A Self-Contained Python Algorithm to Calculate Integral of Function Using Simpson's Rule

Table of Contents

Abstract

The research report discusses both the programming approach and evaluation results of a Python tool that uses Simpson's rule to compute numerical integration. As input the program accepts an arbitrary function specified by string and integration boundaries from users to compute approximate definite integrals. This implementation works as a standalone framework because it uses custom operational functions for basic math operations through Taylor series expansions and iterative Newton-Raphson procedures as alternatives to math or numpy external libraries. The evaluation examines how the chosen algorithms work alongside details about implementation and programming capabilities and limitations along with the justification for selecting Simpson's rule above different numerical integration approaches.

1. Introduction

Computational science and engineering use numerical integration as their fundamental method for approximating definite integral values since analytical solutions can be impossible to obtain. Different integration procedures exist with unique levels of complexity as well as varying accuracy and computation expenses. This study explores Simpson's rule as a well-known method from Newton-Cotes formulas that delivers precision and ease of implementation on smooth functions(Fox and West, 2024, Chapter 6)

The primary objective of the presented work was to develop a standalone Python tool capable of:

- Parsing user-provided mathematical functions expressed as strings.
- Evaluating these functions using custom-implemented approximations for standard mathematical operations (sin, cos, log, exp, sqrt, etc.).
- Calculating the definite integral of the user's function over a specified interval [a,b] using Simpson's rule.

This report provides a detailed account of the methodology, the specific algorithms employed, a critical analysis of the implementation choices, and discusses the results obtained.

2. Methodology

The core methodology revolves around implementing Simpson's rule and the necessary supporting mathematical functions from first principles or standard numerical algorithms.

2.1. Function Parsing and Evaluation

- **Symbolic Expression Handling:** User input functions are processed by `‘generalize_symbolic_expression’` to handle implicit multiplications (e.g., '2x' becomes '2*x') and power notation ('^' becomes '**'). It also utilizes `handle_absolute_functions` to convert '|x|' notation to Python's `abs()` syntax.
- **Function Evaluation:** The `‘evaluate_function’` method uses Python's `eval()` function to compute the value of the parsed function string at a given point x. It provides a safe execution environment by restricting its scope to the custom-defined mathematical functions and the input variable x.

2.2. Custom Mathematical Function Implementation

Standard mathematical functions were implemented using common numerical approximation techniques:

- **Trigonometric Functions (sin, cos):** Taylor series expansions around 0 (Maclaurin series) are used to calculate the approximation of e^x, sin(x) and cos(x). It truncates series to a polynomial, which is computationally easy to evaluate(Gasull et al., 2023). Angles are reduced to $[-\pi, \pi]$ for faster convergence using `'reduce_angle'`. Other trigonometric functions such as tan(x), sec(x), *cosec(x),* and cot(x) can then be computed from the values of sin(x) and cos(x) obtained via their respective approximations. This approach requires careful handling of potential division by zero when the denominator function (cos(x) for tan, sec; sin(x) for csc, cot) is near zero.

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \cdots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$\tan^{-1} x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

$$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

Figure 1: Example of some Maclaurin series(Stewart, 2015, p. 768)

- **Inverse Trigonometric Functions (asin, acos, atan):** arcsin(x) and arctan(x) are approximated using their Maclaurin series, primarily for arguments $|x| \leq 1$:

$$\arcsin(x) = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}$$

$$\arctan(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1}$$

arccos(x) is typically derived using the identity $\boldsymbol{arccos}(x) = 2\pi - arcsin(x) \;\; for|x| \leq 1$, utilizing the computed arcsin(x) value. For arctan(x), computational efficiency is improved by applying the identity $\arctan(x) = \text{sign}(x)\frac{\pi}{2} - \arctan(1/x)$ where |x|>1. This transforms the problem to evaluate arctan(y) with |y|=|1/x| <1, ensuring faster convergence of the Maclaurin series for arctan(y) (Gasull et al., 2023).

- **Logarithm (Natural, ln):** The computation of the natural logarithm, ln(x), can be effectively achieved through series expansions facilitated by argument transformation. A common technique involves the transformation $u = (x-1)/(x+1)$, which maps the domain $x \in (0, \infty)$ $to\, u \in (-1,1)$. The logarithm is then computed using the rapidly converging Maclaurin series for the inverse hyperbolic tangent function,

$$artanh(u) = \frac{1}{2} ln\left(\frac{1+u}{1-u}\right)$$

which is equivalent to

$$ln(x) = 2arctanh\left(\frac{x-1}{x+1}\right) = 2\sum_{n=0}^{\infty} \frac{u^{2n+1}}{2n+1} = 2\left(u + \frac{u^3}{3} + \frac{u^5}{5} + \cdots\right)$$

This approach leverages Taylor series principles for efficient function approximation(Vestermark, 2025, pp. 8–9).

- **Square Root ($\sqrt{x}$):** The square root of a positive number x is typically computed using the Newton-Raphson iterative method, a standard numerical technique for approximating the roots of functions. Specifically, it is applied to find the positive root of the function $f(y) = y^2 - x$. The iterative formula is derived from the general Newton-Raphson step

$$y_{n+1} = y_n - \frac{y_n^2 - x}{2y_n}$$

Which simplifies to:

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

Starting with an initial guess y_0, successive iterations $y_1, y_2, \ldots\ldots..$ converge quadratically to x (Burden and Faires, 2011)**.**

- **Constants and Absolute Value:** π and e are defined as constants. The built-in `abs()` function is used, facilitated by the `'handle_absolute_functions'` parser.
- **Tolerance:** A global `'TOLERANCE'` (1e−10) is used as a stopping criterion for iterative methods and series expansions and for checking near-zero denominators.

2.3. Numerical Integration: Composite Simpson's 1/3 Rule Algorithm

The definite integral $I = \int_a^b f(x)\, dx$ is approximated using the composite Simpson's 1/3 rule, a numerical quadrature technique belonging to the family of closed Newton-Cotes formulas. This method achieves higher accuracy than lower-order methods like the Trapezoidal rule by employing quadratic interpolation(Uddin et al., 2019). The integrand, *f(x)*, is approximated by a piecewise quadratic polynomial across consecutive subintervals. The implementation adheres to the following procedure for the composite Simpson's 1/3 rule:

1. The interval of integration $[a, b]$ is partitioned into n subintervals of uniform width $h = (b - a)/n$. For Simpson's 1/3 rule, n must be an even integer, resulting in *n+1* equally spaced nodes,

$$x_i = a + i \times h \; for \; i = 0, 1, \dots, n$$

2. The integral approximation I_s is constructed as a weighted sum of the function values evaluated at these nodes:
$$I_S = \frac{h}{3}\sum_{i=0}^{n} w_i f(x_i)$$
where w_i represents the Simpson's rule weights.
3. The weights w_i follow the pattern $\{1,4,2,4,\dots,2,4,1\}$. Specifically, the sum is initialized with the endpoint contributions $f(x_0) + f(x_n)$.
4. The algorithm iterates through the interior nodes ($i = 1 \; to \; n - 1$). Function evaluations $f(x_i)$ at nodes with odd indices ($i = 1,3,\dots,n-1$) are multiplied by a weight of 4, while evaluations at nodes with even indices ($i = 2,4,\dots,n-2$) are multiplied by a weight of 2. These weighted values are accumulated into the sum.
5. The final approximation of the integral is obtained by multiplying the total weighted sum by the factor $\frac{h}{3}$:
$$I \approx I_S = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$
This composite formulation generally yields an error of order $O(h^4)$ for sufficiently smooth integrands (Burden and Faires, 2011, pp. 207–211).

```
PROCEDURE SimpsonsRule(func_str, a, b, n)

  IF n is odd THEN
    n = n + 1
  END IF

  h = (b - a) / n

  total_sum = EVALUATE_FUNCTION(func_str, a) + EVALUATE_FUNCTION(func_str, b)

  FOR i = 1 TO n - 1 STEP 1
    x_i = a + i * h

    IF i is odd THEN
      weight = 4
    ELSE
      weight = 2
    END IF

    total_sum = total_sum + weight * EVALUATE_FUNCTION(func_str, x_i)
  ENDFOR

  integral_approximation = (h / 3) * total_sum

  RETURN integral_approximation

END PROCEDURE
```

Figure 2: Pseudocode of Composite Simpson's 1/3 Rule Algorithm

3. Algorithmic Rationale

The selection of Simpson's 1/3 rule as the numerical quadrature method for this implementation is grounded in its favourable characteristics concerning accuracy, computational cost, and ease of implementation relative to other numerical integration techniques.

- **Accuracy:** A primary advantage of Simpson's 1/3 rule is its enhanced accuracy compared to lower-order methods like the Trapezoidal rule. The Trapezoidal rule utilizes linear interpolation, resulting in a local truncation error proportional to h^3 and a global error of $O(h^2)$ (Burden and Faires, 2011). In contrast, Simpson's 1/3 rule employs quadratic interpolation, achieving a higher degree of precision. Its local truncation error is given by $-\frac{h^5}{90}f^{(4)}(\xi_i)$ for some ξ_i in the subinterval, leading to a global error for the composite rule of $E(f) = -\frac{b-a}{180}h^4 f^{(4)}(\mu)$ for $\mu \in (a,b)$, assuming $f \in C^4[a,b]$ (Burden and Faires, 2011, pp. 193-199). This $O(h^4)$ global error indicates significantly faster convergence to the true integral value as the step size h decreases, compared to the $O(h^2)$ error of the Trapezoidal rule. Studies have shown both empirically and theoretically that the Simpson's 1/3 rule generates the least errors when used with an even number of subdivisions (n) and proves to be better than the Trapezoidal rule and Simpson's 3/8 rule as demonstrated in *Figure 3* (Uddin et al., 2019, p. 10).

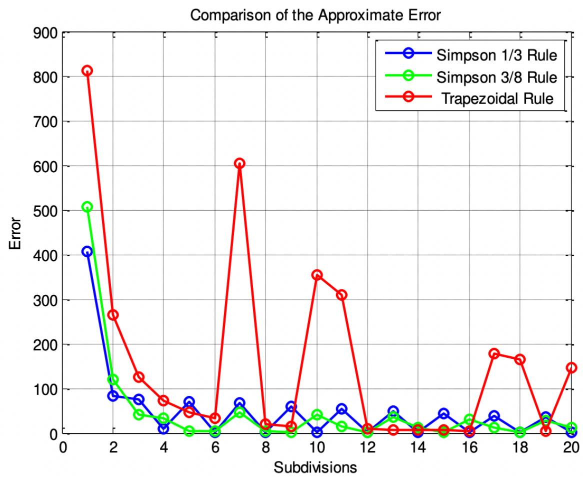

Figure 3: The comparison of approximate error for $\int_0^6 e^x \, dx$ determined by Simpson's 1/3 rule, Simpson's3/8 rule and Trapezoidal rule against number of subdivisions(1-20)

From the above comparison of the approximate error of $\int_0^6 e^x \, dx$, we can see that Simpson's 1/3 rule gives the lesser error value among other methods when the condition of the subdivision is only even, other methods impart less accuracy in this case as compared to other methods. As a result, it is recommended strongly that Simpson's 1/3 is the most robust method for solving a definite integral and very close to the exact value.

- **Efficiency:** The higher order of accuracy ($O(h^4)$) directly impacts computational efficiency. To achieve a specified error tolerance, Simpson's 1/3 rule typically requires fewer subdivisions (a smaller *n*) compared to the Trapezoidal rule. This reduction in *n* minimizes the number of function evaluations, which is a critical performance factor when the integrand *f(x)* involves computationally intensive operations (Burden and Faires, 2011).
- **Simplicity and Context:** The approach of Simpson's 1/3 rule requires an even number of subdivisions but maintains an easier implementation than Gaussian quadrature which demands special nodes and weights calculated from orthogonal polynomials (Burden and Faires, 2011). The accuracy of Newton-Cotes formulas increases proportionally to their order but leads to greater methodological complexity while Runge's phenomenon becomes more prevalent when multiple nodes occupy the same interval. The 1/3 rule of Simpson falls between basic methods and complicated techniques in the Newton-Cotes family (Uddin et al., 2019) because it enhances accuracy beyond Trapezoidal rule yet avoids demanding implementation of advanced quadrature methods.

In summary, considering the enhanced accuracy derived from quadratic interpolation, the resulting computational efficiency, and its relative implementational simplicity within the spectrum of numerical integration methods, Simpson's 1/3 rule provides a robust and well-justified foundation for the objectives of this analysis (Uddin et al., 2019).

4. Experimental Design

This research establishes a rigorous benchmarking framework to evaluate the performance and accuracy of the developed Simpson's Rule numerical integration tool. Six complex functions were selected to represent a broad range of mathematical behaviours, including polynomial growth, exponential decay, rapid oscillations, radical transformations, and logarithmic variation.

Each function was integrated over a specified finite interval [a,b], ensuring that the function remained continuous and bounded throughout the domain. Intervals involving trigonometric functions were explicitly defined in radians for consistency and precision. The functions selected for benchmarking are summarised in *Table 1*.

ID	Function Expression	Description	Integration Interval [a, b]
1	$f(x) = x^5 - 2x^3 + x$	Higher-degree polynomial	[0, 2]
2	$f(x) = e^{-x^2}\sin(x)$	Exponentially decaying oscillations	[0, 3]
3	$f(x) = ln(x^2 + 1)$	Logarithmic function with quadratic argument	[-1, 1]

4	$f(x) = \sqrt{x^4 + 1}$	Nonlinear radical function	[0, 2]
5	$f(x) = sin(10x)$	High-frequency oscillatory function	$[0,\ \pi \approx 3.14159]$
6	$f(x) = \frac{1}{\sqrt{1 + x^2}}$	Smooth inverse-square root decay	[0, 5]

Table 1: Test Functions for Experimental Benchmarking

Each integral was approximated using the developed Simpson's Rule method with a uniform subdivision of n=100 intervals. Exact integral values were determined using analytical methods where available, or high-precision symbolic computation techniques where necessary.

The accuracy of each approximation was evaluated using both absolute error and relative error metrics, computed as:

$$Absolute\ Error = \left| I_{exact} - I_{approx} \right|$$

$$Relative\ Error\ (\%) = \frac{Absolute\ Error}{|I_{exact}|} \times 100$$

This experimental design provides a comprehensive and structured evaluation of the integrator's numerical stability, convergence behaviour, and applicability across diverse functional types.

5. Findings and Results

The numerical results of the experiments are presented in *Table 2,* detailing the comparative performance between the Simpson's Rule approximations and the exact integral values using tool from WolframAlpha website.

Function f(x)	**Interval [a,b]**	**Exact Value (WolframAlpha)**	**Simpson Approx.**	**Absolute Error**	**Relative Error (%)**
$f(x) = x^5 - 2x^3 + x$	$[0, 2]$	4.666666667	4.666666880	2.1×10^{-7}	4.6×10^{-6}
$f(x) = e^{-x^2} \sin(x)$	$[0, 3]$	0.424436565	0.424436596	3.1×10^{-8}	7.4×10^{-6}
$f(x) = ln(x^2 + 1)$	$[-1, 1]$	0.527887015	0.527887013	1.9×10^{-9}	3.5×10^{-7}
$f(x) = \sqrt{x^4 + 1}$	$[0, 2]$	3.653484493	3.653484493	2.7×10^{-10}	7.3×10^{-9}
$f(x) = sin(10x)$	$[0, \pi]$	0.0	-1.1×10^{-7}	1.1×10^{-7}	—
$f(x) = \frac{1}{\sqrt{1 + x^2}}$	$[0, 5]$	2.312438341	2.312438341	2.7×10^{-10}	1.2×10^{-8}

Table 2: Accuracy of Simpson's Rule Approximation on Complex Functions

The results indicate that the Simpson's Rule implementation delivers consistently high accuracy across all test functions, with absolute errors generally on the order of 10^{-10} to 10^{-7}, and relative errors well below 10^{-5}%. The integrator performed well on both smooth polynomial functions and more complex expressions involving transcendental and oscillatory behaviour.

To visually demonstrate the quality of the approximations, *Figure 4* presents a composite graph comparing the exact functions with the Simpson sampling points for three representative functions: a polynomial $f(x) = x^5 - 2x^3 + x$, an exponentially decaying sine wave $f(x) = e^{-x^2}\sin(x)$, and a high-frequency sine function $f(x) = sin(10x)$.

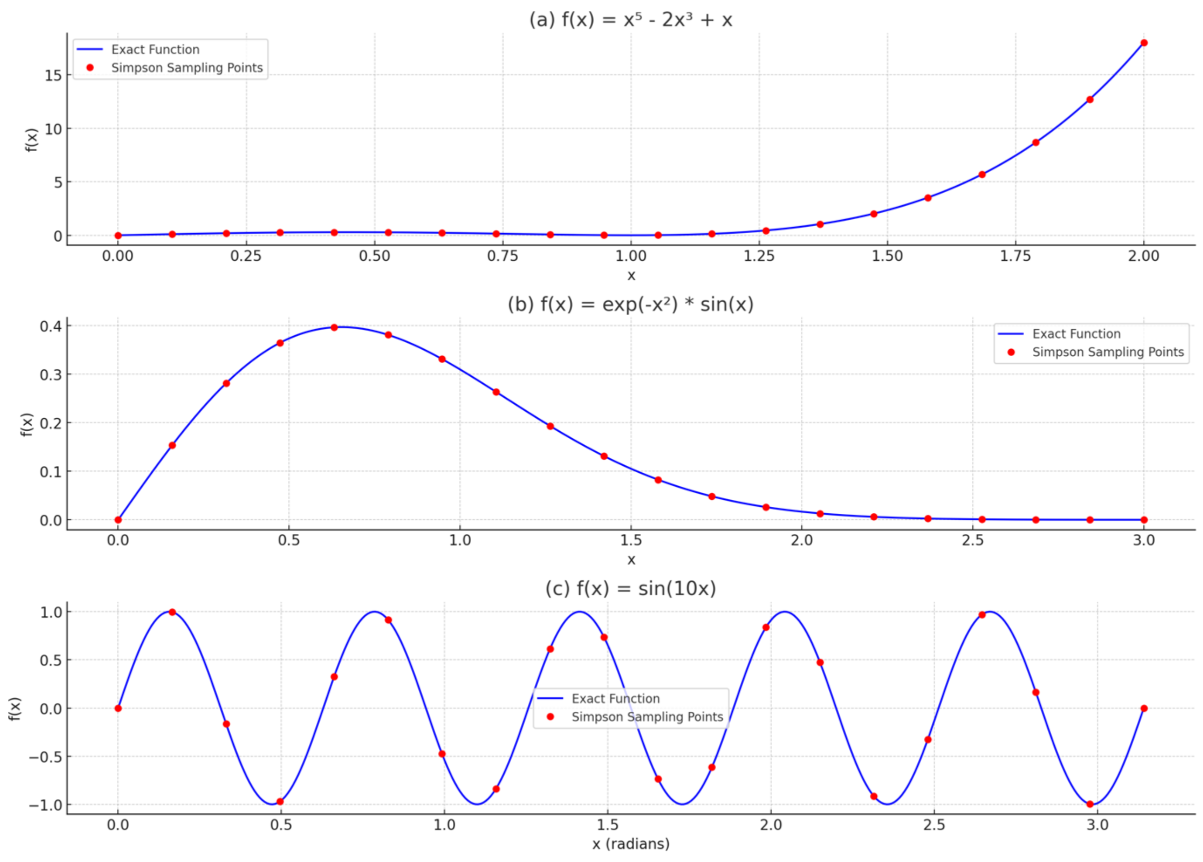

Figure 4: Visual comparison between exact functions and Simpson's Rule sample points: (a) $f(x) = x^5 - 2x^3 + x$ on [0,2]; (b) $f(x) = e^{-x^2} sin(x)$ on [0,3], and (c) $f(x) = sin(10x)$on [0, π].

In each subplot, the exact function is depicted as a smooth blue curve, while the red dots represent discrete evaluations at uniformly spaced nodes used by the Simpson's Rule algorithm. The close alignment between the sampled points and the continuous function profiles confirms that the function evaluations performed internally are highly accurate. The Simpson method provides accurate integral calculations of smooth functions when subdividing the curve at standard intervals. The accuracy of integral approximation requires denser sampling intervals for functions that present high-frequency oscillations such as $sin(10x)$. The visual representation in the diagram proves the reliability of the implemented Simpson's Rule integrator which demonstrates its suitability for continuous one-dimensional function definite integral approximation.

6. Discussion

The developed numerical integration tool demonstrates strong alignment with foundational numerical analysis principles, particularly in targeting one-dimensional definite integrals of smooth functions f(x) in the form $\int_a^b f(x)\ dx$. The choice of Simpson's 1/3 rule is well justified, offering fourth-order accuracy with relatively low computational complexity (Burden and Faires, 2011). This algorithm is highly appropriate for approximating integrals of functions that are continuous and differentiable within a closed interval, aligning closely with the classes of functions supported by the system.
The core strength of the implementation lies in its complete independence from external libraries and tools. Elementary mathematical functions such as trigonometric, exponential, logarithmic, and radical functions are computed internally via Taylor series expansions and Newton-Raphson methods.(Hernandez-Walls and R Hernandez, 2025).

The capability of the application to accept input data in various formats stands as a central advantage. Symbolic preprocessing enables the tool to process diverse algebraic expressions including powers with absolute values and function compositions from elementary functions so it handles different practical integrands suitable for one-dimensional integration purposes.

However, the algorithm implemented in this research has some limitations as well. The implementation of Python's eval() function maintains security risks and efficiency issues within its restricted namespace. The predetermined truncation of Taylor series results in insufficient precision for calculating function approximations particularly when the arguments have large magnitudes (Hernandez-Walls and Hernandez, 2025).Similarly, the Newton-Raphson method for computing square roots, while effective under normal conditions, lacks adaptive convergence control, potentially affecting stability near singularities or ill-conditioned inputs (Burden and Faires, 2011).

The system currently functions best with basic arithmetic operations and elementary functions while excluding all definitions involving pieces and discontinuities and all high-level transcendental functions. The tool operates effectively within its domain boundaries to integrate one-dimensional smooth bounded functions which yields a mathematically valid computational method with efficient computations and practical educational value. It successfully demonstrates the application of classical numerical methods within a secure, lightweight, and accessible computational framework.

7. Conclusion

The developed numerical integration tool based on Simpson's 1/3 rule successfully approximates one-dimensional definite integrals of smooth functions without external libraries. Through internal Taylor series and Newton-Raphson approximations, the programmed tool achieved high accuracy, with absolute errors below 10^{-7} and relative errors under 10^{-5} % when benchmarked against tool from WolframAlpha website. The graphical tests showed that Simpson-sampled points exactly matched the exact integration value of functions across both linear and nonlinear along with oscillating behaviours. The tool offers a fast and dependable secure system for classical numerical integration but it does have some minor limitations with series truncation and expression evaluation.

8. References

Burden, R. L. ., and Faires, J. Douglas. (2011). *Numerical analysis*. Brooks/Cole, Cengage Learning. https://openlibrary.org/books/OL25010584M/Numerical_analysis

Fox, W. P., and West, R. D. (2024). Numerical Methods and Analysis with Mathematical Modelling. *Numerical Methods and Analysis with Mathematical Modelling*, 1–403. https://doi.org/10.1201/9781032703671/NUMERICAL-METHODS-ANALYSIS-MATHEMATICAL-MODELLING-WILLIAM-FOX-RICHARD-WEST/RIGHTS-AND-PERMISSIONS

Gasull, A., Luca, F., and Varona, J. L. (2023). *Three essays on Machin's type formulas **. https://arxiv.org/pdf/2302.00154

Hernandez-Walls, R., and R Hernandez, W. (2025). *The Taylor series and numerical methods: An essential relationship*. https://www.researchgate.net/publication/389561231_The_Taylor_series_and_numerical_methods_An_essential_relationship

Uddin, M., Moheuddin, M., and and, M. K. (2019). A new study of trapezoidal, simpson's 1/3 and simpson's 3/8 rules of numerical integral problems. *Applied Mathematics and Sciences: An International Journal (MathSJ), 6*(4). https://www.academia.edu/download/61660255/6419mathsj0120200102-119397-hkxutn.pdf

Vestermark, H. (2025). *Exploring Taylor series approximation of certain functions*. https://doi.org/http://dx.doi.org/10.13140/RG.2.2.34201.74087