**NC Assignment 02**

**Name**: Bilal Ahmed Khan **Sec:** B **Roll No:** 20K-0183

---------------------------------------------------------------------------------------------------------------------

**a) Numerical Differentiation**

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

def derivative(f,a,method='central',h=0.01):

'''Compute the difference formula for f'(a) with step size h.

Parameters

----------

f : function

Vectorized function of one variable

a : number

Compute derivative at x = a

method : string

Difference formula: 'forward', 'backward' or 'central'

h : number

Step size in difference formula

Returns

-------

float

Difference formula:

central: f(a+h) - f(a-h))/2h

forward: f(a+h) - f(a))/h

backward: f(a) - f(a-h))/h

'''

if method == 'central':

return (f(a + h) - f(a - h))/(2\*h)

elif method == 'forward':

return (f(a + h) - f(a))/h

elif method == 'backward':

return (f(a) - f(a - h))/h

else:

raise ValueError("Method must be 'central', 'forward' or 'backward'.")

**b) Numerical Integration**

**b-i) Closed Newton Cotes Formula**

from scipy.integrate import newton\_cotes

def f(x):

return np.sin(x)

a = 0

b = np.pi

exact = 2

for N in [2, 4, 6, 8, 10]:

x = np.linspace(a, b, N + 1)

an, B = newton\_cotes(N, 1)

dx = (b - a) / N

quad = dx \* np.sum(an \* f(x))

error = abs(quad - exact)

print('{:2d} {:10.9f} {:.5e}'.format(N, quad, error))

**b-ii) Open Newton Cotes Formula**

def integrate(function, a, b):

coeff = [7,32,12,32,7]

result = 0

for i in range(0,len(coeff)):

x = a + (i\*(b-a))/(len(coeff)-1)

result += coeff[i]\*eval(function)

print eval(function)

result = result\*((b-a)/90.)

return result

**b-iii) Composite Trapezoidal Rule**

def trapz(f,a,b,N=50):

'''Approximate the integral of f(x) from a to b by the trapezoid rule.

The trapezoid rule approximates the integral \int\_a^b f(x) dx by the sum:

(dx/2) \sum\_{k=1}^N (f(x\_k) + f(x\_{k-1}))

where x\_k = a + k\*dx and dx = (b - a)/N.

Parameters

----------

f : function

Vectorized function of a single variable

a , b : numbers

Interval of integration [a,b]

N : integer

Number of subintervals of [a,b]

Returns

-------

float

Approximation of the integral of f(x) from a to b using the

trapezoid rule with N subintervals of equal length.

Examples

--------

>>> trapz(np.sin,0,np.pi/2,1000)

0.9999997943832332

'''

x = np.linspace(a,b,N+1) # N+1 points make N subintervals

y = f(x)

y\_right = y[1:] # right endpoints

y\_left = y[:-1] # left endpoints

dx = (b - a)/N

T = (dx/2) \* np.sum(y\_right + y\_left)

return T

**b-iv) Composite Simpson's Rule**

def simps(f,a,b,N=50):

'''Approximate the integral of f(x) from a to b by Simpson's rule.

Simpson's rule approximates the integral \int\_a^b f(x) dx by the sum:

(dx/3) \sum\_{k=1}^{N/2} (f(x\_{2i-2} + 4f(x\_{2i-1}) + f(x\_{2i}))

where x\_i = a + i\*dx and dx = (b - a)/N.

Parameters

----------

f : function

Vectorized function of a single variable

a , b : numbers

Interval of integration [a,b]

N : (even) integer

Number of subintervals of [a,b]

Returns

-------

float

Approximation of the integral of f(x) from a to b using

Simpson's rule with N subintervals of equal length.

Examples

--------

>>> simps(lambda x : 3\*x\*\*2,0,1,10)

1.0

'''

if N % 2 == 1:

raise ValueError("N must be an even integer.")

dx = (b-a)/N

x = np.linspace(a,b,N+1)

y = f(x)

S = dx/3 \* np.sum(y[0:-1:2] + 4\*y[1::2] + y[2::2])

return S

**b-v) Composite Midpoint Formula**

from trapezoidal import trapezoidal

from midpoint import midpoint

from math import exp

g = lambda y: exp(-y\*\*2)

a = 0

b = 2

print ' n midpoint trapezoidal'

for i in range(1, 21):

n = 2\*\*i

m = midpoint(g, a, b, n)

t = trapezoidal(g, a, b, n)

print '%7d %.16f %.16f' % (n, m, t)