Vector Operations and Functions in Python

Mohammed Muqeeth
EE16B026
Electrical Engineering

February 6, 2018

Abstract

This report presents how to plot graph of a function, compute integral using quad function and trapezoidal algorithm and their corresponding error calculations.

1 define a function to take a vector argument

- Take vector as argument
- Return $1/1 + t^2$

code:

```
from pylab import *
from numpy import *
def f(x):
    return 1.0/(1+x*x)
x=arange(0,5.1,.1)
y = f(x)
print y
```

output:

[1. 0.99009901 0.96153846 0.91743119 0.86206897 0.8 0.73529412 0.67114094 0.6097561 0.55248619 0.5 0.45248869 0.40983607 0.37174721 0.33783784 0.30769231 0.28089888 0.25706941 0.23584906 0.21691974 0.2 0.18484288 0.17123288 0.15898251 0.14792899 0.13793103 0.12886598 0.12062726 0.11312217 0.10626993 0.1 0.09425071 0.08896797 0.08410429 0.07961783 0.0754717 0.07163324 0.06807352 0.06476684 0.06169031 0.05882353 0.05614823 0.05364807 0.05130836 0.04911591 0.04705882 0.04512635 0.04330879 0.04159734 0.03998401]

2 Define a vector x

- import numpy ,pylab.
- use linspace

code:

```
from pylab import *
from numpy import *
x=linspace(0,5.0,51)
print x
```

output:

```
[          0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5
           2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5. ]
```

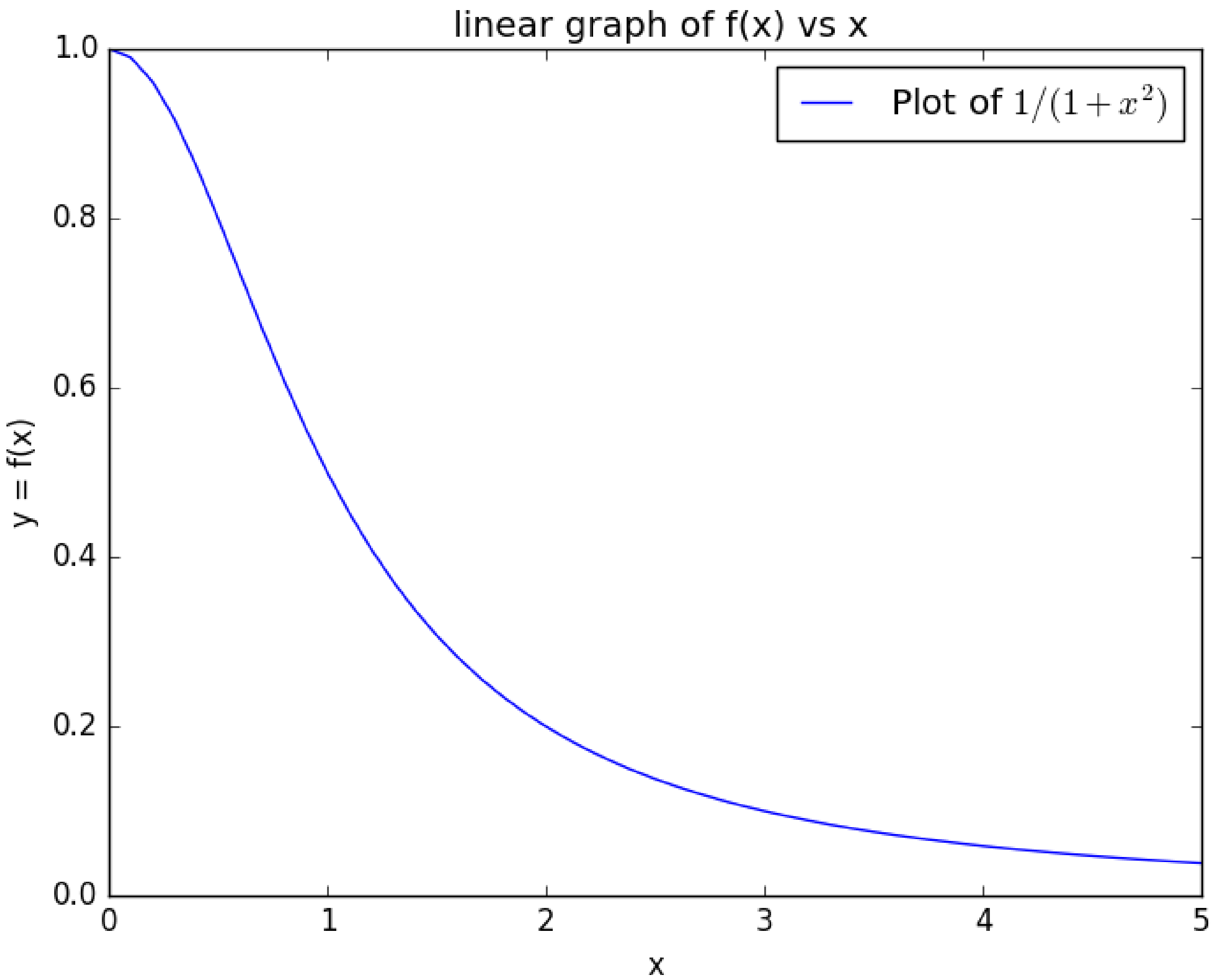
3 plot f(x) vs x

- define function f(x)
- take input x as vector
- plot f(x) vs x using plot function

code:

```
from pylab import *
import matplotlib.pyplot as plt
from numpy import *
def f(x):
    return 1.0/(1+x*x)
x=arange(0,5.1,.1)
y = f(x)
plt.plot(x,y)
plt.ylabel('y = f(x)')
plt.xlabel('x')
plt.title(' linear graph of f(x) vs x')
plt.legend([r'Plot of $1/(1+x^{2})$'])
plt.show()
```

output:



4 finding $tan^{-1}(x)$ using integration and quad function

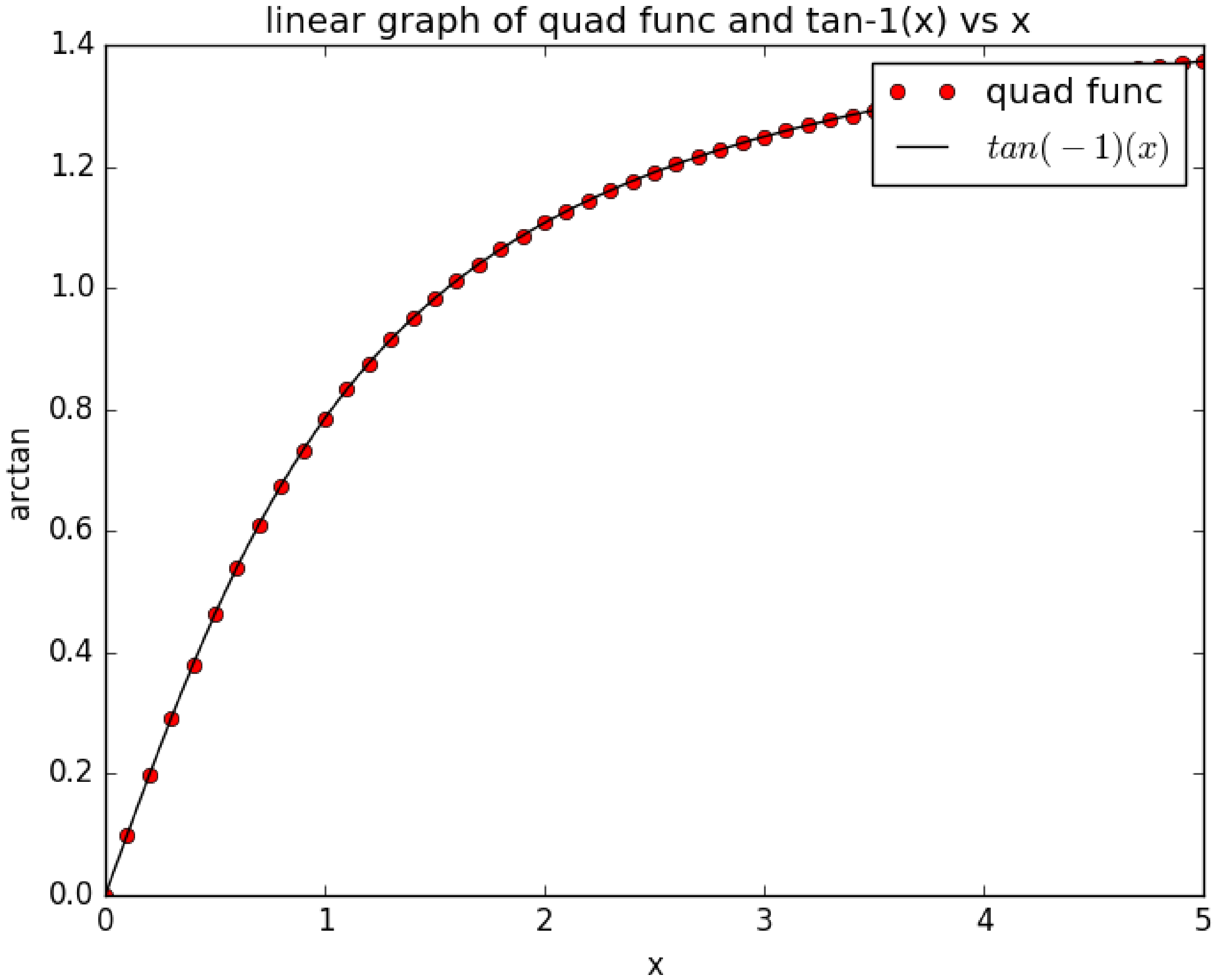
- for each value of x implement quad function .
- append the return values i.e error and integration values to lists.
- tabulate $tan^{-1}x$ and above integration values and plot them.
- plot error values returned by quad function .

code:

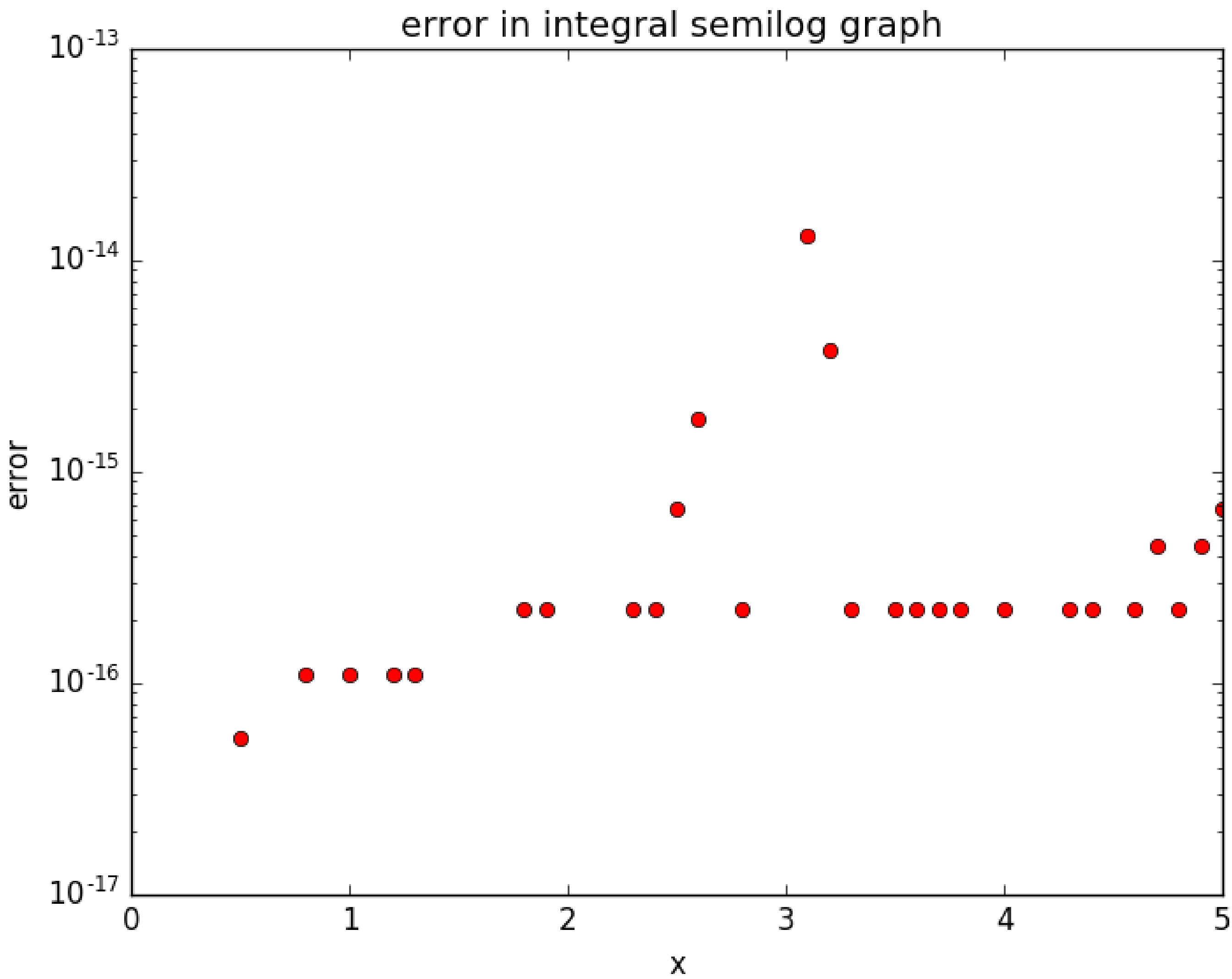
```
from pylab import *
from scipy.integrate import quad
import matplotlib.pyplot as plt
from numpy import *
def func(t):
    return 1.0/(1+t**2)

t=arange(0,5.1,0.1)
l = []
ta =[]
for x in t:
```

```
    temp = quad(func,0,x)[0] #get values
  # temp = round(quad(func,0,x)[0],5) # to round off to 5 decimals
    ta.append( quad(func,0,x)[1]) #get errors
    l.append(temp)

plt.title('linear graph of quad func and tan-1(x) vs x ')
plt.ylabel('arctan')
plt.xlabel('x')
plt.plot(t,y,'ro')
plt.plot(t,y1,'k')
plt.legend(['quad func','$tan(-1)(x)$'])
plt.show()


#uncomment below for error
# y = asarray(l)
# y1 = arctan(t)
# tm =abs( y1 - y)
# tm = asarray(tm)
# plt.title('error in integral semilog graph ')
# plt.ylabel('error')
# plt.xlabel('x')
# plt.semilogy(t,tm,'ro')
# plt.show()
```

output:



linear graph of quad func and tan-1(x) vs x
quad func
$tan(-1)(x)$
arctan
x
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5



5 Trapezoidal algorithm approach for integral

- The integral value between two points by trapezoidal algorithm is calculated by small trapezium areas
- a paticular h is chosed. It is halved for every iteration and exact error , estimated error is calculated
- x ranges form 0 to 1 in steps of h

code:

using cumsum function to implement integral

```
from pylab import *
from numpy import *
def f(x):
    return 1.0/(1+x*x)
h = 0.5
x = arange(1.0,5.0,h)
y = f(x)
I = h*(cumsum(y) - (0.5*(y+y[0])))
print I
```

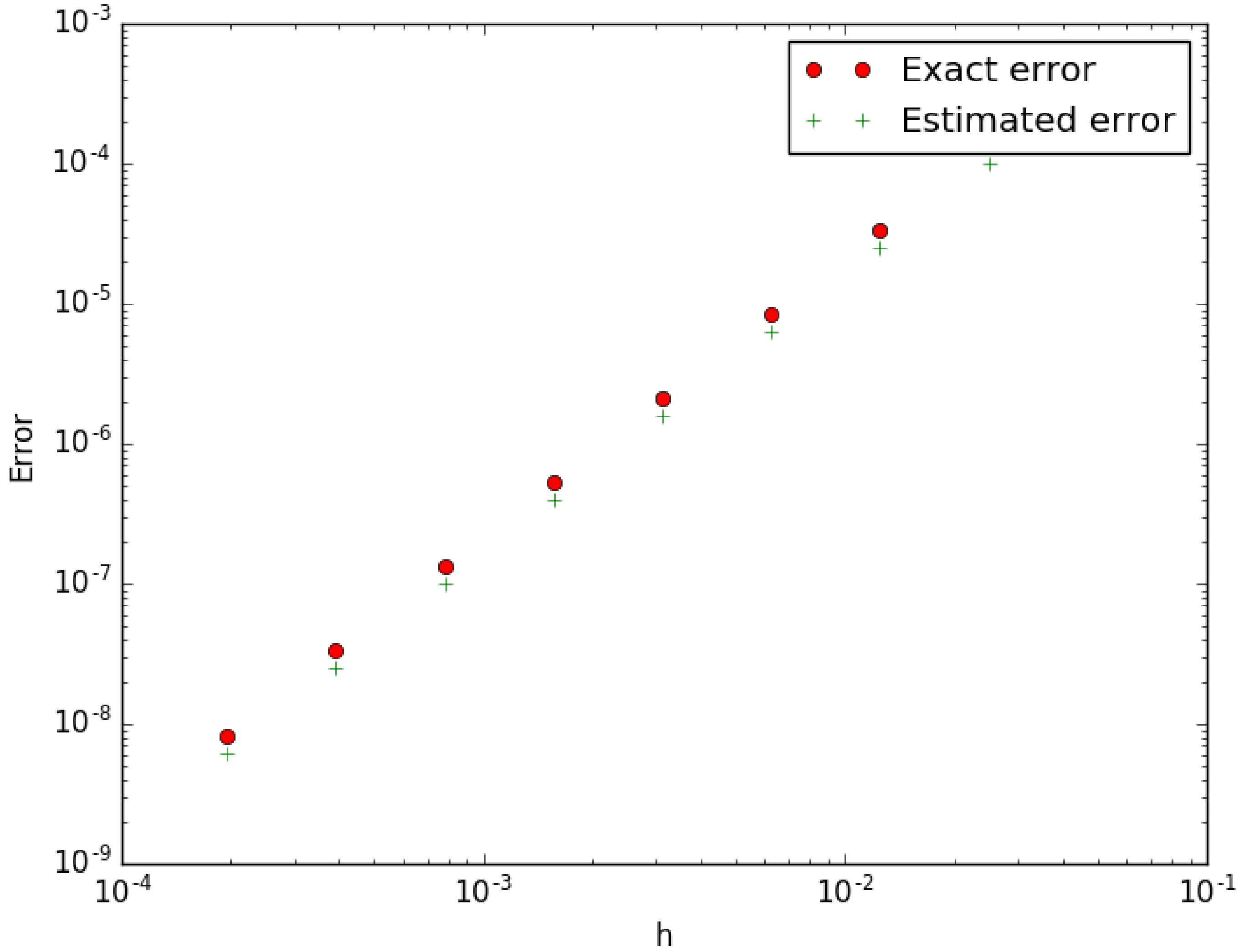
using for loop to implement integrals I(n) = I(n-1) + 0.5*h*y[i]+ 0.5*h*y[i-1]

```
from pylab import *
from numpy import *
def f(x):
    return 1.0/(1+x*x)
h = 0.5
x = arange(1.0,5.0,h)
y = f(x)
I=[0]
for i,v in enumerate(y):
    if(i!=0):
        I.append(I[i-1]+0.5*h*y[i]+0.5*h*y[i-1])

I = asarray(I)
print I
```

```
from pylab import *
import matplotlib.pyplot as plt
from numpy import *
def f(x):
    return 1.0/(1+x*x)

def integral(y,h):
    return h*(cumsum(y) - 0.5*(y+y[0]))

def indices(x,y):
    indices = []
    x = x.tolist()
    y = y.tolist()
    for i in x:
        p = y.index(i)
        indices.append(p)
    indices  = asarray(indices)
    return indices

I_list = []
h_list = []
est_error_list = []
actual_error_list = []
h = 0.1
x = arange(0,1+h,h)
y = f(x)
I_list.append(integral(y,h))
count = 1
while True:
    h = h/2
    h_list.append(h)
    x = arange(0,1+h,h)
    y = f(x)
    I_now = integral(y,h)
    I_list.append(I_now)
    x_prev = arange(0,1+h,2*h)
    indices_list = indices(x_prev,x)
```

```
    I_prev = I_list[count-1]
    I_prev = asarray(I_prev)
    I_now  = asarray(I_now)
    error = abs(I_now[indices_list]-I_prev)
    error = max(error)
    est_error_list.append(error)
    actual_error = abs(arctan(x_prev)-I_prev)
    actual_error = max(actual_error)
    actual_error_list.append(actual_error)
    count+=1
    if(error<10**(-8)):
        break

print est_error_list
print actual_error_list
# print count
print h_list
plt.ylabel('Error')
plt.xlabel('h')
plt.loglog(h_list,actual_error_list,'ro')
plt.loglog(h_list,est_error_list,'g+')
plt.legend(['Exact error','Estimated error'])
plt.show()
```

output:



6 Results and observations

h values	estimated values	exact values
0.05	0.00040584368433571605	0.00054103142650707703
0.025	0.00010139519069252145	0.00013518774217136098
0.0125	2.5373020550945036e-05	3.3830294404291195e-05
0.00625	6.3429741241627369e-06	8.4572738533461589e-06
0.003125	1.585725967423457e-06	2.114299729183422e-06
0.0015625	3.9643479610163013e-07	5.2857963162011856e-07
0.00078125	9.9108631079758425e-08	1.3214483551848843e-07
0.000390625	2.4777187523916666e-08	3.30362490696956e-08
0.0001953125	6.1942947437998441e-09	8.259061545778934e-09

best approximation of h = 0.0001953125