Numerical Methods
Lab 7. Numerical Integration

Maryna Borovyk

November 16, 2024

1 Introduction

Numerical integration is used to approximate the value of definite integrals when analytical methods are impractical or impossible. During this lab we are evaluating the definite integral:

$$I = \int_0^1 \frac{1}{\sqrt{x^2 + 1}}\, dx \tag{1}$$

using three different numerical methods:

- **Rectangle Method:** A simple method that approximates the integral by summing the areas of rectangles under the curve.
- **Trapezoid Method:** An improvement over the rectangle method, using trapezoids to approximate the curve, providing better accuracy.
- **Simpson's Method:** A higher-order approximation that uses parabolic segments to estimate the integral, achieving even greater accuracy for smooth functions.

2 Code Implementation

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
def f(x):
    return 1 / (x**2 + 1)
```

[3]:
```
exact_integral = np.log(1 + np.sqrt(2))
```

2.1 Rectangle (Midpoint) Method

[4]:
```
def rectangle_method(f, a, b, n):
    h = (b - a) / n
    result = 0
    for i in range(n):
        result += f(a + (i + 0.5) * h)
    return result * h
```

2.2 Trapezoidal Method

[5]:
```
def trapezoidal_method(f, a, b, n):
    h = (b - a) / n
    result = 0.5 * (f(a) + f(b))
    for i in range(1, n):
        result += f(a + i * h)
    return result * h
```

2.3 Simpson's Method

[6]:
```
def simpsons_method(f, a, b, n):
    if n % 2 == 1:
        n += 1  # Simpson's rule requires an even number of intervals
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n, 2):
        result += 4 * f(a + i * h)
    for i in range(2, n-1, 2):
        result += 2 * f(a + i * h)
    return result * h / 3
```

3 Error calculation and plotting

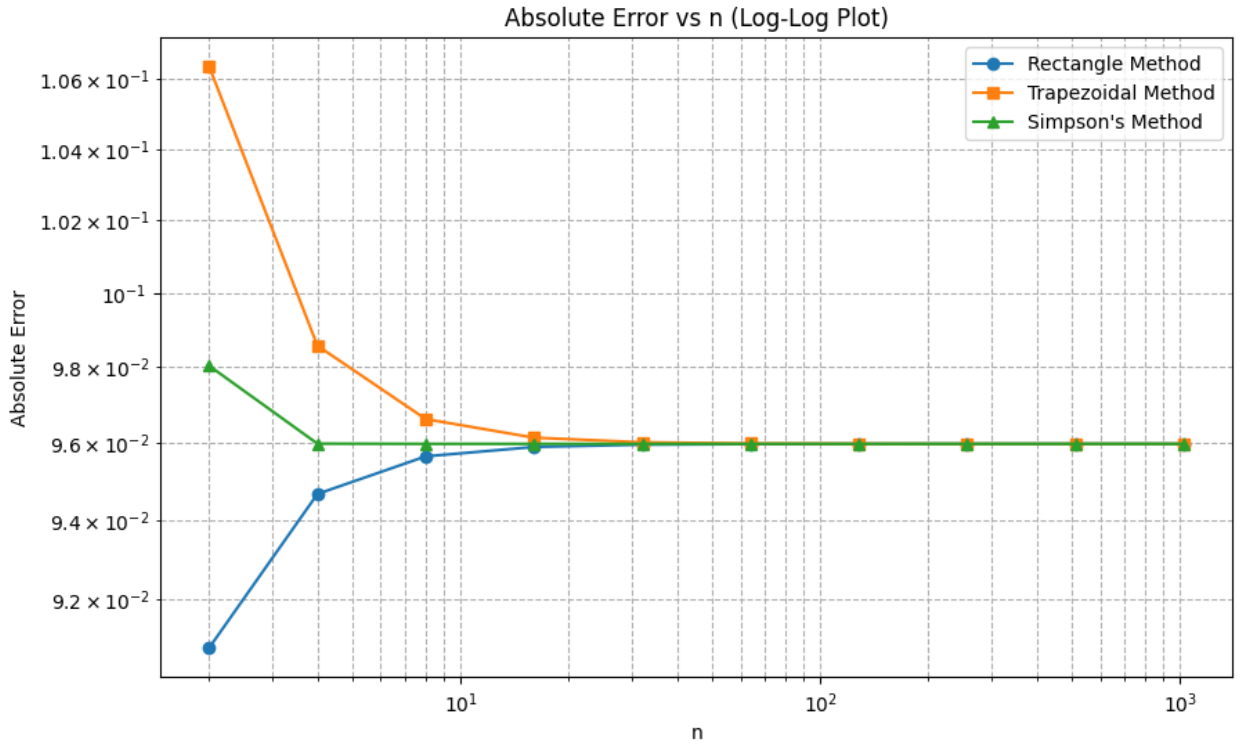
[7]:
```
a, b = 0, 1
n_values = [2**i for i in range(1, 11)]
errors_rectangle = []
errors_trapezoidal = []
errors_simpsons = []

for n in n_values:
    rect_result = rectangle_method(f, a, b, n)
    trap_result = trapezoidal_method(f, a, b, n)
    simp_result = simpsons_method(f, a, b, n)

    errors_rectangle.append(abs(rect_result - exact_integral))
    errors_trapezoidal.append(abs(trap_result - exact_integral))
    errors_simpsons.append(abs(simp_result - exact_integral))

plt.figure(figsize=(10, 6))
plt.loglog(n_values, errors_rectangle, marker='o', label='Rectangle Method')
plt.loglog(n_values, errors_trapezoidal, marker='s', label='Trapezoidal Method')
plt.loglog(n_values, errors_simpsons, marker='^', label='Simpson\'s Method')
plt.xlabel('n')
plt.ylabel('Absolute Error')
plt.title('Absolute Error vs n (Log-Log Plot)')
plt.legend()
```

```
plt.grid(True, which='both', ls='--')
plt.show()
```



4 Conclusion

From the graph, we can draw the following conclusions:

- The **Rectangle Method** exhibits relatively slow convergence compared to the other two methods. As n increases, the error initially decreases but remains consistently higher than that of the Trapezoid and Simpson's Methods.
- The **Trapezoid Method** converges more rapidly than the Rectangle Method. The error decreases significantly as n increases, indicating its improved accuracy over the Rectangle Method due to better approximation of the curve.
- The **Simpson's Method** demonstrates the highest accuracy, with the lowest absolute error across all values of n. The convergence is evident even for smaller values of n, showcasing the method's efficiency and superior approximation capability for smooth functions.

Overall, the lab highlights the trade-off between computational cost and accuracy for numerical integration methods. Simpson's Method stands out as the most accurate, making it the preferred choice for applications requiring high precision. However, for problems where computational resources are limited, the Trapezoid Method may serve as a reasonable compromise. The Rectangle Method, while computationally simple, is the least accurate and may not be suitable for scenarios requiring high precision.