SciComp2-M16

October 3, 2021

0.0.1 22.1

Use order of h8 Romberg integration to evaluate

$\int_0^3 \mathrm{x}^2 \mathrm{e}^x \mathrm{d}x$

Compare et and ea

[1]:
```
import math
```

[2]:
```
def real_integral(F, a, b):
    """
    Calculate integral of f based on PT6.1 in Numerical Methods Book
    F(x)(a-b) = F(b) - F(a)
    """
    I = F(b) - F(a)
    return I

def multi_trapezoidal(f, F, a, b, n):
    """
    f: function of x
    F: integral function of x
    a: lower value of Integral
    b: upper value of Integral
    n: number of segments for multiple-application calculation
    """
    #print(f(0))
    xs = [a]
    h = (b-a)/n

    for _ in range(n):
        xs.append(round(xs[-1] + h, 4))
    #print('x values: ', xs)

    ys = []
    for x in xs:
        ys.append(round(f(x),4))

    #print('f values: ', ys)
```

```
    m = len(ys)

    #print(sum(ys[1:m-1]))

    estI = (b - a)*(ys[0] + 2*sum(ys[1:m-1]) + ys[-1])/(2*n)
    print(f'Integrals by multi_trapezoidal {n} seqgments: {estI}')

    return estI

def romberg_algo(j, k, Ijk_1, Ij1k_1):
    numerator = 4**(k-1)*Ij1k_1 - Ijk_1
    denominator = 4**(k-1)- 1
    #print(numerator, denominator)
    return numerator*1.0/denominator

def romberg_integration_h8(max_iter, f, F, a, b): # k=4, O(h8)
    n = 1
    max_iter = max_iter
    O_h = []
    for j in range(max_iter): # j goes from 1 to max_iter
        Ijk_1 = multi_trapezoidal(f, F, a, b, n)
        n *= 2
        Ij1k_1 = multi_trapezoidal(f, F, a, b, n)
        Ijk = romberg_algo(j, 2, Ijk_1, Ij1k_1)
        O_h.append(Ijk)

    I13 = romberg_algo(1, 3, O_h[0], O_h[1])
    I23 = romberg_algo(2, 3, O_h[1], O_h[2])
    #print(I13, I23)
    I14 = romberg_algo(1, 4, I13, I23)

    realI = real_integral(F, a, b)
    print(f'Real integral: {realI}, O(h8): ', I14)

    Et = round(realI - I14, 2)
    et = round(abs(Et)*100/realI,2)
    ea = abs((I14 - I23)/I14)*100

    print(f'Absolute Error: {Et}, Percent Relative Error: {et}%, Estimate of␣
 ↪Percent Relative Error: {ea}%')
```

[3]:
```
#### -> Apply romberg h8 on Textbook example
def f(x):
    return 0.2 + 25*x - 200*(x**2) + 675*(x**3) - 900*(x**4) + 400*(x**5)

def F(x):
```

```
    return 0.2*x + 25/2*(x**2) - 200/3*(x**3) + 675/4*(x**4) - 900/5*(x**5) +
↪400/6*(x**6)

romberg_integration_h8(3, f, F, 0, 0.8)
```

```
Integrals by multi_trapezoidal 1 seqgments: 0.17280000000000004
Integrals by multi_trapezoidal 2 seqgments: 1.0688000000000002
Integrals by multi_trapezoidal 2 seqgments: 1.0688000000000002
Integrals by multi_trapezoidal 4 seqgments: 1.4848
Integrals by multi_trapezoidal 4 seqgments: 1.4848
Integrals by multi_trapezoidal 8 seqgments: 1.6008
Real integral: 1.6405333333333374, O(h8):  1.6405333333333334
Absolute Error: 0.0, Percent Relative Error: 0.0%, Estimate of Percent Relative
Error: 0.0%
```

[4]:
```
#### -> Apply romberg h8 on Problem 22.1
def f(x):
    return (x**2)*(math.exp(x))

def F(x):
    return (x**2 - 2*x + 2)*(math.exp(x))

romberg_integration_h8(3, f, F, 0, 3)
```

```
Integrals by multi_trapezoidal 1 seqgments: 271.1547
Integrals by multi_trapezoidal 2 seqgments: 150.70305
Integrals by multi_trapezoidal 2 seqgments: 150.70305
Integrals by multi_trapezoidal 4 seqgments: 112.26840000000001
Integrals by multi_trapezoidal 4 seqgments: 112.26840000000001
Integrals by multi_trapezoidal 8 seqgments: 101.94041250000001
Real integral: 98.42768461593835, O(h8):  98.4293126984127
Absolute Error: -0.0, Percent Relative Error: 0.0%, Estimate of Percent Relative
Error: 0.004569067347927538%
```

0.0.2 23.1 (not completed)

Compute forward and backward difference approximations of O(h) and O(h2), and central difference approximations of O(h2) and O(h4) for the first derivative of y = cos x at x = pi/4 using a value of h = p/12.

Estimate the true percent relative error |et| for each approximation.

[5]:
```
def high_accuracy_differentiation(f, fprime, x, h):
    xEst = []
    for i in range(-2, 3):
        xEst.append(x+i)

    print(xEst)
```

```
    print('O(h1)')
    mid = int(len(xEst)//2)
    print(mid)
    est = xEst[mid]
    fw_est = xEst[mid+1]
    bw_est = xEst[mid-1]

    fw_est_derivative = round((fw_est - est)/h, 3)
    bw_est_derivative = round((est - bw_est)/h, 3)
    center_est_derivative = round((fw_est - bw_est)/2*h, 3)

    true_derivative = fprime(x)

    fw_et = round(abs(fw_est_derivative - true_derivative)*100.0/
 ↪true_derivative, 2)
    bw_et = round(abs(bw_est_derivative - true_derivative)*100.0/
 ↪true_derivative, 2)
    center_et = round(abs(center_est_derivative - true_derivative)*100.0/
 ↪true_derivative, 2)
    print(fw_est_derivative, bw_est_derivative, center_est_derivative)
    print(f'{fw_et}%, {bw_et}%, {center_et}%')
```

[6]:
```
#### -> Apply High-Accuracy Differentiation Formulas on Textbook example
def f(x):
    return round(-0.1*(x**4) - 0.15*(x**3) - 0.5*(x**2) - 0.25*x + 1.2, 4)

def fprime(x):
    return -0.4*(x**3) - 0.45*(x**2) - 1.0*x - 0.25

high_accuracy_differentiation(f, fprime, 0.5, 0.25)
```

```
[-1.5, -0.5, 0.5, 1.5, 2.5]
O(h1)
2
4.0 4.0 0.25
-538.36%, -538.36%, -127.4%
```

[ ]: