Adaptive Numerical Integration

Numerical Methods in Calculus and Algebra

Tamir Mashbat
CEU Vienna

December 7, 2025

GitHub Repository:
`https://github.com/tamirmt/AdaptiveNumericalIntegration.git`

Abstract

This report presents the theoretical foundations and practical implementation of adaptive numerical integration. We review classical quadrature rules, explain why adaptive refinement improves computational efficiency, and demonstrate numerical experiments implemented in a Jupyter notebook. The experiments show how adaptive methods achieve high accuracy while reducing function evaluations compared to fixed-step quadrature.

1 Introduction

Numerical integration aims to approximate definite integrals of functions for which a closed-form solution is unavailable or impractical. Classical quadrature rules such as the trapezoidal and Simpson methods use uniform partitions of the interval of integration. While these methods perform well for smooth functions, their efficiency deteriorates when the integrand exhibits rapid variation, sharp peaks, or local irregularities.

Adaptive numerical integration addresses these limitations by refining the integration interval only where needed. The algorithm automatically chooses smaller steps in regions where the integrand changes quickly and larger steps where it behaves smoothly. This results in higher accuracy at lower computational cost.

2 Theoretical Background

2.1 Classical Quadrature Rules

The trapezoidal rule approximates an integral by

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}(f(a)+f(b)),$$

with global error of order $O(h^2)$ for step size h.

Simpson's rule uses a quadratic interpolant:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right),$$

with global error of order $O(h^4)$.

Both rules suffer when f varies sharply in small subregions. A fixed step size forces excessive evaluations even in smooth regions, wasting computational effort.

2.2 Principle of Adaptive Methods

The key idea is to compare a coarse approximation with a finer one. For Simpson's method, if

$$S(a,b) \text{ is Simpson on } [a,b],$$

and

$$S(a,m) + S(m,b) \text{ is Simpson on } [a,m] \cup [m,b], \quad m = \frac{a+b}{2},$$

then the estimated error satisfies

$$|S(a,b) - (S(a,m) + S(m,b))| \approx \frac{1}{15}E,$$

where E is the true error term.

If the error is below the tolerance ε, the approximation is accepted. Otherwise, the interval is recursively subdivided. This strategy forms the basis of the widely used adaptive Simpson algorithm.

2.3 Adaptive Simpson Algorithm

Given tolerance ε, define:

1. Compute $S(a,b)$.
2. Compute $S(a,m)$ and $S(m,b)$.
3. If
   $$|S(a,b) - (S(a,m) + S(m,b))| < 15\varepsilon,$$
   accept $S(a,m) + S(m,b)$ as the integral estimate.
4. Otherwise, recursively apply the same procedure on both subintervals.

The method concentrates evaluation effort in regions where f is difficult, improving accuracy without increasing global computational cost.

3 Implementation

All experiments were implemented in Python using a Jupyter notebook. The repository linked above contains:

- a fully commented notebook,
- reusable Python modules implementing adaptive Simpson and classical quadrature,
- environment files to reproduce the experiments.

3.1 Functions Used in Experiments

We test the method on functions exhibiting different behaviors:

$$\begin{aligned} f_1(x) &= \sin(x^2), \\ f_2(x) &= \frac{1}{1+25x^2}, \\ f_3(x) &= \sqrt{x}, \\ f_4(x) &= e^{-100(x-0.5)^2}. \end{aligned}$$

These functions include oscillatory behavior, near-singularity behavior, and sharp localized peaks.

3.2 Evaluation Metrics

For each integrand, we compare:

1. the number of function evaluations,
2. the absolute error relative to a high-precision reference value,
3. the computational time (Python wall-clock time).

4 Results

Across all test functions, adaptive Simpson consistently required fewer function evaluations than fixed-step Simpson while achieving equal or better accuracy. For example, in the case of $f_1(x) = \sin(x^2)$, the adaptive method focused its refinements in regions where the integrand oscillates rapidly.

For the sharply peaked function $f_4(x)$, fixed-step Simpson required more than an order of magnitude more evaluations to achieve comparable accuracy. Adaptive Simpson refined the interval near $x = 0.5$ while leaving other parts coarse, demonstrating its efficiency.

Tables and plots illustrating these results are included in the accompanying notebook.

5 Discussion

The experiments confirm several theoretical insights:

- Adaptive refinement is highly effective for integrands with nonuniform complexity.
- Fixed-step Simpson wastes resources in smooth regions.
- Recursive subdivision naturally balances accuracy and computational effort.
- The choice of tolerance directly influences both accuracy and runtime.

In practice, adaptive methods form the basis of many professional integration tools (such as MATLAB's `quad` and SciPy's `quad`), demonstrating their robustness and general applicability.

6 Conclusion

Adaptive numerical integration provides a powerful improvement over classical fixed-grid quadrature. By concentrating evaluations where they are needed most, the method achieves high precision with lower computational cost. The numerical experiments implemented in the notebook illustrate these advantages clearly and highlight the relevance of adaptive techniques in scientific computing.

References

1. R. Burden and J. Faires, *Numerical Analysis*, 10th Edition.
2. A. Quarteroni, R. Sacco, F. Saleri, *Numerical Mathematics*, Springer.
3. SciPy documentation: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html`