P346 Comp Phy Lab Project report - Adaptive Quadrature Methods

Chandan Kumar Sahu - 1911055

Numerical Integration -

1. Simpson's $\frac{1}{3}$ method

Here we use three methods of numerical quadrature: Mid-point, Trapezoidal, and Simpson's $\frac{1}{3}$ method

In [8]:

```
# Importing necessary library files

import math
```

In [9]:

```
# Function to calculate the number of iterations which will give
# correct integration value upto eps number of decimal places

def calculate_N(fn_mp, fn_t, fn_s, eps=10**-6):

    # Calculation of N from error calculation formula
    N_mp=((b-a)**3/24/eps*fn_mp)**0.5
    N_t=((b-a)**3/12/eps*fn_t)**0.5
    N_s=((b-a)**5/180/eps*fn_s)**0.25

    # Using integral value, also handling the case where eps=0
    if N_mp==0:
        N_mp=1
    else:
        N_mp=int(N_mp)+1

    if N_t==0:
        N_t=1
    else:
        N_t=int(N_t)+1

    if N_s==0:
        N_s=1
    else:
        N_s=int(N_s)+1

    # Special case with simpson's rule
    # Simpson rule always requires even N_s, increase it by one if it comes odd
    if N_s%2!=0:
        N_s+=1

    return N_mp, N_t, N_s



# numerical integration by mid-point method
def int_mid_point(f, a, b, n):
    Sum=0
    h=(b-a)/n # step size

    # integration algorithm
```

```
    for i in range(1,n+1):
        x=a+(2*i-1)*h/2
        Sum+=f(x)

    return Sum*h



# numerical integration by Trapezoidal method
def int_trapezoidal(f, a, b, n):
    Sum=0
    h=(b-a)/n # step size

    # integration algorithm
    for i in range(1,n+1):
        Sum+=f(a+i*h)+f(a+(i-1)*h)

    return Sum*h/2



# numerical integration by Simpson method
def int_simpson(f, a, b, n):
    Sum=f(a)+f(b)
    h=(b-a)/n

    # integration algorithm
    for i in range(1,n):
        if i%2!=0:
            Sum+=4*f(a+i*h)
        else:
            Sum+=2*f(a+i*h)

    return Sum*h/3
```

2. Adaptive quadrature for simpson's method

Here we apply the adaptive method on all three methods: Mid-point, Trapezoidal, and Simpson's $\frac{1}{3}$ method

In [10]:

```
# numerical integration by adaptive quadrature Simpson method

def int_adap_mid_point(f, a, b, eps, count=[]):
    mp1=int_mid_point(f, a, b, 1)
    mp2=int_mid_point(f, a, b, 2)
    if abs(mp2-mp1) < 3*eps:
        answer = mp2 + (mp2-mp1)/3
    else:
        c = (a+b)/2
        count.append(1)
        L, c1 = int_adap_mid_point(f,a,c,eps/2, count)
        R, c2 = int_adap_mid_point(f,c,b,eps/2, count)
        answer = L+R
    return answer, len(count)



# numerical integration by adaptive quadrature Simpson method

def int_adap_trap(f, a, b, eps, count=[]):
    tr1=int_trapezoidal(f, a, b, 1)
    tr2=int_trapezoidal(f, a, b, 2)
    if abs(tr2-tr1) < 3*eps:
        answer = tr2 + (tr2-tr1)/3
    else:
        c = (a+b)/2
        count.append(1)
        L, c1 = int_adap_trap(f,a,c,eps/2, count)
        R, c2 = int_adap_trap(f,c,b,eps/2, count)
        answer = L+R
    return answer, len(count)



# numerical integration by adaptive quadrature Simpson method

def int_adap_simp(f, a, b, eps, count=[]):
    s1=int_simpson(f, a, b, 2)
    s2=int_simpson(f, a, b, 4)
    if abs(s2-s1) < 15*eps:
        answer = s2 + (s2-s1)/15
```

```
    else:
        c = (a+b)/2
        count.append(1)
        L, c1 = int_adap_simp(f,a,c,eps/2, count)
        R, c2 = int_adap_simp(f,c,b,eps/2, count)
        answer = L+R
    return answer, len(count)
```

In [12]:

```
def func(x):
    return math.exp(-x)*math.cos(5*x)

a=0
b=6
eps=10**-6

# feed here maximum of second derivative of function for Mid-point
fn_mp=24
# feed here maximum of second derivative of function for trapezoidal
fn_t=24
# feed here maximum of fourth derivative of function for simpson
fn_s=476

N_mp, N_t, N_s = calculate_N(fn_mp, fn_t, fn_s, eps)

int_mp = (int_mid_point (func, a, b, N_mp))
adap_mp, count_mp = int_adap_mid_point (func,a,b,eps, count=[])

int_tr = (int_trapezoidal (func, a, b, N_t))
adap_tr, count_tr = int_adap_trap (func,a,b,eps, count=[])

int_sim = int_simpson (func, a, b, N_s)
adap_sim, count_sim = int_adap_simp (func,a,b,eps, count=[])

print("\nUsing normal mid point method      =   "+str(int_mp)+"     with iterations  =   "+str(N_mp))
print("Using adaptive quadrature method   =   "+str(adap_mp)+"     with iterations  =   "+str(count_mp))

print("\nUsing normal trapezoidal method    =   "+str(int_tr)+"     with iterations  =   "+str(N_t))
print("Using adaptive quadrature method   =   "+str(adap_tr)+"     with iterations  =   "+str(count_tr))

print("\nUsing normal simpson's method      =   "+str(int_sim)+"    with iterations  =   "+str(N_s))
print("Using adaptive quadrature method   =   "+str(adap_sim)+"    with iterations  =   "+str(count_sim))
```

```
Using normal mid point method      =   0.03797584763433155     with iterations  =   14697
Using adaptive quadrature method   =   0.03797585488193267     with iterations  =   2668

Using normal trapezoidal method    =   0.03797586168766671     with iterations  =   20785
Using adaptive quadrature method   =   0.03797585479357686     with iterations  =   3759

Using normal simpson's method      =   0.037975828996023885    with iterations  =   380
Using adaptive quadrature method   =   0.037975850426703664    with iterations  =   63
```

In [ ]: