Università della Svizzera italiana

Faculty of Informatics

Introduction to Computational Science

Assignment5

Zhixiang Dai, Daniele del Pozzo

Spring Semester, May 14, 2025

1 Newton-Cotes-Formula

We want to approximate the integral:

$$I = \int_0^4 xe^x \, dx$$

Solution

The antiderivative of the integrand is:

$$F(x) = e^x(x-1)$$

So the exact value of the integral is:

$$I = F(4) - F(0) = e^4(4-1) - e^0(0-1) = 3e^4 + 1 \approx 164.02$$

Part 1: Basic Approximation Methods

Trapezoidal Rule This method approximates the area under the curve using straight lines between the endpoints:

$$T[f] = \frac{4-0}{2}\left(f(0) + f(4)\right) = 2\left(0 + 4e^4\right) = 8e^4 \approx 436.02$$

The error is:

$$|T[f] - I| = |8e^4 - 3e^4 - 1| = |5e^4 - 1| \approx 271.99$$

Simpson's Rule This method uses a quadratic approximation with points $x = 0, 2, 4$:

$$S[f] = \frac{4-0}{6}\left(f(0) + 4f(2) + f(4)\right) = \frac{2}{3}\left(0 + 4 \cdot 2e^2 + 4e^4\right) = \frac{2}{3}(8e^2 + 4e^4) = \frac{16e^2 + 8e^4}{3}$$

$$\Rightarrow S[f] \approx 184.23$$

The error is:

$$|S[f] - I| = \left|\frac{16e^2 + 8e^4}{3} - (3e^4 + 1)\right| = \left|\frac{16e^2 - e^4 - 3}{3}\right| \approx 20.21$$

Part 2: Composite Methods (h = 1)

Composite Trapezoidal Rule Using 4 subintervals with $h = 1$:

$$T_4[f] = \frac{1}{2}\left(f(0) + 2f(1) + 2f(2) + 2f(3) + f(4)\right)$$

$$= \frac{1}{2}\left(0 + 2e^1 + 2 \cdot 2e^2 + 2 \cdot 3e^3 + 4e^4\right) = \frac{1}{2}(2e + 4e^2 + 6e^3 + 4e^4)$$

$$= e + 2e^2 + 3e^3 + 2e^4 \approx 186.17$$

The error is:

$$|T_4[f] - I| \approx 186.17 - 164.02 = 22.15$$

Composite Simpson's Rule Using 8 equally spaced points (4 subintervals + midpoints):

$$S_4[f] = \frac{1}{6}\left(f(0) + 4f(0.5) + 2f(1) + 4f(1.5) + 2f(2) + 4f(2.5) + 2f(3) + 4f(3.5) + f(4)\right)$$

This is often written as:

$$S_4[f] = \frac{1}{3}T_4[f] + \frac{2}{3}\left(f(0.5) + f(1.5) + f(2.5) + f(3.5)\right)$$

$$\Rightarrow S_4[f] \approx \frac{1}{3}(186.17) + \frac{2}{3}(e^{0.5} + 1.5e^{1.5} + 2.5e^{2.5} + 3.5e^{3.5}) \approx 164.15$$

The error is:

$$|S_4[f] - I| \approx |164.15 - 164.02| = 0.13$$

Conclusion

The composite Simpson's rule yields the smallest error (≈ 0.13), making it the most accurate among the methods considered. However, it requires nearly twice the number of function evaluations compared to the composite trapezoidal rule. The composite trapezoidal rule still offers a reasonable compromise between computational cost and accuracy, with an error about 12 times smaller than the basic trapezoidal rule. The composite Simpson's rule is approximately 155 times more accurate than the basic Simpson's rule.

2 Adaptive Quadrature

Part 1: Compute the exact value of the integral.

$$\begin{aligned}
\int_0^1 \left(1 - 4(x - 0.5)^2\right) \mathrm{d}x &= \int_0^1 \left(1 - 4\left(x^2 - x + \frac{1}{4}\right)\right) \mathrm{d}x \\
&= \int_0^1 (-4x^2 + 4x)\, \mathrm{d}x \\
&= 4\int_0^1 \left(\frac{x^2}{2} - \frac{x^3}{3}\right)' \mathrm{d}x \\
&= 4\left[\frac{x^2}{2} - \frac{x^3}{3}\right]_0^1 \\
&= 4\left(\frac{1}{2} - \frac{1}{3}\right) = \frac{2}{3}
\end{aligned}$$

Part 2: Use the algorithm for the adaptive quadrature to approximate the integral with a tolerance of $\epsilon = \frac{1}{10}$

Define $f(x) = 1 - 4(x - 0.5)^2$. We can know $f(0) = 0, f(0.5) = 1, f(1) = 1$.

$$S = \frac{0.5}{3}[f(0) + 4f(0.5) + f(1)] = \frac{2}{3},$$
$$S_L = \frac{0.25}{3}[f(0) + 4f(0.25) + f(0.5)] = \frac{1}{3},$$
$$S_R = \frac{0.25}{3}[f(0.5) + 4f(0.75) + f(1)] = \frac{1}{3},$$
$$E = \frac{|(S_L + S_R) - S|}{15} = 0 < \epsilon.$$

Error meets tolerance, so the approximation is $\frac{2}{3}$.

3 Programming Task I

Python Code

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the integrand
def f(x):
    return 1 - 4 * (x - 0.5)**2

# Trapezoidal rule implementation
def trapezoidal_rule(f, a, b, num_intervals):
    x = np.linspace(a, b, int(num_intervals))
    h = (b - a) / (len(x) - 1)
    y = f(x)
    integral = h / 2 * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
    return integral

# Integration bounds
a, b = 0, 4

# Interval counts
n_values = np.arange(3, 51)
errors = []

# Exact integral value (computed analytically)
exact_value = -160 / 3

# Compute the approximate integral and error for each n
for n in n_values:
    approx = trapezoidal_rule(f, a, b, n)
    error = abs(exact_value - approx)
    errors.append(error)

# Plot the error on a semilog-y scale
plt.figure(figsize=(8, 5))
plt.semilogy(n_values, errors, marker='o')
plt.xlabel('Number of intervals (n)')
```

```
plt.ylabel('Absolute Error')
plt.title('Error of Trapezoidal Rule')
plt.grid(True, which='both', linestyle='--')
plt.show()
```

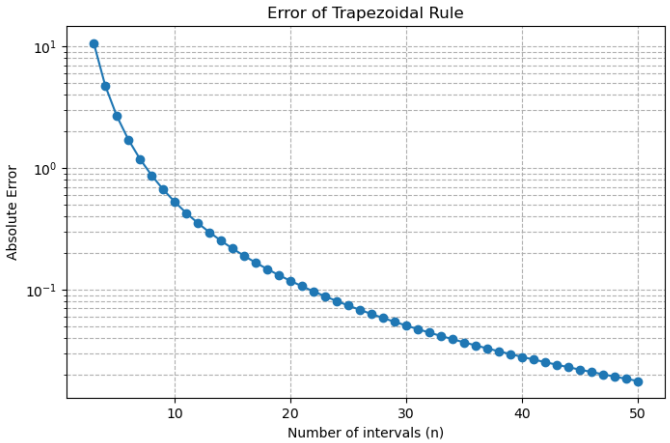

Figure 1: Error of Trapezoidal Rule

4 Programming Task II

Python Code

```
import numpy as np
import matplotlib.pyplot as plt

def midpoint(f, a, b, h):
    n = max(1, round((b - a) / h))
    h = (b - a) / n
    x_mid = a + h * (np.arange(n) + 0.5)
    return h * np.sum(f(x_mid))

def trapezoidal(f, a, b, h):
    n = max(1, round((b - a) / h))
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    fx = f(x)
    return (h / 2) * (fx[0] + 2.0 * np.sum(fx[1:-1]) + fx[-1])

def f(x):
    return np.sqrt(4 * np.sin(x)**2 + np.cos(x)**2)

a, b = 0.0, 2.0 * np.pi
exact = 9.688448220547674

i_vals = np.arange(1, 31)
h_vals = np.pi / i_vals
err_mid = []
err_trap = []

for i in i_vals:
```

```
    h = np.pi / i
    err_mid.append(abs(midpoint(f, a, b, h) - exact))
    err_trap.append(abs(trapezoidal(f, a, b, h) - exact))

err_mid = np.array(err_mid)
err_trap = np.array(err_trap)

plt.figure()
plt.semilogy(i_vals, err_mid, label='Midpoint Error')
plt.semilogy(i_vals, err_trap, label='Trapezoidal Error')
plt.xlabel('i (h = /i)')
plt.ylabel('Absolute Error (log scale)')
plt.title('Error Comparison: Midpoint vs Trapezoidal Rule')
plt.legend()
plt.grid(True)
plt.show()

```

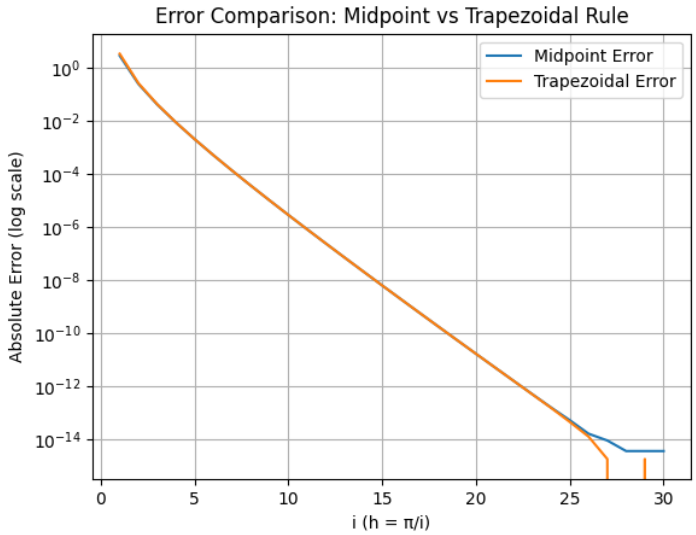

Figure 2: Error Comparison: Midpoint vs Trapezoidal Rule