
             return 1.0/(1+np.square(x))

         #end of function

         #Function to integrate f(x) from 0 to x[i](upper limit) using quad function for
         #all elements in vector x, resulting answer is tan inverse(x[i]) using for loop
         #method

         def tan_inv(x):

             ans = np.zeros(len(x))             #initialising vector answer and error with zeros
             err = np.zeros(len(x))             #with length that of input vector x

             for i in range(len(x)):            #loop to calculate integral for all values of x
                 ans[i],err[i] = quad(f,0,x[i])
             return ans,err

         #end of function tan_inv

         #declaring vector x
         x = arange(0,5,0.1)
         y = f(x)                         # y is another vecotr which stores vector returned by f(x)

         #plotting f(x) vs x
         fig1 = figure()
         plot(x,y)
         fig1.suptitle(r"Plot of $1/(1+t^{2})$", fontsize=20)
         xlabel("x")
```

```
fig1.savefig('1.jpg')

#calculating tan inverse of all elements in x by arctan function
tan_inv_exact = np.arctan(x)

#plotting tan inverse vs x
fig2 = figure()
plot(x,tan_inv_exact)

#calculating tan_inverse through quad function and storing error associated
I_quad,err = tan_inv(x)

table = zip(tan_inv_exact,I_quad)
headers = ["arctan(x)","quad_fn:integral"]
#tabulating arctan values vs quad function values
print tabulate(table,tablefmt="fancy_grid",headers=headers)

#plotting tan_inverse calculated using quad in same plot of arctan
plot(x,I_quad,'ro')
legend( (r"$tan^{-1}x$","quad fn"))
fig2.suptitle(r"Plot of $tan^{-1}x$", fontsize=20)
xlabel("x")
ylabel("$\int_{0}^{x} du/(1+u^{2})$")
fig2.savefig('2.jpg')

#plotting error associated with quad function while calulating tan_inverse

fig3 = figure()
semilogy(x,abs((tan_inv_exact-I_quad)),'r.')
fig3.suptitle(r"Error in  $\int_{0}^{x} dx/(1+t^{2}) $", fontsize=12)
xlabel("x")
ylabel("Error")
fig3.savefig('3.jpg')

show()
```

arctan(x)	scipy : $\int_0^x dx/(1+t^2)$
0.00000	0.00000
0.09967	0.09967
0.19740	0.19740
0.29146	0.29146
0.38051	0.38051
0.46365	0.46365
0.54042	0.54042
0.61073	0.61073
0.67474	0.67474
0.73282	0.73282
0.78540	0.78540
0.83298	0.83298
0.87606	0.87606
0.91510	0.91510
0.95055	0.95055
0.98279	0.98279
1.01220	1.01220
1.03907	1.03907
1.06370	1.06370
1.08632	1.08632
1.10715	1.10715
1.12638	1.12638
1.14417	1.14417
1.16067	1.16067
1.17601	1.17601
1.19029	1.19029
1.20362	1.20362
1.21609	1.21609
1.22777	1.22777
1.23874	1.23874
1.24905	1.24905
1.25875	1.25875
1.26791	1.26791
1.27656	1.27656
1.28474	1.28474
1.29250	1.29250
1.29985	1.29985
1.30683	1.30683
1.31347	1.31347
1.31979	1.31979
1.32582	1.32582
1.33156	1.33156
1.33705	1.33705
1.34230	1.34230
1.34732	1.34732
1.35213	1.35213
1.35674	1.35674
1.36116	1.36116
1.36540	1.36540
1.36948	1.36948

output_0_1.png

output_0_2.png

output_0_3.png

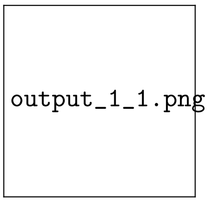
1.2.2 Integration using Trapezoidal rule

Implementing with for loops without vectorization

In [38]:
```
#Now we use numerical methods to calculate integral of f(x) using trapezoidal rule
#instead of quad function with for loops without vectorizing it
import time as t

#I is the vector which stores the integral values of f(x) using trapezoidal rule
I = []
h=0.1               #h is stepsize
x=arange(0,5,h)     #x is input vector from 0 to 5 with stepsize 0.1

#Function which takes index of lower limit and upperlimit and stepsize as arguments
#and calulates using trapezoidal rule

def trapez(lower_index,i,h):
    Ii = h*((cumsumlike(i))-0.5*(f(x[lower_index])+f(x[i])))
    return Ii

#Its function to calculate cumulative sum till upper limit index i of input vector x
#this is implemented with for loop
def cumsumlike(i):
    temp=0
    for k in range(i):
        temp+=f(x[k])
    return temp

#noting down time it takes to  run
t1 = t.time()
for k in range(len(x)):
    I.append(trapez(0,k,h))          #appending the values in vector
t2 = t.time()

print ("Time took without vectorization : %g" %(t2-t1))
#plotting Integral of x vs x
```

```
fig4 = figure()
plot(x,I,'r.')
fig4.suptitle(r"Trapezoid rule : $\int_{0}^{x} dx/(1+t^{2}) $",fontsize=12)
xlabel("x")
ylabel("$\int_{0}^{x} dx/(1+t^{2}) $")
fig4.savefig('4.jpg')
show()
```

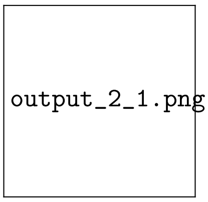
```
Time took without vectorization : 0.005651
```



Trapezoidal rule using Vectorized Method

```
In [39]: #Using Vectorized code and noting the time it takes to run
         t3 = t.time()
         I_vect = h*(cumsum(f(x))-0.5*(f(x[0])+f(x)))          #vectorized code
         t4 = t.time()

         print ("Time took with vectorization : %g" %(t4-t3))
         print ("Speed up factor while vectorizing code : %g" % ((t2-t1)/(t4-t3)))

         #plotting integral  vs x using vectorized technique
         fig5 = figure()
         plot(x,I_vect)
         fig5.suptitle(r"Vectorized method : $\int_{0}^{x} dx/(1+t^{2}) $",fontsize=12)
         xlabel("x")
         ylabel("$\int_{0}^{x} dx/(1+t^{2}) $")
         fig5.savefig('5.jpg')
         show()
```

```
Time took with vectorization : 0.000273943
Speed up factor while vectorizing code : 20.6284
```



Calculating Estimate error and Exact error associated with Trapezoidal rule by halving stepsize

```
In [37]: #Estimating error by halving stepsize when greater the certain tolerance
         #initialising h vector of length 10 with 0.1 as initial value

         h = 0.1+np.zeros(10)
         tol = 10**-8                    #tolerance of 10^(-8)
         est_err = 1 + np.zeros(10)      #estimated error initialisation
         act_err = 1 + np.zeros(10)      #actual_error initialisation
         i=0

         #while loop runs until est_err is less than tolerance
```

```
while(est_err[i]>tol):

    #temperory estimated_Error array,used to find max error among common points

    est_err_temp = []
    h[i+1]=h[i]/2.0                 # halving h by 2
    x=arange(0,5,h[i])              # creating input with current stepsize
    x_next = arange(0,5,h[i+1])     #input with half of current stepsize

    #calculating Integrals with current h and h/2
    I_curr = h[i]*(cumsum(f(x))-0.5*(f(x[0])+f(x)))
    I_next = h[i+1]*(cumsum(f(x_next))-0.5*(f(x_next[0])+f(x_next)))

    #finding common elements
    x_com = np.intersect1d(x,x_next)

    #finding error between Integrals at common elements
    for k in range(len(x_com)):
        est_err_temp.append(I_next[2*k]-I_curr[k])

    #finding index of max error among common elements
    arg_max_err = argmax(absolute(est_err_temp))

    #finding actual error and estimated error
    act_err[i] = arctan(x_com[arg_max_err])-I_curr[arg_max_err]
    est_err[i] = est_err_temp[arg_max_err]

    #incrementing i when est_error is greater than tolerance
    if(est_err[i]>tol):
        i+=1

#Tabulating h values vs est_error vs act_errors
table = zip(h[:-1],est_err[:-1],act_err[:-1])
headers = ["Stepsize h","Estimated Error","Actual Error"]
#tabulating arctan values vs quad function values
print tabulate(table,tablefmt="fancy_grid",headers=headers)

print"Best value of h is : %g" %(h[8])

fig6 = figure()
loglog(h[:-1],est_err[:-1],'g+')
loglog(h[:-1],act_err[:-1],'ro')
legend(("Estimated error","Exact error"))
fig6.suptitle(r"Estimated Error vs Exact error :$\int_{0}^{x} dx/(1+t^{2}) $", fontsize=12)
xlabel("h")
ylabel("Error")
fig6.savefig('6.jpg')
show()
```

h	estimated error	exact error
0.1	0.000405844	0.000541031
0.05	0.000101395	0.000135188
0.025	2.5373e-05	3.38303e-05
0.0125	6.34297e-06	8.45727e-06
0.00625	1.58573e-06	2.1143e-06
0.003125	3.96435e-07	5.2858e-07
0.0015625	9.91086e-08	1.32145e-07
0.00078125	2.47772e-08	3.30362e-08
0.000390625	6.19429e-09	8.25906e-09

Best value of h is : 0.000390625

output_3_1.png