Vectors and Functions

EE16B014
G.Harsha Kanaka Eswar

6th April,2018

Abstract

In this Report we are going to see how to define a function,and vectors.We define functions in such a way that it takes a vector arguments. We will also discuss use of functions and the integration operation using quad from scipy.Then we will take a look at trapezoidal rule of computation of integration by using cumsum.

1 Functions:

In python Function generally takes the inputs as its arguments,and does the desired mathematical operations on them and gives the output.Its is most helpful because for performing the same operation we need not write the same code again and again thus reducing the length of the code.
Now lets define a function $f(t) = \frac{1}{1+t^2}$:

```
def f(t):
     a=1/(1+(t)**2)
     return a
```

What the above function does is it takes values of 't' as input and performs the assigned operation and stores the value, and 'return' is used to get the output from function for further use of computed value in the code.
Now lets define a vector 'x' (we have to import *numpy* for using linspace) and give it as an input to the above function and get the output:

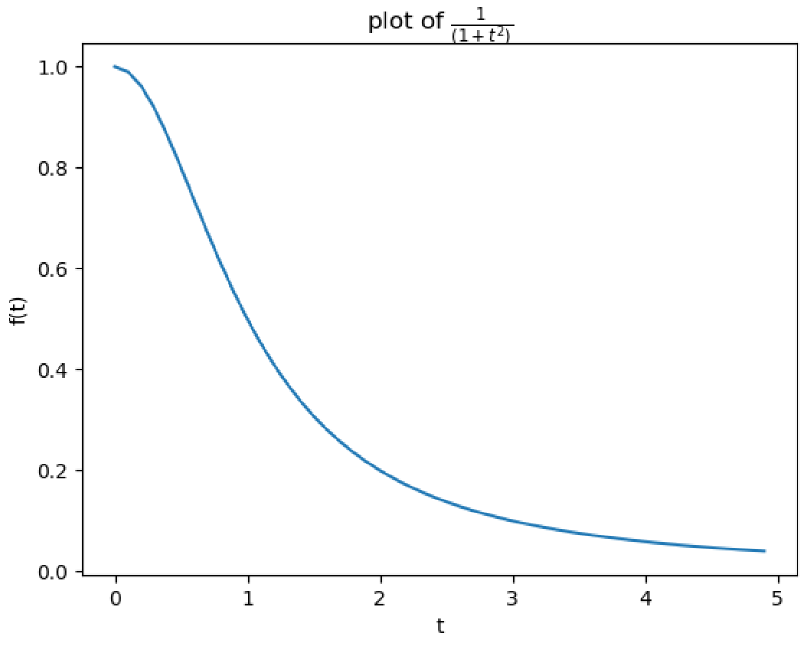
```
from pylab import *
import numpy
x=arange(0,5,0.1)#Define a Vector
```

Lets call the function 'f(t)' giving above vector 'x' s the input and store the output in y And plot it.

```
y=f(x)
plot(x,y)
```

```
xlabel('t')
ylabel('f(t)')
plt.title(r'plot of $\frac{1}{(1+t^2)}$')
show()
```

Lets take a look at the graph:



2 Quad Usage:

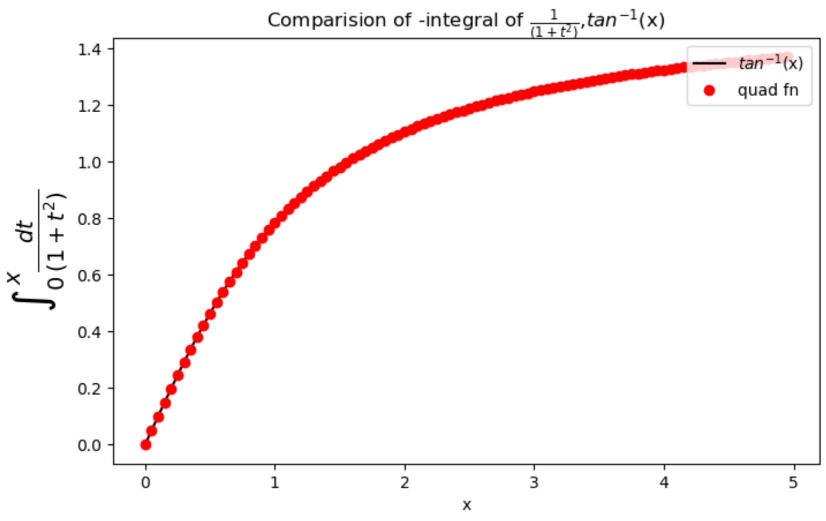
Okay we got the function $f(t) = \frac{1}{1+t^2}$ now lets integrate this function,Which should give the output as $\int_0^x \frac{1}{1+t^2}$dt=tan^{-1}x,By using '$quad$' we get two values as output one is integrated value and other gives an estimated error in the computation.
Lets see the code:

```
d=[]
e=[]
for i in x:
    a,b=quad(f,0,i)
    b=abs(arctan(i)-a)
    e.append(b)
    d.append(a)
z=arctan(x)
```

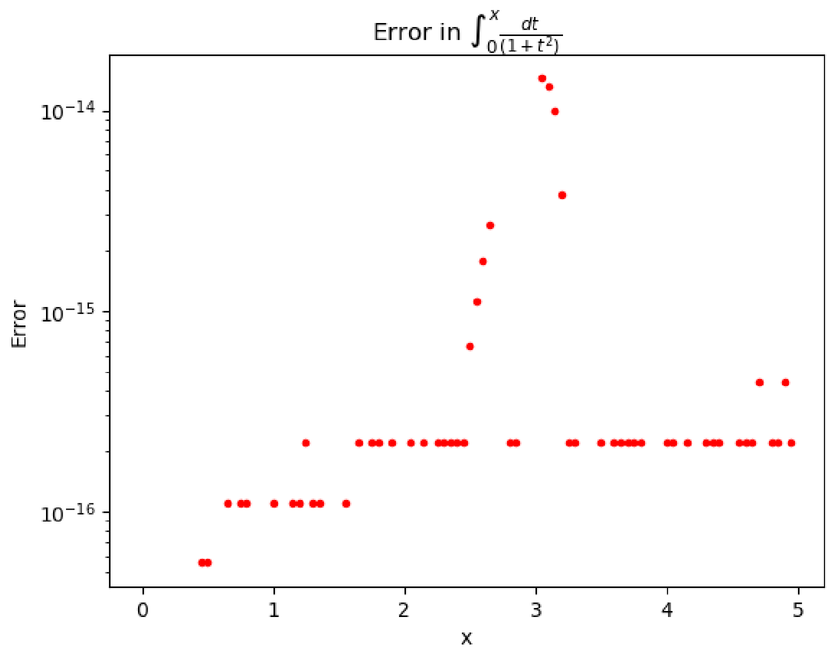
```
f1=figure()
plot(x,z,'k',label="$tan^{-1}$(x)")
plot(x,d,'ro',label="quad fn")
legend(loc="upper right")
title(r'Comparision of -integral of $\frac{1}{(1+t^2)}$,$tan^{-1}$(x)')
plt.ylabel(r"$\int_0^x \frac{dt}{(1+t^2)}$",fontsize=20)
xlabel('x')

f2=figure()
semilogy(x,e,'r.')
plt.ylabel('Error')
plt.xlabel('x')
plt.title(r"Error in $\int_0^x \frac{dt}{(1+t^2)}$")
show()
```

Okay then lets plot the integral output along with tan^{-1}x and see how good is the integration:



As we can see the quad function is almost accurately matching with function,but let us see the error in a semilogy axis that how much the quad function deviates from original output:



So the error is very minimum which is in order of 10^{-15}.

3 Trapezoidal Rule:

If a function is known at different points between points 'a,b' at an interval 'h' ,i.e $a, a+h, a+2h, ..., b$.Then we can compute the integral of function with limits a,b.

$$I = \begin{cases} 0 & x = a \\ 0.5(f(a) + f(x_i)) + \sum_{j=2}^{i-1} f(x_j) & x = a + h \end{cases}$$

Where I is the integral we can also write it as:

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

Now lets compute the above equation for tan^{-1}x using code:

```
def f(t):
    a=1/(1+(t)**2)
    return a
c=[]
```

```
for x in l
     a=h*(f(x))+a
     b=a-0.5*h*(f(0)+f(x))
c.append(b)
```

In The above code we are using *'for' loop* for computation and we have used lists to append the values obtained now lets see a more optimized one using vectors and **Cumsum**:

```
l=arange(0,5,0.1)
h=0.1
k=arange(0,len(l),1)
print(k)
def f(t):
    a=1/(1+(t)**2)
    return a
c=f(l)
b=cumsum(c)
e=h*(b[k]-0.5*(f(0)+f(l)))
```

Now we can see the for loop is avoided and this is much faster because of usage of vectors.

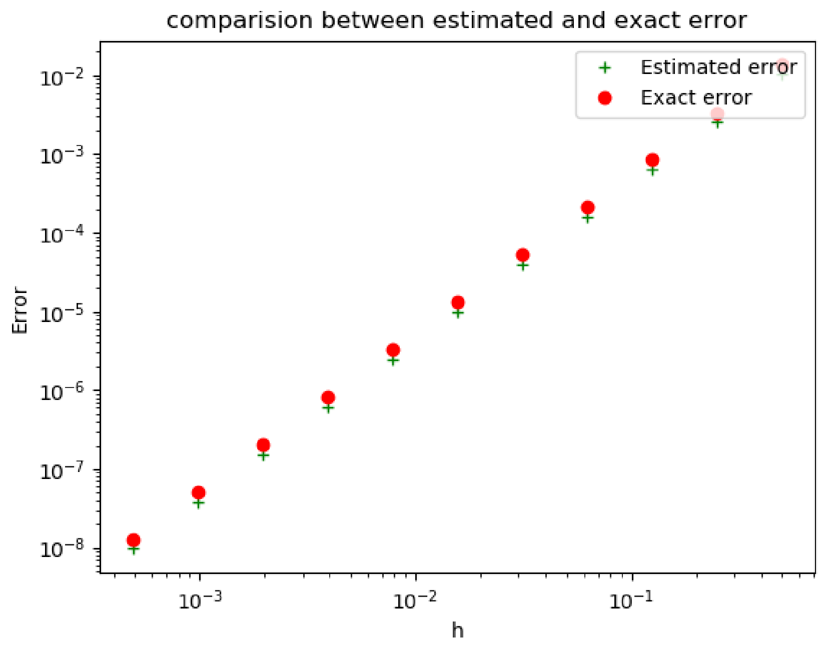
3 Tolerance in Error:

We computed the values of function using Trapezoidal rule but there will be definitely some error in the values when compared to exact integration,And it is mainly depends on 'h' Value,The more smaller 'h' value we take the smaller the error is.
But we can't just give a random small 'h' value which may take a lot of computational time so what we will do is we will get the *estimated error* for different values of 'h' and gradually reducing 'h' to half until we get the error under the tolerance level.

```
def integral(h=0):
     l=arange(0,1,h)
     k=arange(0,len(l),1)
     c=f(l)
     b=cumsum(c)
     e=h*(b[k]-0.5*(f(0)+f(l)))
     return e
h=0.5
e=1
e1=[]
h1=[]
```

```
e2=[]
while (e>10**(-8)):
    d1=integral(h)
    d2=integral(h/2)
    err=[]
    l=arange(0,1,h)
    y=arange(0,len(l),1)
    err=abs(d2[2*y]-d1[y])
    e=max(err)
    b=where(err==e)[0]
    m=arctan(b*h)-d1[b]
    h1.append(h)
    e2.append(m)
    e1.append(e)
    h=h/2
title('comparision between estimated and exact error')
xlabel('h')
ylabel('Error')
loglog(h1,e1,"+",color="green",label="Estimated error")
loglog(h1,e2,"ro",label="Exact error")
legend(loc="upper right")
show()
```

Here is difference between estimated and exact error plot:



Here we can observe that there is no big difference between estimated and exact error. so when we don't know the exact integral of a function, we choose 'h' value from the estimated error.