DS228: Numerical Methods
Assignment 5

November 18, 2024

Sandeep Kumar
24022

---

1. Considering that Simpson's rule is accurate for $f(x) = x^n$ when $n = 0, 1, 2, 3$, we derive the following expressions:

$$\int_{x_0}^{x_2} dx = a_0 + a_1 + a_2,$$
$$\int_{x_0}^{x_2} x\,dx = a_0x_0 + a_1x_1 + a_2x_2,$$
$$\int_{x_0}^{x_2} x^2\,dx = a_0x_0^2 + a_1x_1^2 + a_2x_2^2,$$
$$\int_{x_0}^{x_2} x^3\,dx = a_0x_0^3 + a_1x_1^3 + a_2x_2^3.$$

On solving the integrals on the left-hand side, we obtain:

$$\frac{x_2 - x_0}{2} = a_0 + a_1 + a_2,$$
$$\frac{x_2^2 - x_0^2}{2} = a_0x_0 + a_1x_1 + a_2x_2,$$
$$\frac{x_2^3 - x_0^3}{3} = a_0x_0^2 + a_1x_1^2 + a_2x_2^2,$$
$$\frac{x_2^4 - x_0^4}{4} = a_0x_0^3 + a_1x_1^3 + a_2x_2^3.$$

These equations can be represented in a matrix form as follows:

$$\begin{bmatrix} x_0 & x_1 & x_2 \\ x_0^2 & x_1^2 & x_2^2 \\ x_0^3 & x_1^3 & x_2^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \frac{x_2^2 - x_0^2}{2} \\ \frac{x_2^3 - x_0^3}{3} \\ \frac{x_2^4 - x_0^4}{4} \end{bmatrix}.$$

The solution for the coefficients a_0, a_1, and a_2 can then be written as:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_0^2 & x_1^2 & x_2^2 \\ x_0^3 & x_1^3 & x_2^3 \end{bmatrix}^{-1} \begin{bmatrix} \frac{x_2^2 - x_0^2}{2} \\ \frac{x_2^3 - x_0^3}{3} \\ \frac{x_2^4 - x_0^4}{4} \end{bmatrix}.$$

If x_0, x_1, and x_2 are equally spaced, then we substitute:

$$x_1 = x_0 + h, \quad x_2 = x_0 + 2h.$$

With this, the coefficients simplify to:

$$a_0 = \frac{h}{3}, \quad a_1 = \frac{4h}{3}, \quad a_2 = \frac{h}{3}.$$

Hence, Simpson's rule is established with these expressions.

Verification

To verify, substitute the calculated values of a_0, a_1, and a_2 into the integral equations. We find that:

$$x_2 - x_0 = h, \quad \frac{4h}{3} + \frac{h}{3} + \frac{h}{3} = 2h,$$

demonstrating consistency across all equations. For $n = 4$, the fourth equation becomes:

$$\int_{x_0}^{x_2} x^4 \, dx = a_0 x_0^4 + a_1 x_1^4 + a_2 x_2^4 + 24k.$$

Since $f^{(4)}(x) = 24$ for all $x \in \mathbb{R}$, the integral simplifies as:

$$k = -\frac{(x_1 - x_0)^5}{90} = -\frac{h^5}{90}.$$

Substituting the values of x_0, x_1, x_2, a_0, a_1, and a_2, the final form of Simpson's rule is given as:

$$\int_{x_0}^{x_2} f(x) \, dx = \frac{h}{3} \left[f(x_0) + 4f(x_1) + f(x_2) \right] - \frac{h^5}{90} \cdot f^{(4)}(\zeta).$$

Here, $h = x_1 - x_0 = x_2 - x_1$ is the step size.

2. Romberg Integration was applied to the following functions until the condition

$$|R_{n-1,n-1} - R_{n,n}| \leq 10^{-5}$$

was satisfied:

Function	$R_{n,n}$	n	**Number of Function Evaluations**
$\int_0^1 x^{1/3}\,dx$	0.74999542	12	2049
$\int_0^1 x^2e^{-x}\,dx$	0.16060280	4	9

For the same n, applying the Composite Trapezoidal Rule with 2^{n-1} subintervals produced $R_{n,1}$. The results, true values, and absolute errors are summarized below:

Integral	$R_{n,n}$	$R_{n,1}$	**True Value**	$\epsilon_{n,n}$	$\epsilon_{n,1}$	**Step Size** (h)
$\int_0^1 x^{1/3}\,dx$	0.7499954	0.7499893	0.7500000	0.0000046	0.0000107	0.0004882
$\int_0^1 x^2e^{-x}\,dx$	0.1606028	0.1610798	0.1606028	0.0000000	0.0004770	0.1250000

Theoretical analysis indicates that the Composite Trapezoidal Method (CTM) exhibits an error term proportional to $O(h^2)$, while Romberg Integration is theoretically capable of achieving an error term proportional to $O(h^{2n})$. As a result, Romberg Integration should, in principle, produce errors that are n orders of h smaller than those produced by CTM. However, this expected behavior is not evident in the results for either integral. In particular, Romberg's Method does not outperform CTM for the first integral.

This inconsistency could be attributed to the fact that the calculation of the first integral involves extremely small values, which are especially susceptible to numerical errors. Furthermore, although the first integral appears simpler, it requires three times as many iterations as the second integral to achieve the same tolerance level of 10^{-5}. This increased effort is likely a result of the numerical challenges posed by the small values involved in the computation.

3. The following results are obtained for $\int_0^1 x^{1/3}$ and $\int_0^1 x^2e^{-x}$ using Gaussian Quadrature:

f / n	2	3	4	5
$x^{1/3}$	0.759778	0.753855	0.751946	0.751132
x^2e^{-x}	0.15941	0.160595	0.160603	0.160603

where n is the no of function evaluation.

The values obtained by Romberg Integration for 2 ($R_{2,2}$) and 4 ($R_{4,4}$) intervals are tabulated below:

f / n	3	9
$x^{1/3}$	0.695800	0.742500
x^2e^{-x}	0.161080	0.160603

Gaussian quadrature clearly outperforms Romberg integration. From the tables, Gaussian quadrature gives 0.160603 with 4 function evaluations, while Romberg integration requires 9 function evaluations to achieve the same result for x^2e^{-x}. Similarly, Gaussian quadrature gives 0.753855 with 3 function evaluations, whereas Romberg integration gives only 0.742500 with 9 function evaluations for $x^{1/3}$.

The number of function evaluations increases exponentially with n (related to $R_{n,n}$) for Romberg integration, but only linearly in the case of Gaussian Quadrature. Hence, Gaussian Quadrature is clearly the better method in both cases.

Appendix

```
# Q 2
import numpy as np
# Trapezoidal integration function
def trapezoidal(f, a, b, h):
    n = int((b - a) / h)
    sum = f(a) + f(b)
    temp_a = a
    # Initial evaluations at a and b
    for k in range(1, n):
        temp_a += h
        sum += 2 * f(temp_a)

    integral_trap = sum * h / 2
    return integral_trap

# Romberg Integration function
def Romberg_Integration(f, tol, a, b):
    R = [[0.0]]  # Initialize with a single element
    n = 1  # Start with the first level

    while True:
        h = (b - a) / (2 ** (n - 1))  # Halving step size at each level
        trap_result = trapezoidal(f, a, b, h)

        if n == 1:
            R[0][0] = trap_result
        else:
            R.append([trap_result])  # Add the new trapezoidal result

            # Richardson extrapolation
            for j in range(1, n):
                R[n - 1].append((4 ** j * R[n - 1][j - 1] - R[n - 2][j - 1]) / (4 ** j -

            # Check tolerance
            if abs(R[n - 1][n - 1] - R[n - 2][n - 2]) < tol:
                break
        n += 1
    return R, n

# Define functions to integrate
```

```
def f_a(x):
    return x**(1/3)
def f_b(x):
    return x**2 * np.exp(-x)
# Parameters for integration
tolerance = 10**(-5)
a = 0
b = 1

# Calculate Romberg integration result for f_a
print("Integration for f_a(x) = x^(1/3):")
R_out_a, n_a, = Romberg_Integration(f_a, tolerance, a, b)
print("Romberg integration matrix:")
for row in R_out_a:
    print(row)
print(f"Final result (R[n-1, n-1]): {R_out_a[-1][-1]}")
print(f"Number of levels (n): {n_a}")

# Calculate Romberg integration result for f_b
print("\nIntegration for f_b(x) = x^2 * e^(-x):")
R_out_b, n_b = Romberg_Integration(f_b, tolerance, a, b)
print("Romberg integration matrix:")
for row in R_out_b:
    print(row)
print(f"Final result (R[n-1, n-1]): {R_out_b[-1][-1]}")
print(f"Number of levels (n): {n_b}")

import numpy as np

# Predefined Gauss-Legendre roots and coefficients for different values of n
gauss_legendre_data = {
    2: [
        {"root": 0.5773502692, "coefficient": 1.0000000000},
        {"root": -0.5773502692, "coefficient": 1.0000000000}
    ],
    3: [
        {"root": 0.7745966692, "coefficient": 0.5555555556},
        {"root": 0.0000000000, "coefficient": 0.8888888889},
        {"root": -0.7745966692, "coefficient": 0.5555555556}
    ],
    4: [
        {"root": 0.8611363116, "coefficient": 0.3478548451},
```

```
        {"root": 0.3399810436, "coefficient": 0.6521451549},
        {"root": -0.3399810436, "coefficient": 0.6521451549},
        {"root": -0.8611363116, "coefficient": 0.3478548451}
    ],
    5: [
        {"root": 0.9061798459, "coefficient": 0.2369268850},
        {"root": 0.5384693101, "coefficient": 0.4786286705},
        {"root": 0.0000000000, "coefficient": 0.5688888889},
        {"root": -0.5384693101, "coefficient": 0.4786286705},
        {"root": -0.9061798459, "coefficient": 0.2369268850}
    ]
}

# Define functions to integrate
def f_a(x):
    """Function f_a(x) = x^(1/3) transformed for Gauss-Legendre interval [-1, 1] to [0,
    x = (x + 1) / 2
    return x ** (1 / 3)

def f_b(x):
    """Function f_b(x) = x^2 * exp(-x) transformed for Gauss-Legendre interval [-1, 1] t
    x = (x + 1) / 2
    return x ** 2 * np.exp(-x)
# Q 3
# Gauss Quadrature method
def Gauss_Quadrature(f, n):
    """Calculates the integral of function f using Gauss-Legendre quadrature with n poin
    integral = 0
    gauss_legendre_data_n = gauss_legendre_data[n]

    # Sum the function values at each root, weighted by the corresponding coefficient
    for data in gauss_legendre_data_n:
        integral += data["coefficient"] * f(data["root"])

    # Divide by 2 to adjust for the interval [0, 1]
    return integral / 2, len(gauss_legendre_data_n)

# Array of values for n
n_values = np.array([2, 3, 4, 5])
Integral_a = np.zeros_like(n_values, dtype=float)
Integral_b = np.zeros_like(n_values, dtype=float)
evals_a = np.zeros_like(n_values, dtype=float)
```

```
evals_b = np.zeros_like(n_values, dtype=float)

# Calculate Integrals for f_a and f_b
for i, n in enumerate(n_values):
    Integral_a[i], evals_a[i] = Gauss_Quadrature(f_a, n)
    Integral_b[i], evals_b[i] = Gauss_Quadrature(f_b, n)

# Display Results
print("Integration results:")
print("f_a(x) = x^(1/3):")
for i, n in enumerate(n_values):
    print(f"n = {n}: {Integral_a[i]:.6f},  {int(evals_a[i])}")

print("\nf_b(x) = x^2 * e^(-x):")
for i, n in enumerate(n_values):
    print(f"n = {n}: {Integral_b[i]:.6f}, {int(evals_b[i])}")
```