Computational Efficiency and Stability Analysis in Numerical Integration Methods for Stiff Second-Order Dynamic Systems

Michaely Silva[†], Caio Souza[*‡], Israel Duarte[*], Rodolfo Wanderley[*], Andson Balieiro[*], Augusto de Oliveira[†]
[†] *Centro de Tecnologia e Geociências, Universidade Federal de Pernambuco (UFPE), Recife, Brazil.*
[*] *Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife, Brazil.*
[‡] *Universidade de Pernambuco (UPE), Surubim, Brazil.*
{michaely.raiany, augusto.coliveira}@ufpe.br, {cbbs, ibsd, rjpw, amb4}@cin.ufpe.br, caiobruno.souza@upe.br

***Abstract*—Numerical integration of second-order Ordinary Differential Equations (ODEs) is fundamental for accurately modeling dynamic physical systems, particularly those operating under transient disturbances and numerical stiffness. However, existing benchmarks frequently evaluate integrators on generic, non-stiff equations or rely strictly on execution time without considering function evaluation overhead and system complexity. This paper presents a comparative performance evaluation of three numerical integrators, explicit Euler, fourth-order Runge-Kutta (RK4), and the implicit Trapezoidal method, applied to synchronous machine swing equations across three dynamic scenarios with increasing state-vector dimensions (from two to six state variables). Implemented in MATLAB and benchmarked against across extensive runs, the solvers were evaluated on execution time, function evaluations, absolute error, and global mean error. Results show that while Euler diverges under high complexity and RK4 suffers from excessive evaluation overhead, the implicit Trapezoidal scheme achieves the optimal balance. It provides superior numerical stability, minimal execution latency, and bounded error growth, establishing it as the most viable solution for real-time control and fault-sensitive industrial monitoring applications.**

***Index Terms*—Computational Efficiency, Second-Order ODEs, Stiff Dynamic Systems, Transient Stability, Numerical Integration**

I. Introduction

Computational modeling of physical phenomena plays a fundamental role in scientific research and technological development. These phenomena are commonly described by Ordinary Differential Equations (ODEs) and solved using numerical methods [1], [2]. Among them, first- and second-order ODEs are widely used to model dynamic systems with high accuracy and predictive capability [3]. While Artificial Intelligence (AI) is widely used for system modeling, its reliability is often limited by incomplete training data, and its computational overhead introduces significant latency. Even with recent advances in Physics-Informed Neural Networks (PINNs) [4], [5], mathematical inefficiencies and latency persist in continuous simulations of highly stiff systems. Thus, this computational burden may make AI solutions prohibitive in environments with strict real-time requirements, such as machine synchronization in factory automation or in highly interconnected electrical systems.

Numerical methods with high stability and low computational cost are essential for systems requiring instantaneous responses and ultra-fast and predictable controls [6]. Selecting a method compatible with the system dynamics ensures convergence and accurate real-time simulation while satisfying application-specific requirements and error tolerances [7]. Conversely, inaccurate or computationally expensive methods may introduce delays and numerical errors that compromise synchronization, large-scale system stability or even lead to the complete collapse of automated control system. These issues are especially critical in safety-critical real-time control systems and highly coupled dynamic systems, such as those found in aerospace, flight control, robotics, networked control, and industrial automation. In such applications, numerical inaccuracies can lead to trajectory deviations, control failures, or even catastrophic accidents. Therefore, evaluating the stability and computational efficiency of numerical methods is essential to ensure reliable and safe system operation.

The application of numerical integration methods to Ordinary Differential Equations (ODEs) has been extensively studied [8]. However, many studies validate new models using generic ODEs with known exact solutions, overlooking stress factors that impact complex physical systems. Comparative efficiency analyses are limited to superficial execution-time measurements on generic nonlinear equations [9] [10], ignoring the effects of real-world numerical stiffness on performance. Moreover, some works compare software environments without reporting hardware specifications or considering the error tolerances required to ensure system stability [11].

To address these gaps, this paper presents a systematic evaluation of numerical integrators for solving second-order ODEs that describe stiff physical systems. The methods are benchmarked across three systems with different levels of complexity and stiffness, assessing reliability, accuracy, and performance under computational stress. Results show that Euler diverges from the analytical solution over long simulations due to truncation error accumulation. Although the fourth-order Runge-Kutta (RK4) provides high accuracy, its computational cost is increased by four function evaluations per iteration. In contrast, the implicit Trapezoidal achieves

the best trade-off between accuracy and efficiency while maintaining numerical stability for stiff systems.

The remainder of this paper is organized as follows: Section II reviews related work and its contributions. Section III details the proposed simulation algorithm and evaluation metrics. Section IV analyzes the experimental results and their physical implications. Finally, Section V concludes the paper and outlines future research directions.

II. Related Work

Analyzing physical systems through second-order Ordinary Differential Equations (ODEs) is essential for effective planning and control. However, systems with widely varying time scales often exhibit numerical stiffness, demanding high computational cost and execution time of numerical integration to accurately reflect these phenomena and motivating extensive performance evaluation studies [9], [11], [12]. To address this trade-off, recent studies have proposed optimized integration methods. For instance, [13] introduced an adaptive spectral method for linear oscillatory ODEs using a Chebyshev-approximated correction step to reduce residual proportionally to the local frequency, while [14] proposed a four-step trigonometric method for stiff equations, which interpolates basis functions at key points, to improve stability, accuracy, and convergence with lower computational cost.

In addition to the algorithm design, the literature broadly addresses the classical integrators under two aspects: rigorous theoretical foundations and comparative benchmarks. Regarding the former, [15] investigated the convergence properties of deterministic probability-flow samplers, establishing bounds on distribution variance. From a mechanical stability perspective, [16] applied the Picard–Lindelöf and Banach fixed-point theorems to assess system sensitivity to initial conditions and external forces. In the second perspective, many studies emphasize computational cost-efficiency. For instance, [9] and [10] compared traditional methods, such as Euler and fourth-order Runge-Kutta (RK4), with adaptive solvers (e.g., MATLAB's ode45) for nonlinear initial value problems, demonstrating how error and CPU consumption scale over time. Extending these benchmarks across computational environments (MATLAB vs. Python), [11] correlated convergence rates with runtime efficiency. For higher-complexity applications, [12] underscored the critical choice of numerical scheme by evaluating six integrators on gravitational perturbations, using the number of function evaluations required for convergence as the primary performance metric.

Despite these contributions and advantages, applying numerical integration methods to second-order ODEs in stiff physical systems still requires deeper investigation, leaving critical gaps in the literature. First, even when proposing novel solvers, studies including [13] and [14] validate their approaches on generic ODEs with known exact solutions, ignoring the dynamic complexities of real-world physical systems. Second, comparative benchmarks of established integrators [9], [10] also rely on generic nonlinear equations, evaluating performance almost exclusively through execution time. This yields a superficial assessment, as efficiency observed in controlled settings may degrade when exposed to the severe numerical stiffness of real physical systems. Finally, current benchmarks lack rigorous control parameters: cross-software comparisons (e.g., Python vs. MATLAB), such as [11], frequently omit hardware specifications and continuous computational stress parameters.

III. System Model

To evaluate the accuracy and computational efficiency of numerical integrators, a comprehensive evaluation environment was designed, where each solver was initially implemented independently and validated separately to ensure accurate system function evaluations. By using the proposed framework, we analyzed three numerical integrators: Euler, fourth-order Runge-Kutta (RK4), and Trapezoidal. To assess the schemes robustness and how system complexity impacts on solver performance, the governing ODEs were derived from three physical systems, categorized by their complexity level (Case 1, Case 2, and Case 3). The following sections detail the problem formulation (Section III-A), evaluation pipeline (Section III-B), and adopted performance metrics.

A. System Modeling and Dynamics

The systems investigated in this study consist of interconnected synchronous generators designed for continuous power generation, though subject to sudden interruptions from faults or disturbances, exhibit physical complexity and inherent numerical stiffness as the number of machines and their interconnections increases. Their dynamics are governed by the synchronous machine swing equation, given in Eq. (1), which is reformulated into a state-space representation to obtain an equivalent system of first-order ordinary differential equations, as shown in Eq. (2), where ω_s, ω_i, H_i, P_{mi}, P_{ei}, and D_i denote, respectively, the system synchronous speed, the rotor speed, the inertia constant, the mechanical power, the electrical power, and the damping factor of $i{-}th$ machine. The resulting state vector, comprising the rotor angle and angular velocity $[\text{angle}, \quad \text{angular velocity}]^T$, contains the minimum information required to fully describe the system state at any given time. This state-space formulation directly supports Industry 4.0 applications, providing the essential data layer required by digital twins for high-frequency edge computing.

$$\begin{cases} \dfrac{d^2\delta_i}{dt^2} = \dfrac{\omega_s}{2H_i}\left[P_{mi} - P_{ei}(t) - D_i(\omega_i(t) - \omega_s)\right] \end{cases} \tag{1}$$

$$\begin{cases} \dfrac{d\delta_i}{dt} = \omega_i(t) - \omega_s \\ \dfrac{d\omega_i}{dt} = \dfrac{\omega_s}{2H_i}\left[P_{mi} - P_{ei}(t) - D_i(\omega_i(t) - \omega_s)\right] \end{cases} \tag{2}$$

The complexity of the analyzed systems increases progressively across the test cases, with the dimension of the state vector growing from two variables in the single-machine Case 1, to four variables in the two-machine infinite bus system of Case 2, up to six variables in the three-machine,

nine-bus configuration of Case 3. In terms of computational implementation, the system equation is formulated adopting a vector state-space representation, as defined in Eq. (3).

$$\dot{\mathbf{y}} = \frac{d}{dt}\begin{bmatrix}\delta_i \\ \omega_i\end{bmatrix} = \begin{bmatrix}\omega_i \\ \frac{\omega_s}{2H_i} \cdot (P_{mi} - P_{ei} \cdot \sin(\delta_i) - D_i \cdot \omega_i)\end{bmatrix} \quad (3)$$

Simulations follow a sequence that mirrors the occurrence of real-world system events. To achieve this, the model is divided into two operational stages: a function governing the disturbance period and another for the post-fault period. Fault clearance occurs when the fault duration elapses, after which the post-fault function dictates the continuous dynamic behavior of the system. During initialization, the starting state vector is applied to the fault equation.

B. Evaluation Pipeline

The system architecture relies on an orchestrator (illustrated in Fig. 1), which manages executions by calling each integration algorithm as an external function. The evaluation pipeline employs a vectorized implementation, handling system state variables as matrices and updating them in block operations per iteration. Its operational workflow is illustrated in Fig. 2. Upon initialization, the system parameters are setting, including step size (h), time switch for the post-fault period, and the total simulation time. During the system solution and dynamics calculation phases, the implementations compute derivatives and generate a predictor step, applying a corrector step where applicable. Operating within iterative loops, the algorithm advances in increments of step size h until the fault period concludes at $t = T_{switch}$, then transitions to the post-fault formulation and continues iterating until the total simulation time is reached. For implicit solvers utilizing predictor-corrector schemes, each step evaluates the system until the convergence criterion is met, ultimately checking whether state variables have stabilized or the total simulation time has expired.

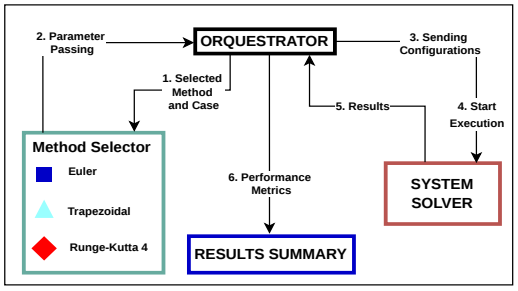


Fig. 1. Solution Architecture Framework

Regarding the orchestrator introduced above, it manages manages executions by calling each integration algorithm as an external function to conduct scalable evaluations. This orchestrator defines the sequence and iteration frequency across a suite of algorithms, each evaluating a specific level of system complexity over a designated duration and number of runs. Upon completing the routine orchestration, the script automatically generates solution plots and populates result

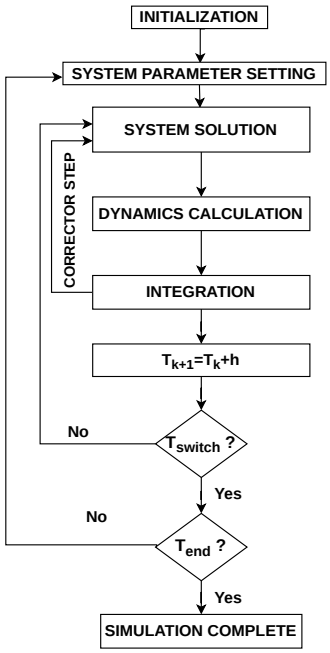


Fig. 2. System Solver Flowchart

tables with computational effort and performance metrics in a results summary.

C. Performance Metrics

The performance analysis of numerical integration methods applied to stiff systems, relies on the trade-off between solution accuracy and computational cost. In this work, the local precision of each method is evaluated using the Absolute Error, defined as the absolute difference between the numerical solution delivered by the algorithm and the reference solution (provided by `ode45` in MATLAB) at the final point of each interval. Because a method may reach the final point despite experiencing trajectory deviations along the solution curve, the Global Mean Error is also analyzed to reflect the overall stability of the method throughout the evaluation. It is calculated as the arithmetic mean of absolute deviations collected across all integration steps. Regarding execution performance two metrics are considered, Computational Time and Computational Effort. The former represents the total CPU processing time (in milliseconds) required to complete the evaluation pipeline. However, as runtime can be influenced by hardware specifications and software versions, the computational effort is adopted alongside execution time to assess the final performance. Computational effort is quantified by the total number of derivative function evaluations $f(t, y)$ performed during the simulation. Although this study adopts the previous presented metrics, our evaluation framework is easily extensible to encompass new ones.

IV. Results and Discussion

The proposed framework is implemented MATLAB and the methods are benchmarked against MATLAB's native solver,

`ode45`, configured with an absolute tolerance of 10^{-6} to serve as a high-precision baseline. The evaluations were structured around four primary metrics (see III-C): execution time and number of function evaluations to quantify computational effort and absolute error and global mean error to measure numerical accuracy and convergence stability.

Across all test scenarios, the system equations were evaluated according to power system stability criteria, which dictates the continuous operational integrity of synchronous machines over time. The algorithms solve the swing equations to model both the application and clearance of a disturbance. This clearing must occur within the Critical Time, which represents the physical limit the system can tolerate before losing synchronism and collapsing. Therefore, the adopted switching times are set below this limit to promote stability, allowing the system to return to a stable operating point or a new equilibrium state. The simulation time parameters are summarized in Table I, which follow studies as [17] [18] [19]. The metrics are apresented in terms of average results obtained considering more than 1,000 independent runs, executed both locally and remotely via the MATLAB web platform.[1]

TABLE I
TIMES PRAMETERS

Case	Critical Time	Switching Time (T_{switch})	Simulated Time	States
1	0.432 s	0.3 s	1 s	2
2	0.225 s	0.2 s	1 s	4
3	0.132 s	0.083 s	2 s	6

A. Computational Effort Analysis

This section examines the impact of system complexity on the execution time of the three methods. Figure 3a illustrates that higher system complexity increases processing time when transitioning from Case 1 to Case 2. For the first case, which involves only two state variables, the Euler and Trapezoidal implementations yield execution times near 10^{-2} s with a step size of $h = 0.01$. Decreasing the step size to $h = 0.001$ increases these times to 0.1025 s and 0.1333 s, respectively. As expected, the fourth-order Runge-Kutta (RK4) code is the most computationally demanding, registering 0.0664 s for the larger step size and 0.7135 s for the smaller step size in Case 1. Although all solvers operate within similar orders of magnitude, their evaluation counts differ significantly: RK4 requires nearly 7 times more function evaluations than Euler code for the smaller step size and almost 3 times more for the larger step size in Case 1. Similar discrepancies are observed in the Case 2.

When solving a system expanded to four state variables (Case 2), the Trapezoidal code proves to be eight times faster than RK4 and 245% faster than Euler code. Interestingly, despite its architectural advantage in execution speed, the Trapezoidal code performs twice as many evaluations as Euler code in Case 2. This highlights the trade-off between theoretical convergence effort and actual processing cost across different complexity levels. In terms of algorithmic overhead, Figure 3b reveals that while the Trapezoidal code achieves the shortest runtime, it demands more evaluations than Euler code. Generally, transitioning from Case 1 to Case 2 causes all solvers to increase their evaluation counts almost proportionally to the increase in state variables. This behavior underscores the efficiency of these solvers during integration, ensuring computational predictability and prompt responsiveness during system fault conditions.

Contrary to expectations, the solvers exhibit tighter control over function evaluations in Case 3. From a computational standpoint, an increase in vector dimensionality would typically suggest higher computational effort and a greater number of evaluations. However, while the per-step algebraic complexity rises, the overall computational burden decreases. Physically, the multi-machine system exhibits a higher equivalent inertia H, which constrains abrupt variations and leads to smaller angular oscillations $\Delta\delta$. These reduced oscillations damp the system dynamics, thereby mitigating numerical stiffness and decreasing the required function evaluations by optimizing the performance of the numerical methods [19]. For instance, using the same step size, the Trapezoidal code requires only 61.2% of the evaluations needed in Case 2 and 76.55% of those in Case 1. This optimization effect stems from the system's structural arrangement, specifically the coupling of machine angles and angular velocities, and how the algorithms handle variable correlations during iteration. Despite expanding to a six-component state vector, the methods no longer contend with severe numerical oscillations previously caused by weak component interdependence; the smoother trajectories facilitate rapid convergence. The Euler implementation, for example, requires only 60% of its Case 1 evaluations to reach a solution. This self-regulating capability is particularly valuable in systems modeled by second-order ODEs, where the complex interactions among state variables can uniquely influence system dynamics during both fault and steady-state conditions.

B. Accuracy Analysis: Absolute and Mean Errors

This section investigates the influence of varying mathematical complexity levels on the attained numerical accuracy. Fig. 4a shows the limitations of Euler implementation become evident when transitioning from Case 1 to Case 2, where the state variables display greater independence and significantly different oscillation patterns. Conversely, the fourth-order Runge-Kutta (RK4) implementation shows the lowest sensitivity to system expansion, maintaining superior stability in absolute error. For instance, in Case 3, RK4 remained stable with errors on the order of 10^{-6} for a step size of $h = 0.001$, two orders of magnitude smaller than those of the Trapezoidal implementation under the same conditions.

Figure 4b also displays the mean error, illustrating the global stability of each method. The RK4 code once again yields the lowest mean error among the three, particularly in

[1]https://matlab.mathworks.com/

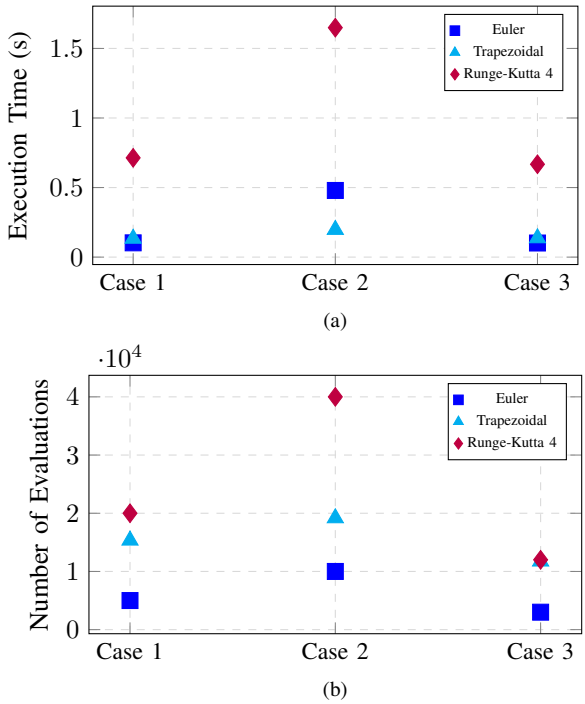


Fig. 3. Comparative analysis of computational effort for numerical integration methods under different levels of system complexity ($h = 0.001$).

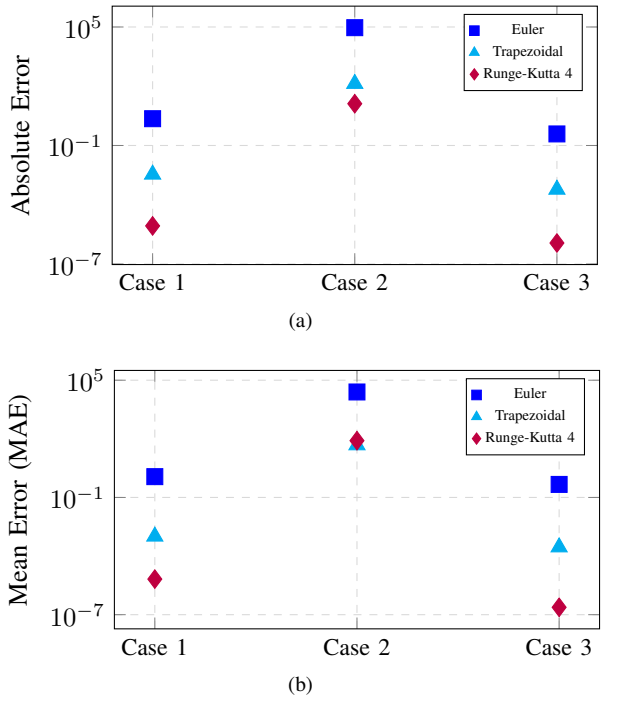


Fig. 4. Performance analysis of numerical accuracy and convergence stability ($h = 0.001$).

Case 3, where it reaches 2.3860×10^{-7} for Machine 1. At this complexity level, the method converges smoothly without numerical oscillations, effectively minimizing residual errors. In contrast, Euler code exhibits the highest error accumulation over time, even at the smallest step size, reaching 2.4980×10^4 for Machine 1 in Case 2. When complexity exceeds the method's stabilization capability, errors lose uniformity and begin to fluctuate. This stands in stark contrast to the Trapezoidal code, which remained stable with a maximum error of 4.7576×10^1 for Machine 1 in Case 2. While the RK4 ensures superior stability by mitigating mean residual errors and eliminating peak absolute errors, its high accuracy sharply contrasts with Euler. The rapid accumulation of numerical divergence in Euler renders it unviable for fault-sensitive systems—in control loops or aerospace applications, such continuous fluctuations can induce phase-shifted torques, destructive oscillations, and structural failures. Conversely, the Trapezoidal consistently keeps both absolute and mean errors bounded, ensuring high predictability and stability.

C. *Efficiency vs. Accuracy Trade-Off*

Combining efficiency and accuracy evaluation, Fig. 5a indicates that pursuing the lowest absolute error inevitably requires longer execution times and a higher number of function evaluations. This trade-off is particularly clear in the fourth-order Runge-Kutta (RK4) code, where achieving errors on the order of 10^{-6} requires up to four times more evaluations than the simplest method. In Case 3, where the ODE features greater complexity among state variables, RK4's robust convergence rate highlights the method's capability to effectively manage strong system coupling. This analysis underscores the importance of balancing solver selection according to system constraints. RK4 demonstrates superior solution stability by mitigating mean errors and eliminating peaks in absolute error. Despite its numerical accuracy, RK4 incurs higher computational overhead and latency, making it less suitable for time-critical power systems where temporal synchronization demands millisecond-level responses for protection schemes.

Conversely, Euler code exhibits acceptable execution times, but its rapid error accumulation precludes its application in fault-sensitive systems. Within a feedback loop, this accumulated error can cause actuators or motors to deliver out-of-phase torque, leading to severe mechanical damage. Similarly, in aerospace applications, such drift could induce Pilot-Induced Oscillations (PIO), leading to structural failure.

On the other hand, the Trapezoidal code provides the optimal trade-off between computational cost and accuracy for control applications, as can be seen in Figure 5b, by delivering high predictability with low processing overhead. Its reduced execution time and lower evaluation count confirm that it satisfies real-time requirements without introducing response delays. Furthermore, its consistently minimized mean and absolute errors guarantee the stability of closed-loop dynamic systems.

V. Conclusions and Future Directions

This work presented a comparative evaluation of numerical integrators applied to solving dynamic systems modeled by second-order ordinary differential equations (ODEs), mapping solution behavior in terms of computational effort and stability

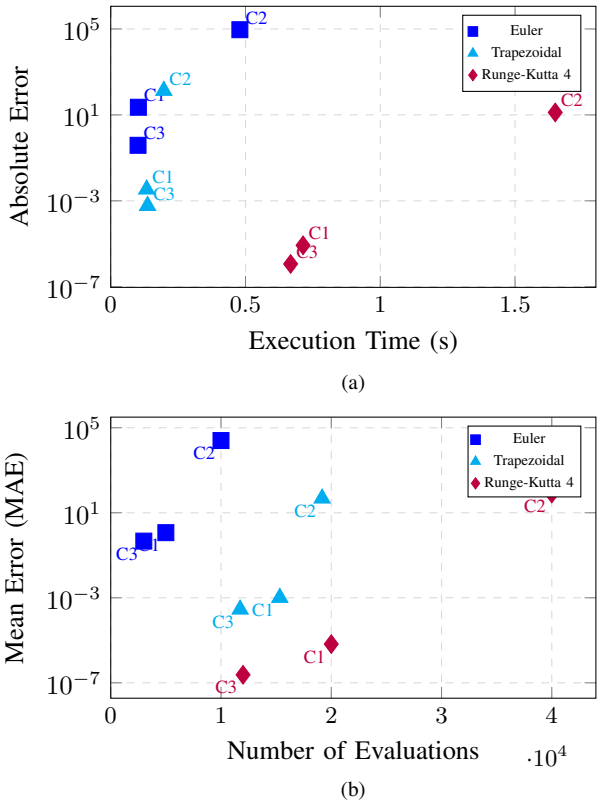


Fig. 5. Trade-off between computational efficiency and numerical accuracy.

as system complexity and stiffness increase. The findings confirm that the pursuit of maximum accuracy using the fourth-order Runge-Kutta (RK4) implementation encounters a severe execution bottleneck due to requiring four function evaluations per iteration, thereby limiting its feasibility under high computational load. Conversely, the initial low cost of the explicit Euler implementation leads to numerical instability and divergence, underscoring the inherent limitations of first-order algorithms in suppressing error propagation within coupled oscillatory systems.

Given the limitations of these extremes, the Trapezoidal implementation proved to be the most computationally efficient solution. As an implicit scheme, it guaranteed the numerical stability required to handle stiff systems while keeping errors strictly bounded. Despite introducing additional overhead within its iterative convergence loop, the code achieved minimal execution times and maintained high fidelity to the analytical solution across the most complex scenarios. Consequently, for monitoring and control applications in dynamic systems, the Trapezoidal excels by delivering high-response fidelity with low computational demand.

In summary, this study demonstrates that selecting an appropriate numerical integrator extends beyond the isolated pursuit of mathematical precision. For real-world physical simulations, balancing reliability, stability, and performance, as provided by the Trapezoidal solver, is the key to ensuring not only model robustness under stress, but also resource efficiency and computational reliability for practical deployment.

References

[1] W. Yang, "Application and solution methods of differential equations in physical modeling," *International Journal of Engineering and Technology Research (IJETR)*, vol. 1, no. 4, pp. 535–544, 2025. [Online]. Available: https://atlantic-press-journals.com/index.php/IJETR/article/view/545

[2] R. Creswell, K. M. Shepherd, B. Lambert, G. R. Mirams, C. L. Lei, S. Tavener, M. Robinson, and D. J. Gavaghan, "Understanding the impact of numerical solvers on inference for differential equation models," *Journal of The Royal Society Interface*, vol. 21, no. 212, p. 2023.0369, 2024.

[3] M. V. B. Santana, "Exact solutions of nonlinear second-order autonomous ordinary differential equations: Application to mechanical systems," *Dynamics*, vol. 3, no. 3, pp. 444–467, aug 2023. [Online]. Available: https://doi.org

[4] G. Fabiani, E. Bollt, C. Siettos, and A. Yannacopoulos, "Stability analysis of physics-informed neural networks for stiff linear differential equations," 08 2024.

[5] L. Laguzet and G. Turinici, "Physics-informed neural networks for coupled stiff transport systems," 2026. [Online]. Available: https://api.semanticscholar.org/CorpusID:289623228

[6] R. F. Gatto *et al.*, "A revisitação dos fundamentos de sistemas de potência na era da geração distribuída: uma análise crítica da estabilidade e controle," *RCMOS-Revista Científica Multidisciplinar O Saber*, vol. 1, no. 2, pp. 1–15, 2024.

[7] S. L. Eshkabilov, "Stiff odes," in *Practical MATLAB Modeling with Simulink: Programming and Simulating Ordinary and Partial Differential Equations*. Berkeley, CA: Apress, apr 2020. [Online]. Available: https://doi.org

[8] S. C. Chapra and R. P. Canale, *Métodos numéricos para engenharia*, 5th ed. São Paulo: McGraw-Hill, 2008.

[9] P. Tumba, A. A. Okeke, and M. A. Islam, "Analysis and comparative study of numerical solutions of initial value problems (ivp) in ordinary differential equations (ode) with euler and runge kutta methods," *American Journal of Engineering Research (AJER)*, vol. 8, no. 8, pp. 40–53, 2019. [Online]. Available: https://ajer.org

[10] H. M. S. Ahmed, "A comparative numerical study of classical and adaptive methods for solving nonlinear differential equations using matlab," *Comprehensive Journal of Science*, vol. 10, no. 39, 2026. [Online]. Available: https://histr.edu.ly

[11] D. M. M. Gasangue, A. M. Laguardia, P. P. M. Espineda, and S. A. Magon, "Optimizing computational efficiency in numerical methods: A comparative analysis of algorithms and implementation in matlab," in *Proceedings of the International Conference on Technological Advancements in Science and Engineering (TSIS)*, 2024.

[12] J. Atallah, R. Woollands, A. Probe, and J. L. Junkins, "Accuracy and efficiency comparison of six numerical integrators for propagating perturbed orbits," *The Journal of the Astronautical Sciences*, vol. 67, no. 2, pp. 511–536, 2020.

[13] I. Bogaert and J. Michiels, "An adaptive spectral method for oscillatory second-order linear odes with frequency-independent cost," *SIAM Journal on Scientific Computing*, vol. 44, no. 3, pp. A1560–A1583, 2022.

[14] J. O. Kuboye, A. Omame, and L. I. Oisadaghe, "A fifth order block methods for solving second-order stiff ordinary differential equations using trigonometric functions and polynomial function as the basis function," *Journal of Applied Mathematics and Computing*, vol. 70, no. 2, pp. 1425–1443, 2024.

[15] S. Chen, G. Daras, and A. G. Dimakis, "Convergence analysis of probability flow ode for score-based generative models," in *International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://openreview.net

[16] X. Wang, Z. Liu, and Q. Zhang, "Stability and numerical solutions for second-order ordinary differential equations with application in mechanical systems," *Mechanics Based Design of Structures and Machines*, vol. 52, no. 4, pp. 2105–2124, 2024.

[17] P. Kundur, *Power System Stability and Control*. New York: McGraw-Hill Education, 1994.

[18] P. M. Anderson and A. A. Fouad, *Power System Control and Stability*, 2nd ed. Piscataway: IEEE Press, 2003.

[19] J. J. Grainger and W. D. Stevenson, *Power System Analysis*, 1st ed. New York: McGraw-Hill, 1994.