DS288 (AUG) 3:0 Numerical Methods

Naman Pesricha
namanp@iisc.ac.in
SR - 24115

Homework-5

Q1 Derive Simpsons Rule with error term by using

$$\int_{x_0}^{x_2} f(x)\,dx = a_0 f(x_0) + a_1 f(x_1) + a_2 f(x_2) + k f^{(4)}(\xi)$$

Find a_0, a_1, and a_2 from the fact that Simpson's rule is exact for $f(x) = x^n$ when n = 0, 1, 2, and 3. Then find k by applying the integration formula to f(x) = x^4. [3 points]

We use equispaced points for Simpson's rule $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$. We can substitute these values to simplify our system of equations. We have the following 4 equations

$$\int_{x_0}^{x_2} x^0\,dx = a_0 + a_1 + a_2 = x_2 - x_0 = 2h \tag{1}$$

$$\int_{x_0}^{x_2} x^1\,dx = a_0 x_0 + a_1\left(h + x_0\right) + a_2\left(2h + x_0\right) = -\frac{x_0^2}{2} + \frac{\left(2h + x_0\right)^2}{2} \tag{2}$$

$$\int_{x_0}^{x_2} x^2\,dx = a_0 x_0^2 + a_1\left(h + x_0\right)^2 + a_2\left(2h + x_0\right)^2 = -\frac{x_0^3}{3} + \frac{\left(2h + x_0\right)^3}{3} \tag{3}$$

$$\int_{x_0}^{x_2} x^3\,dx = a_0 x_0^3 + a_1\left(h + x_0\right)^3 + a_2\left(2h + x_0\right)^3 = -\frac{x_0^4}{4} + \frac{\left(2h + x_0\right)^4}{4} \tag{4}$$

Using equation 2 , 3 and 4 we get our system of equations :

$$X = \begin{bmatrix} x_0 & h + x_0 & 2h + x_0 \\ x_0^2 & \left(h + x_0\right)^2 & \left(2h + x_0\right)^2 \\ x_0^3 & \left(h + x_0\right)^3 & \left(2h + x_0\right)^3 \end{bmatrix} \; b = \begin{bmatrix} -\frac{x_0^2}{2} + \frac{(2h+x_0)^2}{2} \\ -\frac{x_0^3}{3} + \frac{(2h+x_0)^3}{3} \\ -\frac{x_0^4}{4} + \frac{(2h+x_0)^4}{4} \end{bmatrix} \; a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

$$Xa = b \implies a = X^{-1} b$$

$$a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} x_0 & h + x_0 & 2h + x_0 \\ x_0^2 & \left(h + x_0\right)^2 & \left(2h + x_0\right)^2 \\ x_0^3 & \left(h + x_0\right)^3 & \left(2h + x_0\right)^3 \end{bmatrix}^{-1} \begin{bmatrix} -\frac{x_0^2}{2} + \frac{(2h+x_0)^2}{2} \\ -\frac{x_0^3}{3} + \frac{(2h+x_0)^3}{3} \\ -\frac{x_0^4}{4} + \frac{(2h+x_0)^4}{4} \end{bmatrix}$$

$$a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \frac{2h^2+3hx_0+x_0^2}{2h^2x_0} & \frac{-3h-2x_0}{2h^2x_0} & \frac{1}{2h^2x_0} \\ \frac{-2hx_0-x_0^2}{h^3+h^2x_0} & \frac{2}{h^2} & -\frac{1}{h^3+h^2x_0} \\ \frac{hx_0+x_0^2}{4h^3+2h^2x_0} & \frac{-h-2x_0}{4h^3+2h^2x_0} & \frac{1}{4h^3+2h^2x_0} \end{bmatrix} \begin{bmatrix} -\frac{x_0^2}{2} + \frac{(2h+x_0)^2}{2} \\ -\frac{x_0^3}{3} + \frac{(2h+x_0)^3}{3} \\ -\frac{x_0^4}{4} + \frac{(2h+x_0)^4}{4} \end{bmatrix} = \begin{bmatrix} \frac{h}{3} \\ \frac{4h}{3} \\ \frac{h}{3} \end{bmatrix}$$

$$a_0 = \frac{h}{3},\; a_1 = \frac{4h}{3},\; \text{and } a_2 = \frac{h}{3}$$

These are computed using eq 2, 3 and 4 and also satisfies equation 1 : $a_0 + a_1 + a_2 = \frac{h}{3} + \frac{4h}{3} + \frac{h}{3} = 2h$. Hence it satisfies all 4 equations.

For f(x) = x^4, we get $\boxed{f^4(\xi) = 4 * 3 * 2 * 1 * x^0 = 24}$, therefore we get the following equation:

$$\int_{x_0}^{x_2} x^4\,dx = a_0 x_0^4 + a_1\left(h+x_0\right)^4 + a_2\left(2h+x_0\right)^4 + 24k = -\frac{x_0^5}{5} + \frac{\left(2h+x_0\right)^5}{5} \tag{5}$$

$$\frac{h}{3}x_0^4 + \frac{4h}{3}\left(h+x_0\right)^4 + \frac{h}{3}\left(2h+x_0\right)^4 + 24k = -\frac{x_0^5}{5} + \frac{\left(2h+x_0\right)^5}{5}$$

$$24k = -\frac{x_0^5}{5} + \frac{\left(2h+x_0\right)^5}{5} - \frac{h}{3}x_0^4 - \frac{4h}{3}\left(h+x_0\right)^4 - \frac{h}{3}\left(2h+x_0\right)^4$$

Simplifying LHS will yield

$$24k = -\frac{x_0^5}{5} + \frac{32h^5}{5} + 16h^4x_0 + 16h^3x_0^2 + 8h^2x_0^3 + 2hx_0^4 + \frac{x_0^5}{5} - \left[\frac{hx_0^4}{3}\right] - \left[\frac{4h^5}{3} + \frac{16h^4x_0}{3} + 8h^3x_0^2 + \frac{16h^2x_0^3}{3} + \frac{4hx_0^4}{3}\right] - \left[\frac{16h^5}{3} + \frac{32h^4x_0}{3} + 8h^3x_0^2 + \frac{8h^2x_0^3}{3} + \frac{hx_0^4}{3}\right]$$

All terms will get cancelled except the coefficient of h^5.

$$24k = \frac{32h^5}{5} - \frac{16h^5}{3} - \frac{4h^5}{3} = \frac{-4}{15}h^4$$

$$\boxed{k = \frac{-h^5}{90}}$$

Therefore the final Simpson's rule with error term is:

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{h}{3}f(x_0) + \frac{4h}{3}f(x_1) + \frac{h}{3}f(x_2) - \frac{h^5}{90}f^{(4)}(\xi)$$

Q2 Apply Romberg Integration to the following integrals until $R_{n-1,n-1}$ and $R_{n,n}$ agree to within 10^{-5}. Report the value of n and the number of function evaluations. Also, compute the result to that obtained from the Trapezoidal rule for the same number n (note that you already calculate this value to get $R_{n,n}$). [3.5 points]

$$\textbf{(a)} \quad \int_0^1 x^{1/3}\,dx; \quad \textbf{(b)} \int_0^1 x^2 e^{-x}\,dx$$

Note: For this question, I will be following the class notes' convention where the n starts from 0 [in the text book. n starts from 1]

Integral	Romberg = $R_{n,n}$	n	True Value	$\epsilon_{n,n}$	#Function Evaluations
$\int_0^1 x^{1/3}\,dx$	**0.749995**	11	0.750000	0.000005	**2049**
$\int_0^1 x^2 e^{-x}\,dx$	**0.160603**	3	0.160603	0.000000	**9**

Table 1: Romberg Integral for a) and b) with true values, errors, and function evaluations.

The number of unique function evaluations is the same as that of a Trapezoidal Rule for n, i.e, $2^n + 1$.

Integral	n	Trapezoidal = $R_{n,0}$	True Value	$\epsilon_{n,0}$	h
$\int_0^1 x^{1/3}\,dx$	11	0.749989	0.750000	0.000011	0.000488
$\int_0^1 x^2 e^{-x}\,dx$	3	0.161080	0.160603	0.000477	0.125000

Table 2: Trapezoidal Integral with true values, errors, and step sizes for each integral.

In both the cases, $\boxed{\epsilon_{n,n} < \epsilon_{n,0}}$. This checks out with what we studied in class as $\epsilon_{n,n}$ is $O(h^{2n+2})$ and $\epsilon_{n,0} = O(h^2)$. An attempt was made to evaluate the ratio of $\frac{\log \epsilon_{n,n}}{\log \epsilon_{n,0}}$ which should've been $\approx 2n$, but results were not consistent probably because of small value of h.

It is also noticed that $\int_0^1 x^{1/3}\,dx$ converged slower than $\int_0^1 x^2 e^{-x}\,dx$ even though the former has a simpler form. This is because Rhomberg Integration is based on trapezoidal rule, which integrates the function by linearly interpolating points (x_i, f_i) and (x_{i+1}, f_{i+1}).

From the figure, we can see that for $x \in [0, 0.2]$, the linear interpolation will be very inaccurate for large h and hence the slow convergence.

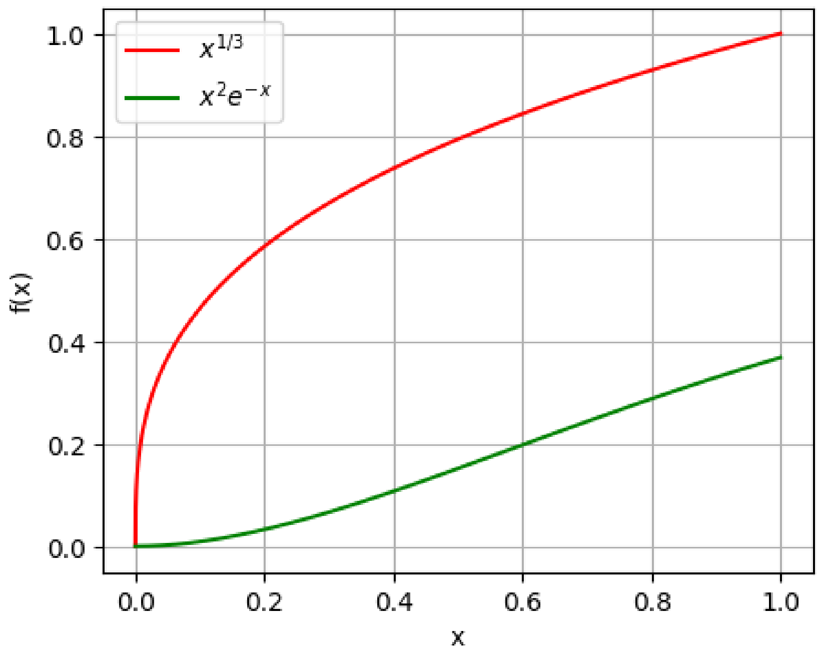

Figure 1: $f(x) = x^{1/3}$ and $f(x) = x^2 e^{-x}$

Q3 Approximate the integrals in Problem 2(a) and 2(b) using Gaussian Quadrature with n=2, 3, 4, and 5. Report the number of function evaluations and compare the results with those obtained using Romberg Integration in Problem-2. [3.5 points]

n	$GQ_n(x^{1/3})$	$GQ_n(x^2e^{-x})$	# function evaluations	$e_n(x^{1/3})$	$e_n(x^2e^{-x})$
2	0.759778	0.159410	2	0.009778	0.001193
3	0.753855	0.160595	3	0.003855	0.000008
4	0.751946	0.160603	4	0.001946	0.000000
5	0.751132	0.160603	5	0.001132	0.000000

Table 3: Gaussian Quadrature for , corresponding function evaluations, and error estimates.

Since the number of function evaluations is 2,3,4 and 5 for Gaussian Quadrature, we will these values to Romberg's $n = 0$, 1 and 2 (corresponding to 2, 3 and 5 function evaluations respectively.)

n	$R_{n,n}(x^{1/3})$	$R_{n,n}(x^2e^{-x})$	# function evaluations	$\epsilon_{n,n}(x^{1/3})$	$\epsilon_{n,n}(x^2e^{-x})$
0	0.500000	0.183940	2	0.250000	0.023337
1	0.695800	0.162402	3	0.054200	0.001799
2	0.730634	0.160611	5	0.019366	0.000008

Table 4: Table with n, $R_{n,n}$, function evaluations, and absolute errors.

#F.E	$\epsilon_{n,n}(x^{1/3})$ (R)	$\epsilon_n(x^{1/3})$ (GQ)	$\epsilon_{n,n}(x^2e^{-x})$ (R)	$\epsilon_n(x^2e^{-x})$ (GQ)
2	0.250000	$\underline{\mathbf{0.009778}_L}$	0.023337	$\underline{\mathbf{0.001193}_L}$
3	0.054200	$\underline{\mathbf{0.003855}_L}$	0.001799	$\underline{\mathbf{0.000008}_L}$
5	0.019366	$\underline{\mathbf{0.001132}_L}$	0.000008	$\underline{\mathbf{0.000000}_L}$

Table 5: Comparison of error values for function evaluations 2, 3, and 5, with errors from the Gaussian Quadrature (GQ) and Romberg (R) methods. The lesser error is marked in underline bold with subscript $_L$.

From Table 5 it is evident that **for the same number of function evaluations, Gaussian Quadrature gives lesser error than Romberg integration** for both the problems.

Also to be noted that for problem a), **to beat the gaussian quadrature corresponding to <u>5 function evaluations</u>, Romberg requires n = 6 ($\epsilon_{6,6} = 0.000465$) that corresponds to <u>65 evaluations</u>!**

```python
# %% [markdown]
# ## Q2

# %%
import numpy as np
import sympy as sp
import pandas as pd
import matplotlib.pyplot as plt

# %matplotlib inline


# %%
x = sp.symbols('x')
func_a = x**(1/3)
func_b = (x**2)*sp.exp(-x)

f_a = sp.lambdify((x), func_a)
f_b = sp.lambdify((x), func_b)

integral_func_a = sp.integrate(func_a, x)
integral_func_b = sp.integrate(func_b, x)

print(integral_func_a, "|" , integral_func_b)
true_integral_f_a = sp.lambdify((x), integral_func_a)
true_integral_f_b = sp.lambdify((x), integral_func_b)

print(f"True Integral f_a : {true_integral_f_a(1) - true_integral_f_a(0)}")
print(f"True Integral f_b : {true_integral_f_b(1) - true_integral_f_b(0)}")

# %%
trapezoidal_rule = lambda f, a, b : (f(a) + f(b)) * (b-a)/2

# f_a = lambda x: x**(1/3)
# f_b = lambda x: (x**2)*np.exp(-x)

call_set = set()

def composite_trapezoidal_rule(
        f: callable,
        a: np.float64,
        b: np.float64,
        m: np.int64
):
    integral = 0
    panels = 2**(m)
    h = (b-a)/panels

    for panel in range(1, panels+1):
        integral += trapezoidal_rule(f, (panel-1)*h, panel*h)
        call_set.add(f"f_{(panel-1)*h}")
        call_set.add(f"f_{(panel)*h}")


    return integral

def romberg_integration(f: callable, a: np.float64, b: np.float64, m_upper_limit: int = 50) -> np.ndarray:
    R = np.zeros((m_upper_limit, m_upper_limit))

    for m in range(m_upper_limit):
        R[m, 0] = composite_trapezoidal_rule(f, a, b, m)
        for j in range(1, m + 1):
            R[m, j] = ( (4**j) * R[m, j-1] - R[m-1, j-1] ) / ( 4**j - 1 )

        print(abs(R[m, m] - R[m-1, m-1]))
        if m > 0 and abs(R[m, m] - R[m-1, m-1]) < 1e-5:
            return R[:m+1, :m+1]

    return R


# %%
call_set.clear()
R_f_a = pd.DataFrame(romberg_integration(f_a, 0, 1))
print(f"# unique function evaluations calls {len(call_set)}")
R_f_a


# %%
```

```python
call_set.clear()
R_f_b = pd.DataFrame(romberg_integration(f_b, 0, 1))
print(f"# unique function evaluations calls {len(call_set)}")
R_f_b


# %%
def func_evaluations_romberg(m):
    return 2**m + 1

print("Function eval calls from formula for m = 11:", func_evaluations_romberg(11))
print("Function eval calls from formula for m = 3 : ", func_evaluations_romberg(3))

# %%
x_values = np.linspace(0,1,10000)
f_a_values = f_a(x_values)
f_b_values = f_b(x_values)

plt.figure(figsize=(5,4))
plt.plot(x_values, f_a_values, label=r'$x^{1/3}$', color = 'red')
plt.plot(x_values, f_b_values, label=r"$x^2 e^{-x}$", color = 'green')
plt.legend()
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid()
plt.show()

# %% [markdown]
# ## Q3

# %%
# Initialize the LEGENDRE constants

LEGENDRE = {
    2: {
        'roots': [-0.5773502692, 0.5773502692],
        'weights': [1.0, 1.0]
    },
    3: {
        'roots': [-0.7745966692, 0.0, 0.7745966692],
        'weights': [0.5555555556, 0.8888888889, 0.5555555556]
    },
    4: {
        'roots': [-0.8611363116, -0.3399810436, 0.3399810436, 0.8611363116],
        'weights': [0.3478548451, 0.6521451549, 0.6521451549, 0.3478548451]
    },
    5: {
        'roots': [-0.9061798459, -0.5384693101, 0.0, 0.5384693101, 0.9061798459],
        'weights': [0.2369268851, 0.4786286705, 0.5688888889, 0.4786286705, 0.2369268851]
    }
}

yi = lambda xi, a, b : (b+a)/2 + xi*(b-a)/2

def gaussian_quadrature( f: callable, a: np.float64, b: np.float64, m: int) -> np.float64:
    integral = 0
    roots = LEGENDRE[m]['roots']
    weights = LEGENDRE[m]['weights']

    for i in range(len(roots)):
        adjusted_root = yi(roots[i], a, b)
        f_at_root = f(adjusted_root)
        integral += f_at_root * weights[i]

    integral = integral * (b-a)/2

    return integral

# %%
gaussian_outputs = pd.DataFrame(data= np.zeros((4,2)), index= (2, 3, 4, 5), columns= ('f(a)', 'f(b)'))

for i in (2, 3, 4, 5):
    gaussian_outputs.loc[i, 'f(a)'] = gaussian_quadrature(f_a, 0, 1, i)
    gaussian_outputs.loc[i, 'f(b)'] = gaussian_quadrature(f_b, 0, 1, i)

gaussian_outputs['#evals'] = gaussian_outputs.index
gaussian_outputs

# %%
gaussian_outputs['e_n(a)'] = np.abs(np.round(gaussian_outputs['f(a)'], 6) - 0.75)
gaussian_outputs['e_n(b)'] = np.abs(np.round(gaussian_outputs['f(b)'], 6)- 0.160603)
```

```
# %%
gaussian_outputs

# %%
R_f_a - 0.75

# %%
2**6 + 1
```