Introduction to Programming - Rectangular and Trapezoidal Rule

Demir Kucukdemiral 2883935K

January 3, 2025

Abstract

This report compares the performance of trapezoidal and rectangular numerical integration methods on three different continuous-time functions and also provides a solution to mild singularity integrals. The code finds a way to avoid these singularities in both methods, computes the integrals successfully, and plots the results of integrals against the number of iterations, enabling the user to observe the convergence performances of the considered methods. The results of these experiments show that the trapezoidal method is significantly more computationally efficient than the rectangular rule.

1 Introduction

1.1 Integration

The attached Python program uses two different methods, rectangular and trapezoidal, to find the result of the integrals each using a different theory.

The Rectangular Rule divides the function into vertically placed rectangles of different heights but of the same width ϵ, and sums up the areas of the rectangles to approximate the area under the graph. For a function $f(x)$ the integral can be approximated using the rectangular rule as

$$I = \sum_{i=0}^{n} \epsilon f(a + i\epsilon) \approx \int_a^b f(x)\mathrm{d}x, \tag{1}$$

where, $\epsilon = \frac{b-a}{n}$, is the constant width and n is the number of iterations which designates the precision.

On the other hand, the trapezoidal rule improves the precision of the rectangular rule by adding triangles to connect the rectangles. This creates a much smoother curve and allows us to compute accurate integrals with fewer iterations. In this case, the numerical integration can be expressed as:

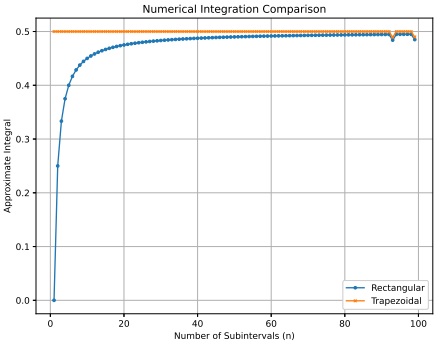
$$I = \frac{\epsilon}{2} f(a) + \sum_{i=1}^{n-1} \epsilon(f(a + i\epsilon)) + \frac{\epsilon}{2} f(b). \tag{2}$$

1.2 Singularity Avoidance

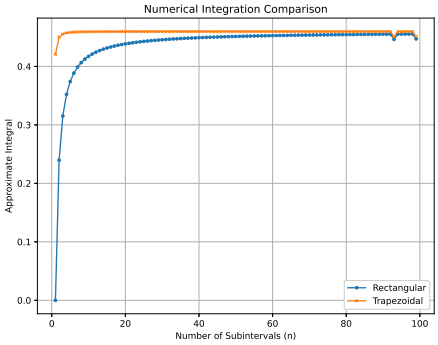
While basic numerical integration techniques apply to proper integrals, improper integrals may be required to approximate functions that are undefined or diverge to infinity at specific points (singularities). This can be achieved by excluding the point where the function is not properly defined and integrating up to just before and just after that point. For instance, consider a function $f(x)$ with a singularity at $x = \alpha$. Then the improper integral can be expressed as

$$I = \int_a^b f(x)\mathrm{d}x = \int_a^{\alpha-\epsilon} f(x)\mathrm{d}x + \int_{\alpha+\epsilon}^b f(x)\mathrm{d}x \approx \sum_{i=a}^{\alpha-\epsilon} \epsilon f(i) + \sum_{i=\alpha+\epsilon}^{b} \epsilon f(i) \tag{3}$$

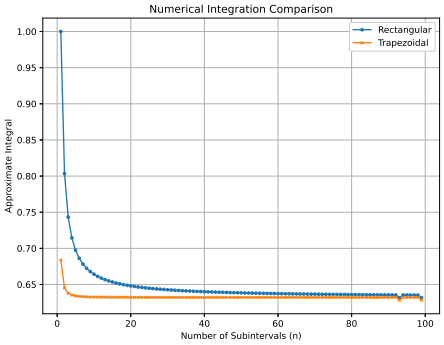
using the rectangular rule. This methodology can also be extended to the trapezoidal method as well. It is worth noting that while this method can handle both proper and improper integrals, it cannot address the numerical integration of non-integrable functions.



(a) Integral of x against n.



(b) Integral of $\sin(x)$ against n.



(c) Integral of e^{-x} against n.

Figure 1: Comparison of numerical integration methods, trapezoidal (orange) and rectangular (blue) for functions (a) $f(x) = x$, (b) $f(x) = \sin(x)$ and (c) $f(x) = \exp(-x)$ between $[0, 1]$.

To be able to do this, the code must be able to identify the points where the domain of the function is not properly defined under the set of real numbers. This identification is performed by using the *try-except* method in Python, where the code attempts to compute the function at a certain point, α, and if an error occurs, it sets the point, $\alpha - \epsilon$, and $\alpha + \epsilon$, as part of boundaries of the integral. These points, along with a and b, are then stored in a 'points' array such as:

$$points = \{a, \alpha_1 - \epsilon, \alpha_1 + \epsilon, \ldots, \alpha_m - \epsilon, \alpha_m + \epsilon, b\} \tag{4}$$

Then, these *points* become the limits in the sums given in equation (3).

2 Results and Conclusion

In this section, the results of the numerical integration of the functions $f(x) = x$, $f(x) = \sin(x)$, and $f(x) = e^{-x}$ are presented and plotted against the number of iterations n (as $\epsilon = \frac{b-a}{n}$) where $n \in \{2, \ldots, 100\}$. The integrations are performed using two methods: the rectangular rule and the trapezoidal rule. These graphs in Figure 1 show the precision of the two methods for a given value of n highlighting which method is more computationally efficient to use for larger programs.

Figures 1a, 1b, and Figure 1c plot the values of numerical integration against the iteration number n. It is apparent from these graphs that the convergence with the trapezoidal method ($n \approx 5$) occurs with fewer iterations than with the rectangular rule ($n \approx 50$). This could mean that the trapezoidal rule would be much more beneficial in continuous-time integration, where integration has to be conducted after each time constant.

The trapezoidal (orange line) approach usually gives better approximations for all three functions than the rectangular approach (blue line). This shows the benefit of trapezoidal integration, which ensures smoother convergence and accuracy.

The pseudo-code provided in the Appendix A shows the integration algorithm used in Python coding. Please note that the complete Python code is uploaded to Moodle as a separate file. Alternatively, you can access the code on GitHub through my page using this link.

A Pseudo code

Algorithm 1 Integration of functions with mild singularities (Rect/Trap Methods)

Inputs:
  `f`: integrand (possibly singular),
  `a,b`: interval limits,
  `eps`: step size.

Step 1: Singularity Finder (basic idea)
`points` $\leftarrow$ empty list
If $f(a)$ fails **then**
  append $(a + \texttt{eps})$ to `points`
Else
  append a to `points`
EndIf
for $i \leftarrow 1$ to $\lfloor (b-a)/\texttt{eps} \rfloor$ **do**
  $x \leftarrow a + i \times \texttt{eps}$
  If $f(x)$ fails **then**
    append $(x - \texttt{eps})$, $(x + \texttt{eps})$ to `points`
  EndIf
end for
append b to `points`

Step 2: Rectangular Rule on [p, q]
$n \leftarrow \lfloor (q-p)/\texttt{eps} \rfloor$
$S \leftarrow 0$
for $j \leftarrow 0$ to $n-1$ **do**
  $x_j \leftarrow p + j \times \texttt{eps}$
  $S \leftarrow S + f(x_j) \times \texttt{eps}$
end for
Return S

Step 3: Trapezoidal Rule on [p, q]
$n \leftarrow \lfloor (q-p)/\texttt{eps} \rfloor$
$T \leftarrow 0.5 \times \texttt{eps} \times (f(p) + f(q))$
for $j \leftarrow 1$ to $n-1$ **do**
  $x_j \leftarrow p + j \times \texttt{eps}$
  $T \leftarrow T + f(x_j) \times \texttt{eps}$
end for
Return T

Step 4: Piecewise Integration (skipping singularities)
Rect $\leftarrow 0$, **Trap** $\leftarrow 0$
for $k \leftarrow 0$ to len(`points`) $- 2$ step 2 **do**
  $p \leftarrow \texttt{points}[k]$
  $q \leftarrow \texttt{points}[k+1]$
  Rect $\leftarrow$ **Rect** + RectangularRule(p, q, eps)
  Trap $\leftarrow$ **Trap** + TrapezoidalRule(p, q, eps)
end for

Output: Rect and Trap