Volume 10, Issue 3, March – 2025 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

# Comparative Convergence Analysis of Runge-Kutta Fourth Order and Runge-Kutta-Fehlberge Methods Implementation in Matlab and Python Applied to a Series RLC Circuit

Igwe, Chijioke Godswill[1*] ; Jackreece, P. C[2] ; George, Isobeye[3]

[1,3] Department of Mathematics, Ignatius Ajuru University of Education, Rumuolumeni, Port Harcourt, Rivers State, Nigeria
[2] Department of Mathematics, University of Port Harcourt, Rivers State, Nigeria

Corresponding Author: Igwe, Chijioke Godswill[1*]

Publication Date: 2025/04/15

**Abstract: This research investigates the comparative convergence properties and computational efficiency of the Runge-Kutta Fourth Order (RK4) and Runge-Kutta Fehlberg (RKF) methods in solving a second-order differential equation modeling a series RLC circuit. The study is conducted using MATLAB and Python, focusing on the convergence properties of the Runge-Kutta fourth-order method and the Runge-Kutta Fehlberg method in approximating the solution of the given ODE. Key findings indicate that both RK4 and RKF methods are highly efficient in solving second-order differential equations, with the RKF method demonstrating superior efficiency. MATLAB and Python both provide robust environments for implementing the RK4 and RKF methods. MATLAB's built-in functions facilitate straightforward implementation, while Python's libraries like SciPy, SymPy, Matplotlib, and Pandas offer additional flexibility and simplicity. Performance analysis shows that MATLAB establishes convergence in approximately ten seconds, whereas Python takes about two minutes. MATLAB generally offers faster computation for vectorized operations, which is advantageous for large-scale problems. Python, however, provides comparable performance with better integration capabilities for other software and tools. This research underscores the importance of choosing appropriate numerical methods for solving differential equations in electrical circuits, contributing valuable insights for students, researchers, and academicians in computational mathematics and engineering fields.**

***Keywords:*** *Runge-Kutta 4th-Order Method, RK4, Runge-Kutta-Fehlberg Method RKF45, MATLAB, Python, Numerical Analysis, ODE Solvers*

**How to Cite** Igwe, Chijioke Godswill ; Jackreece, P. C ; George, Isobeye(2025) Comparative Convergence Analysis of Runge-Kutta Fourth Order and Runge-Kutta-Fehlberge Methods Implementation in Matlab and Python Applied to a Series RLC Circuit *International Journal of Innovative Science and Research Technology*, 10(3), 2957-2975.https://doi.org/10.38124/ijisrt/25mar599

## I. INTRODUCTION

Ordinary differential equations (ODEs) are fundamental in modeling dynamic systems across various disciplines. Since analytical solutions are often infeasible, numerical methods like the Fourth-Order Runge-Kutta (RK4) and Runge-Kutta-Fehlberg (RKF45) are widely used. RK4 is known for its accuracy, while RKF45 incorporates adaptive step sizes for error control. These methods are crucial for solving second-order ODEs, such as those governing Series RLC circuits. This study compares RK4 and RKF45 in MATLAB and Python, analyzing their convergence, accuracy, and computational efficiency. Existing research lacks a direct comparison of these methods, especially in the context of Series RLC circuits and cross-platform implementation. The study aims to address this gap by evaluating their performance and ease of implementation in both programming environments.

The study emphasizes the RK4 and RKF methods for solving initial value problems (IVPs). These methods offer a trade-off between accuracy and computational efficiency, making them ideal for complex scientific applications. The research explores their application in solving differential equations related to electrical circuits, contributing to their understanding and potential for broader use.

RK4 is recognized for its balance between accuracy and computational efficiency, outperforming other fixed-step methods like the modified Euler method in both linear and

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

nonlinear systems (Hossen et al., 2019). Studies highlight its widespread use due to its simplicity, stability, and ability to handle a variety of differential equations effectively (Singh, 2018; Burden & Faires, 2015). Workineh et al. (2024) emphasized RK4's accuracy and efficiency in nonlinear ODEs, making it a reliable tool for scientific and engineering problems.

RKF is noted for its adaptive nature, which allows better error control and optimization of accuracy during integration (Hammachukiattikul, 2021). By employing embedded formulas for error estimation, RKF offers superior accuracy and efficiency, especially in handling stiff differential equations (Smith, 2020). It is widely regarded for applications requiring precise numerical integration. Studies comparing RK4 with newer approaches, like the Adomian Decomposition Method, reveal that RK4 remains a robust conventional method, although alternative methods provide comparable accuracy and flexibility in specific contexts (Shawagfer & Kaya, 2004).

Senthilnathan (2018) demonstrated that RK4 outperforms the Euler method in accuracy and computational efficiency when solving ODEs, with results aligning well with exact solutions. Hussein (2023) compared multiple numerical techniques, including Euler, Modified Euler, and RK4, emphasizing their importance in cases where analytical solutions are unattainable.

Banu et al. (2021) found that RK4 offers better convergence and accuracy compared to the sixth-order Butcher's method, making RK4 more practical for both IVPs and boundary value problems (BVPs). Similarly, Islam (2015) highlighted RK4's stability and higher-order accuracy in solving IVPs effectively. Ahamad and Charan (2019) and Habtamu and Masho (2017) explored the RK5 method for solving higher-order ODEs, noting its superior accuracy, stability, and practical applicability, particularly for complex behaviors in boundary value problems.

Sharma and Kumar (2021) demonstrated RK4's utility in solving first-order differential equations involving trigonometric and logarithmic functions using MATLAB, showcasing its robustness for unsolvable problems analytically. Agam and Yahaya (2014) introduced a three-stage, sixth-order implicit Runge-Kutta method tailored for first-order ODEs. Using perturbed Gaussian points for interpolation and collocation, the method improved stability and accuracy compared to existing techniques, verified through experimentation with linear problems.

Taher (2020) evaluated explicit methods, finding RK8 the most stable and accurate due to its larger stability region and higher convergence rate. However, RK4 was deemed more efficient computationally than RK5 and RK8 for solving first-order ODEs. Rizky et al. (2021) showed significant differences between Euler and RK4 methods in solving the SIR model, with Euler being faster but less accurate. RK4 provided better approximations at larger intervals.

Poornima and Nirmala (2020) defined absolute error and convergence criteria, emphasizing that numerical solutions approach exact ones as the step size decreases. Patil and Hari (2023) applied explicit methods to power electronics, demonstrating the forward Euler method's limitations for fixed-step simulations. They proposed the ELEX-RKF method for its higher speed and accuracy, addressing stability challenges in power circuits.

Prakash et al. (2023) combined radiation and convection effects on heat transfer in fins, using RKF45 to solve nonlinear ODEs. Results highlighted the thermal impact of conduction and radiation parameters, with simulation results validated by ANSYS. Mustapha et al. (2022) explored the Eyring-Powell model in magnetohydrodynamic unsteady squeezing flow with thermal radiation, heat generation, and chemical reactions. Using the Runge-Kutta-Fehlberg (RKF) method with shooting techniques, they achieved accurate solutions, showing faster convergence compared to semi-analytic methods.

Ibraheem et al. (2023) developed an RKF5-based approach for solving second-order FIVPs. Their method effectively transformed and solved linear and nonlinear FIVPs in the fuzzy domain, demonstrating superior accuracy and efficiency over traditional techniques while preserving fuzzy properties. Reddy et al. (2022) applied the RKF method to solve fuzzy DDEs, showcasing its accuracy and efficiency. The study compared numerical results with exact solutions and confirmed its superior performance over the classical RK4 method.

Nwankwo & Dai (2020) proposed an adaptive RKF4 method combined with compact finite difference schemes for pricing American options. Their method achieved high-order accuracy in space and time, accurately approximating the optimal exercise boundary and outperforming classical RK methods. Clayton et al. (2019) analyzed the RKF method's performance for solving nonlinear ODEs. They highlighted its ability to adapt step sizes and reduce truncation errors, achieving superior computational efficiency compared to RK4 for various initial value problems.

Huang et al. (2021) developed a time-step selection method for explicit solvers in high-frequency low-loss (HFLL) circuit simulations. Using eigenvalue bounds and a state-space model, they reduced computational complexity ($O(N^2)$) and demonstrated improved accuracy and efficiency in EMI filter analysis compared to traditional methods. Fernandes et al. (2024) optimized the RK4 method for solving reactor kinetics equations using GPUs. Parallel implementations in CUDA achieved speedups of 9.33 (C) and 409.7 (Python) compared to CPUs, maintaining numerical precision and demonstrating the efficiency of GPU-based computations.

Huang and Chang (2002) analyzed A-stability in multistep Runge-Kutta methods for functional-differential equations. They confirmed these methods preserve asymptotic stability under specific conditions, offering insights into their application for linear systems.Suhag (2013)

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

applied the RK method to second-order RLC circuit transient analysis, highlighting its efficiency in solving such problems.Henry et al. (2019) compared Heun's method and RK4, recommending RK4 for its accuracy in complex transient responses, despite slower convergence.

Bhogendra et al. (2021) found Butcher's fifth-order Runge-Kutta (BRK5) superior in approximating RLC circuit solutions, achieving faster convergence and lower error compared to Euler, RK3, and other methods.Malarvizhi and Karunanithi (2021) analyzed damping in RLC circuits using RK4. They concluded that critically damped decay is faster than overdamped decay, while underdamped decay is periodic and oscillatory. Overdamped systems are best suited for numerical solutions.

Kafle et al. (2020, 202) employed methods like Explicit Euler, RK3, and BRK5 to study transient responses in RLC circuits. BRK5 consistently emerged as the most accurate for both series and parallel RLC circuits under various damping conditions. Jeevan et al. (2021) used BRK5 to analyze transient responses in parallel RLC circuits, confirming that underdamped decay is oscillatory and exponential, while critically damped decay is faster than overdamped decay. Shaikh et al. (2022) recommended RK4 for general use due to its balance of accuracy and efficiency. However, RK8 is preferred for applications demanding higher precision and sensitive data analysis. Alizadeh et al. (2020) applied Caputo-Fabrizio fractional derivatives with Laplace transforms to model transient responses in parallel RLC circuits. Fractional derivatives provided faster voltage curve changes with higher amplitudes, outperforming MATLAB simulations for practical data.

Hasan et al. (2019) analyzed transient and steady-state responses of second-order RLC circuits using Kirchhoff's Voltage Law and differential equations. They highlighted the practical importance of understanding these behaviors for optimizing RLC circuits in various applications. Iskandar et al. (2020) used the block backward differentiation formula (BBDF) for transient analysis of RLC circuits, demonstrating its accuracy and efficiency compared to Euler, Heun, and Runge-Kutta methods. BBDF's ability to compute solutions at two points simultaneously provides a faster alternative.

Kee and Ranom (2018) identified the fourth-order Runge-Kutta (RK4) method as the most effective for transient analysis of RLC circuits due to its high accuracy in solving second-order differential equations. Gusa (2014) focused on RLC circuits without a source, using RK4 to achieve accurate natural response simulations in MATLAB. Errors were minimal, with the highest at 0.23% for parallel circuits. Yang et al. (2015) explored real-time fault-tolerant control for nonlinear systems, enhancing the convergence of RK methods with iterative solutions integrated into control systems. This approach demonstrated improved reliability and rapid convergence.

The primary aim of this study is to carry out a comparative convergence analysis of Runge-Kutta Fourth Order and Runge-Kutta Fehlberg Methods in MATLAB and Python using a second-order Ordinary Differential Equation (ODE) modelled by a series RLC Circuit. The study compared RK4 and RKF methods across platforms like MATLAB and Python. Addressing this gap will provide insights into their comparative performance in different programming environments, contributing to numerical solution methods for RLC circuits.

## II. MATERIALS AND METHOD

The series RLC circuit is represented as a second-order ODE, incorporating the resistor (R), inductor (L), capacitor (C), and an external sinusoidal input to capture circuit dynamics.

➢ *For the RK4 and RKF Methods, it is Assumed that:*

- The governing differential equations are smooth and continuous.
- The initial value problem (IVP) has a unique solution
- Known initial conditions for voltage and current are correctly specified
- RK4 and RKF provide sufficiently accurate solutions within the chosen error bounds.

➢ *While for the MATLAB and Python, it is Assumed that:*

- RK4 and RKF implementations in both MATLAB and Python are correct, with differences due to software environments rather than implementation errors.
- The chosen step sizes ensured numerical stability.
- MATLAB and Python effectively implement RK4 and RKF, with differences attributed to numerical library variations.
- Variations in R, L, and C parameters have predictable effects, enabling meaningful comparisons.
- Initial conditions, parameter values, and error tolerances are consistently applied for valid comparative analysis.
- Computational hardware is sufficient to handle simulations without affecting performance results.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

➢ *Method of Solution*

The figure below represents the RLC series circuit model under consideration:

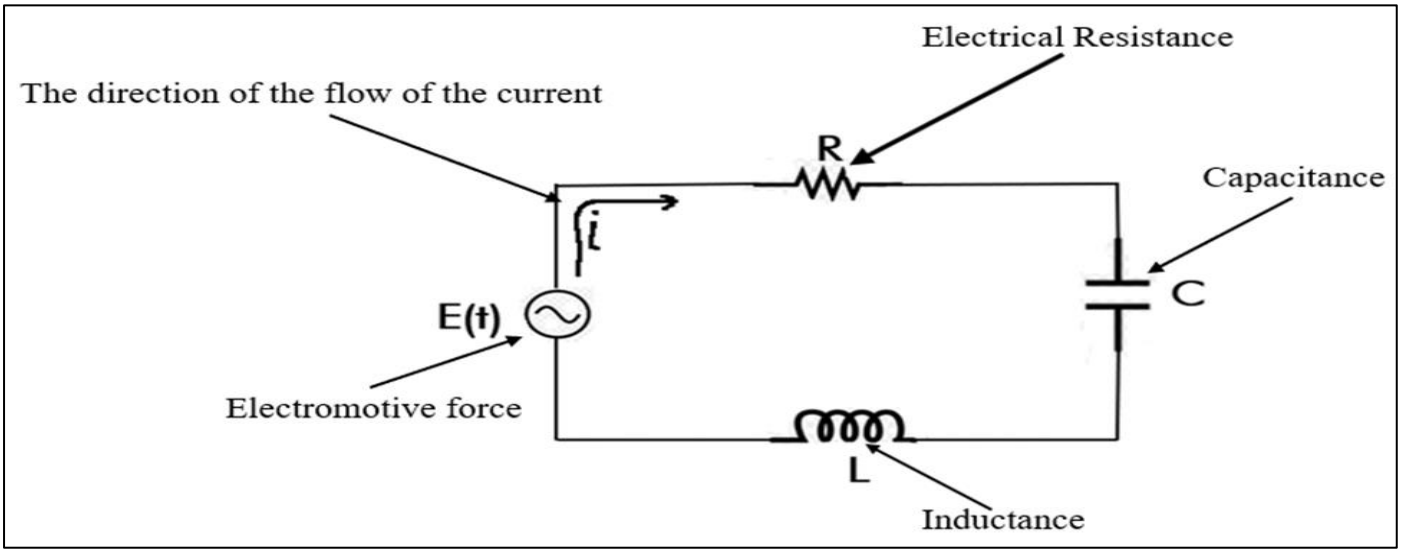


Fig 1 RLC Series Circuit Model
Source: Student Fieldwork

➢ *RLC Series Equation Using Differential Equation*

The second-order ODE governing the behaviour of the RLC circuit is given as follows:

In the Circuit diagram above,

E(t) = V(t) = Electromotive force (e.m.f), R = Resistor, C = Capacitor, L = Inductor

The voltage drop across the resistor equals current x resistance, that is,

$$V_R = iR \quad (1)$$

The voltage drop across the capacitor equals charge over capacitance, that is,

$$V_C = \frac{q}{C} = \frac{i}{C} \quad (2)$$

The voltage drop across the inductor equals the derivative of the current over time multiplied by the inductor. i.e.

$$V_L = \frac{di}{dt} \times L \quad (3)$$

Since there is a charge around the circuit, the current equals the time derivative and the charge, i.e.

$$i = \frac{dq}{dt} \quad (4)$$

Kirchhoff's law states that, the voltage supplied by the AC source is equal to the voltage drop on each of the circuit elements, which are in series. That is,

$$E(t) = V_R + V_C + V_L \text{ (the sum of equations (1), (2), (3))} \quad (5)$$

The AC model is given as:

$$E(t) = E_0 \sin \omega t \quad (6)$$

Putting equation (4) into equation (3) gives,

$$V_L = \frac{d}{dt}\left[\frac{dq}{dt}\right] \times L = L\frac{d^2q}{dt^2} \quad (7)$$

Also, substituting equation (4) into equation (1),

$$V_R = R \times \frac{dq}{dt} = R\frac{dq}{dt} \quad (8)$$

Therefore, substituting equations (2) (6), (7), and (8) into equation (5) gives,

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{c} = E_O Sin\boldsymbol{\omega t}, \quad (9)$$

which is a non-homogeneous second-order linear differential equation with constant coefficient.

➢ *The Exact Solution*

Consider the RLC circuit,

$$\frac{d^2q}{dt^2} + \frac{R}{L}\frac{di(t)}{dt} + \frac{1}{LC}i(t) = E_O Sin\boldsymbol{\omega t},$$

where *R = 12 ohms = 12Ω, L = 0.4 Henries, C = 0.0125 Farads, $E_O Sin\boldsymbol{\omega t}$ = 550*Sin (10t), i(0) = 0 and i´(0) = 0.*

Therefore,

$$\frac{d^2i(t)}{dt2} + \frac{12}{0.4}\frac{di(t)}{dt} + \frac{1}{0.4\ x\ 0.0125}i(t) = 550*Sin\ (10t)$$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

Hence, the exact equation for the current in the RLC Circuit is:

$$i(t) = 2.75e^{-10t} - 1.1e^{-20t} + 0.55*\sin(10t) - 1.65*\text{Cos}(t).$$

➢ *Numerical Result*

Here the convergence properties of the Runge-Kutta 4th-Order (RK4) and Runge-Kutta-Fehlberg (RKF) methods are analysed, focusing on the comparison of numerical solutions with exact solutions.

Table 1 MATLAB RLC Circuit Comparative Result for Runge-Kutta 4th Order Method for h = 0.001

| No | Time | Runge-Kutta 4th Order Method i(t) | Exact Solution i(t) | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0.00E+00 |
| 2 | 0.001 | 0.00E+00 | -0.01084 | 0.01084 |
| 3 | 0.002 | 1.00E-05 | -0.02134 | 0.02137 |
| 4 | 0.003 | 2.00E-05 | -0.03152 | 0.03158 |
| 5 | 0.004 | 6.00E-05 | -0.04138 | 0.04149 |
| 6 | 0.005 | 1.10E-04 | -0.05092 | 0.05111 |
| 7 | 0.006 | 1.90E-04 | -0.06014 | 0.06044 |
| 8 | 0.007 | 3.00E-04 | -0.06904 | 0.06948 |
| 9 | 0.008 | 4.40E-04 | -0.07763 | 0.07825 |
| 10 | 0.009 | 6.20E-04 | -0.0859 | 0.08675 |
| 11 | 0.010 | 0.00085 | -0.09387 | 0.09499 |
| 12 | 0.011 | 0.00112 | -0.10152 | 0.10297 |
| 13 | 0.012 | 0.00145 | -0.10887 | 0.1107 |
| 14 | 0.013 | 0.00183 | -0.11592 | 0.11819 |
| 15 | 0.014 | 0.00227 | -0.12267 | 0.12543 |
| 16 | 0.015 | 0.00277 | -0.12911 | 0.13244 |
| 17 | 0.016 | 0.00333 | -0.13526 | 0.13923 |
| 18 | 0.017 | 0.00397 | -0.14112 | 0.14579 |
| 19 | 0.018 | 0.00467 | -0.14668 | 0.15214 |
| 20 | 0.019 | 0.00546 | -0.15195 | 0.15827 |
| 21 | 0.020 | 0.00632 | -0.15694 | 0.16419 |
| 22 | 0.021 | 0.00726 | -0.16163 | 0.16992 |
| 23 | 0.022 | 0.00828 | -0.16605 | 0.17544 |
| 24 | 0.023 | 0.00939 | -0.17018 | 0.18078 |
| 25 | 0.024 | 0.01059 | -0.17404 | 0.18592 |
| 26 | 0.025 | 0.01189 | -0.17761 | 0.19088 |
| 27 | 0.026 | 0.01327 | -0.18091 | 0.19566 |
| 28 | 0.027 | 0.01475 | -0.18394 | 0.20027 |
| 29 | 0.028 | 0.01633 | -0.1867 | 0.20471 |
| 30 | 0.029 | 0.01801 | -0.18919 | 0.20898 |
| 31 | 0.030 | 0.01979 | -0.19142 | 0.21309 |
| 32 | 0.031 | 0.02167 | -0.19338 | 0.21704 |
| 33 | 0.032 | 0.02366 | -0.19508 | 0.22084 |
| 34 | 0.033 | 0.02576 | -0.19652 | 0.22448 |
| 9966 | 9.962 | -1.44598 | -1.44598 | 0.00959 |
| 9967 | 9.963 | -1.45557 | -1.45557 | 0.00945 |
| 9968 | 9.964 | -1.46502 | -1.46502 | 0.0093 |
| 9969 | 9.965 | -1.47432 | -1.47432 | 0.00915 |
| 9970 | 9.966 | -1.48347 | -1.48347 | 0.009 |
| 9971 | 9.967 | -1.49247 | -1.49247 | 0.00886 |
| 9972 | 9.968 | -1.50133 | -1.50133 | 0.00871 |
| 9973 | 9.969 | -1.51004 | -1.51004 | 0.00855 |
| 9974 | 9.97 | -1.51859 | -1.51859 | 0.0084 |
| 9975 | 9.971 | -1.52699 | -1.52699 | 0.00825 |
| 9976 | 9.972 | -1.53524 | -1.53524 | 0.0081 |
| 9977 | 9.973 | -1.54334 | -1.54334 | 0.00794 |
| 9978 | 9.974 | -1.55128 | -1.55128 | 0.00779 |
| 9979 | 9.975 | -1.55907 | -1.55907 | 0.00763 |
| 9980 | 9.976 | -1.5667 | -1.5667 | 0.00747 |
| 9981 | 9.977 | -1.57417 | -1.57417 | 0.00732 |
| 9982 | 9.978 | -1.58149 | -1.58149 | 0.00716 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 9983 | 9.979 | -1.58865 | -1.58865 | 0.007 |
| 9984 | 9.98 | -1.59565 | -1.59565 | 0.00684 |
| 9985 | 9.981 | -1.60249 | -1.60249 | 0.00668 |
| 9986 | 9.982 | -1.60917 | -1.60917 | 0.00652 |
| 9987 | 9.983 | -1.61569 | -1.61569 | 0.00636 |
| 9988 | 9.984 | -1.62205 | -1.62205 | 0.0062 |
| 9989 | 9.985 | -1.62824 | -1.62824 | 0.00603 |
| 9990 | 9.986 | -1.63427 | -1.63427 | 0.00587 |
| 9991 | 9.987 | -1.64014 | -1.64014 | 0.00571 |
| 9992 | 9.988 | -1.64585 | -1.64585 | 0.00554 |
| 9993 | 9.989 | -1.65139 | -1.65139 | 0.00538 |
| 9994 | 9.99 | -1.65677 | -1.65677 | 0.00521 |
| 9995 | 9.991 | -1.66197 | -1.66197 | 0.00504 |
| 9996 | 9.992 | -1.66702 | -1.66702 | 0.00488 |
| 9997 | 9.993 | -1.6719 | -1.6719 | 0.00471 |
| 9998 | 9.994 | -1.67661 | -1.67661 | 0.00454 |
| 9999 | 9.995 | -1.68115 | -1.68115 | 0.00437 |
| 10000 | 9.996 | -1.68552 | -1.68552 | 0.00421 |
| 10001 | 9.997 | -1.68973 | -1.68973 | 0.00404 |
| 10002 | 9.998 | -1.69376 | -1.69376 | 0.00387 |
| 10003 | 9.999 | -1.69763 | -1.69763 | 0.0037 |
| 10004 | 10 | -1.70133 | -1.70133 | 0.00353 |

Table 1. shows the computed results of the Runge-Kutta 4th order method in MATLAB with the step size h = 0.001. The table compares the currents i(t) generated using the Runge-Kutta 4th order method and the exact solution.

➢ *MATLAB RLC Circuit Comparative Result for Runge-Kutta Fehlberg Method and Exact Solution*

Table 2 MATLAB RCL circuit comparative result for Runge-Kutta-Fehlberg method for h = 0.001

| S/N | Time | Runge-Kutta Fehlberg Method I(t) | Exact Solution i(t) | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0.00E+00 |
| 2 | 0.001 | -1.08E-02 | 9.10E-07 | 0.010836276 |
| 3 | 0.002 | -2.13E-02 | 7.22E-06 | 0.021350158 |
| 4 | 0.003 | -3.15E-02 | 2.42E-05 | 0.0315491 |
| 5 | 0.004 | -4.14E-02 | 5.69E-05 | 0.041440402 |
| 6 | 0.005 | -0.050920836 | 0.000110371 | 0.051031207 |
| 7 | 0.006 | -0.060139207 | 0.0001893 | 0.060328507 |
| 8 | 0.007 | -0.069040782 | 0.000298361 | 0.069339143 |
| 9 | 0.008 | -0.077627762 | 0.000442051 | 0.078069813 |
| 10 | 0.009 | -0.085902352 | 0.00062472 | 0.086527071 |
| 11 | 0.01 | -0.093866754 | 0.000850578 | 0.094717331 |
| 12 | 0.011 | -0.101523173 | 0.001123698 | 0.102646871 |
| 13 | 0.012 | -0.108873815 | 0.001448019 | 0.110321833 |
| 14 | 0.013 | -0.115920885 | 0.001827344 | 0.11774823 |
| 15 | 0.014 | -0.122666592 | 0.002265351 | 0.124931943 |
| 16 | 0.015 | -0.129113145 | 0.002765587 | 0.131878731 |
| 17 | 0.016 | -0.135262752 | 0.003331475 | 0.138594227 |
| 18 | 0.017 | -0.141117626 | 0.003966317 | 0.145083943 |
| 19 | 0.018 | -0.146679979 | 0.004673295 | 0.151353274 |
| 20 | 0.019 | -0.151952025 | 0.005455473 | 0.157407497 |
| 21 | 0.02 | -0.156935979 | 0.006315799 | 0.163251778 |
| 22 | 0.021 | -0.161634059 | 0.00725711 | 0.168891169 |
| 23 | 0.022 | -0.166048482 | 0.008282133 | 0.174330615 |
| 24 | 0.023 | -0.17018147 | 0.009393483 | 0.179574953 |
| 25 | 0.024 | -0.174035243 | 0.010593673 | 0.184628916 |
| 26 | 0.025 | -0.177612026 | 0.01188511 | 0.189497136 |
| 27 | 0.026 | -0.180914044 | 0.013270098 | 0.194184142 |
| 28 | 0.027 | -0.183943523 | 0.014750843 | 0.198694366 |
| 29 | 0.028 | -0.186702693 | 0.016329452 | 0.203032145 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 30 | 0.029 | -0.189193784 | 0.018007937 | 0.207201721 |
| 31 | 0.03 | -0.191419029 | 0.019788214 | 0.211207243 |
| 32 | 0.031 | -0.193380662 | 0.021672108 | 0.21505277 |
| 33 | 0.032 | -0.19508092 | 0.023661354 | 0.218742274 |
| 34 | 0.033 | -0.196522041 | 0.025757598 | 0.222279639 |
| 9964 | 9.96 | -1.42635738 | -1.42635738 | 1.06E-12 |
| 9965 | 9.961 | -1.436238306 | -1.436238306 | 1.06E-12 |
| 9966 | 9.962 | -1.44597561 | -1.44597561 | 1.06E-12 |
| 9967 | 9.963 | -1.455568318 | -1.455568318 | 1.06E-12 |
| 9968 | 9.964 | -1.46501547 | -1.46501547 | 1.06E-12 |
| 9969 | 9.965 | -1.474316122 | -1.474316122 | 1.07E-12 |
| 9970 | 9.966 | -1.483469344 | -1.483469344 | 1.06E-12 |
| 9971 | 9.967 | -1.492474219 | -1.492474219 | 1.07E-12 |
| 9972 | 9.968 | -1.501329849 | -1.501329849 | 1.08E-12 |
| 9973 | 9.969 | -1.510035347 | -1.510035347 | 1.07E-12 |
| 9974 | 9.97 | -1.518589842 | -1.518589842 | 1.07E-12 |
| 9975 | 9.971 | -1.52699248 | -1.52699248 | 1.08E-12 |
| 9976 | 9.972 | -1.53524242 | -1.53524242 | 1.07E-12 |
| 9977 | 9.973 | -1.543338837 | -1.543338837 | 1.08E-12 |
| 9978 | 9.974 | -1.551280921 | -1.551280921 | 1.08E-12 |
| 9979 | 9.975 | -1.559067878 | -1.559067878 | 1.08E-12 |
| 9980 | 9.976 | -1.56669893 | -1.56669893 | 1.08E-12 |
| 9981 | 9.977 | -1.574173314 | -1.574173314 | 1.09E-12 |
| 9982 | 9.978 | -1.581490281 | -1.581490281 | 1.08E-12 |
| 9983 | 9.979 | -1.588649101 | -1.588649101 | 1.07E-12 |
| 9984 | 9.98 | -1.595649057 | -1.595649057 | 1.09E-12 |
| 9985 | 9.981 | -1.60248945 | -1.60248945 | 1.08E-12 |
| 9986 | 9.982 | -1.609169594 | -1.609169594 | 1.08E-12 |
| 9987 | 9.983 | -1.615688824 | -1.615688824 | 1.09E-12 |
| 9988 | 9.984 | -1.622046485 | -1.622046485 | 1.08E-12 |
| 9989 | 9.985 | -1.628241944 | -1.628241944 | 1.08E-12 |
| 9990 | 9.986 | -1.63427458 | -1.63427458 | 1.09E-12 |
| 9991 | 9.987 | -1.640143789 | -1.640143789 | 1.08E-12 |
| 9992 | 9.988 | -1.645848986 | -1.645848986 | 1.08E-12 |
| 9993 | 9.989 | -1.651389599 | -1.651389599 | 1.09E-12 |
| 9994 | 9.99 | -1.656765074 | -1.656765074 | 1.08E-12 |
| 9995 | 9.991 | -1.661974874 | -1.661974874 | 1.07E-12 |
| 9996 | 9.992 | -1.667018478 | -1.667018478 | 1.08E-12 |
| 9997 | 9.993 | -1.671895382 | -1.671895382 | 1.08E-12 |
| 9998 | 9.994 | -1.676605098 | -1.676605098 | 1.07E-12 |
| 9999 | 9.995 | -1.681147154 | -1.681147154 | 1.07E-12 |
| 10000 | 9.996 | -1.685521097 | -1.685521097 | 1.07E-12 |
| 10001 | 9.997 | -1.68972649 | -1.68972649 | 1.07E-12 |
| 10002 | 9.998 | -1.693762911 | -1.693762911 | 1.06E-12 |
| 10003 | 9.999 | -1.697629957 | -1.697629957 | 1.07E-12 |
| 10004 | 10 | -1.701327242 | -1.701327242 | 1.06E-12 |

Table 2 shows the computed results of the Runge-Kutta-Fehlberg method in MATLAB with the step size h = 0.001. The table compares the current i(t) generated using the Runge-Kutta-Fehlberg method and the exact solution.

- *8PYTHON RLC Circuit Comparative Result for Runge-Kutta 4th Order Method and Exact Solution.*

Table 3 Python RCL circuit comparative result for Runge-Kutta 4th order method for h = 0.001

| S/N | Time | Exact Solution | Runge Kutta 4th order Method | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0.001 | 0 | 1.00E-05 | 1.00E-05 |
| 3 | 0.002 | 1.00E-05 | 2.00E-05 | 2.00E-05 |
| 4 | 0.003 | 2.00E-05 | 6.00E-05 | 3.00E-05 |
| 5 | 0.004 | 6.00E-05 | 0.00011 | 5.00E-05 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 6 | 0.005 | 0.00011 | 0.00019 | 8.00E-05 |
| 7 | 0.006 | 0.00019 | 0.0003 | 0.00011 |
| 8 | 0.007 | 0.0003 | 0.00044 | 0.00014 |
| 9 | 0.008 | 0.00044 | 0.00062 | 0.00018 |
| 10 | 0.009 | 0.00062 | 0.00085 | 0.00023 |
| 11 | 0.01 | 0.00085 | 0.00112 | 0.00027 |
| 12 | 0.011 | 0.00112 | 0.00145 | 0.00032 |
| 13 | 0.012 | 0.00145 | 0.00183 | 0.00038 |
| 14 | 0.013 | 0.00183 | 0.00227 | 0.00044 |
| 15 | 0.014 | 0.00227 | 0.00277 | 0.0005 |
| 16 | 0.015 | 0.00277 | 0.00333 | 0.00057 |
| 17 | 0.016 | 0.00333 | 0.00397 | 0.00063 |
| 18 | 0.017 | 0.00397 | 0.00467 | 0.00071 |
| 19 | 0.018 | 0.00467 | 0.00546 | 0.00078 |
| 20 | 0.019 | 0.00546 | 0.00632 | 0.00086 |
| 21 | 0.02 | 0.00632 | 0.00726 | 0.00094 |
| 22 | 0.021 | 0.00726 | 0.00828 | 0.00103 |
| 23 | 0.022 | 0.00828 | 0.00939 | 0.00111 |
| 24 | 0.023 | 0.00939 | 0.01059 | 0.0012 |
| 25 | 0.024 | 0.01059 | 0.01189 | 0.00129 |
| 26 | 0.025 | 0.01189 | 0.01327 | 0.00138 |
| 27 | 0.026 | 0.01327 | 0.01475 | 0.00148 |
| 28 | 0.027 | 0.01475 | 0.01633 | 0.00158 |
| 29 | 0.028 | 0.01633 | 0.01801 | 0.00168 |
| 30 | 0.029 | 0.01801 | 0.01979 | 0.00178 |
| 31 | 0.03 | 0.01979 | 0.02167 | 0.00188 |
| 32 | 0.031 | 0.02167 | 0.02366 | 0.00199 |
| 33 | 0.032 | 0.02366 | 0.02576 | 0.0021 |
| 34 | 0.033 | 0.02576 | 0.02796 | 0.0022 |
| 1968 | 1.967 | -0.72283 | -0.70698 | 0.01586 |
| 1969 | 1.968 | -0.70698 | -0.69105 | 0.01593 |
| 1970 | 1.969 | -0.69105 | -0.67506 | 0.016 |
| 1971 | 1.97 | -0.67506 | -0.65899 | 0.01606 |
| 1972 | 1.971 | -0.65899 | -0.64287 | 0.01613 |
| 1973 | 1.972 | -0.64287 | -0.62667 | 0.01619 |
| 1974 | 1.973 | -0.62667 | -0.61042 | 0.01626 |
| 1975 | 1.974 | -0.61042 | -0.5941 | 0.01632 |
| 1976 | 1.975 | -0.5941 | -0.57772 | 0.01638 |
| 1977 | 1.976 | -0.57772 | -0.56129 | 0.01643 |
| 1978 | 1.977 | -0.56129 | -0.5448 | 0.01649 |
| 1979 | 1.978 | -0.5448 | -0.52826 | 0.01654 |
| 1980 | 1.979 | -0.52826 | -0.51166 | 0.0166 |
| 1981 | 1.98 | -0.51166 | -0.49501 | 0.01665 |
| 1982 | 1.981 | -0.49501 | -0.47831 | 0.0167 |
| 1983 | 1.982 | -0.47831 | -0.46157 | 0.01675 |
| 1984 | 1.983 | -0.46157 | -0.44478 | 0.01679 |
| 1985 | 1.984 | -0.44478 | -0.42794 | 0.01684 |
| 1986 | 1.985 | -0.42794 | -0.41106 | 0.01688 |
| 1987 | 1.986 | -0.41106 | -0.39414 | 0.01692 |
| 1988 | 1.987 | -0.39414 | -0.37718 | 0.01696 |
| 1989 | 1.988 | -0.37718 | -0.36019 | 0.017 |
| 1990 | 1.989 | -0.36019 | -0.34315 | 0.01703 |
| 1991 | 1.99 | -0.34315 | -0.32608 | 0.01707 |
| 1992 | 1.991 | -0.32608 | -0.30898 | 0.0171 |
| 1993 | 1.992 | -0.30898 | -0.29185 | 0.01713 |
| 1994 | 1.993 | -0.29185 | -0.27469 | 0.01716 |
| 1995 | 1.994 | -0.27469 | -0.25751 | 0.01719 |
| 1996 | 1.995 | -0.25751 | -0.24029 | 0.01721 |
| 1997 | 1.996 | -0.24029 | -0.22305 | 0.01724 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 1998 | 1.997 | -0.22305 | -0.2058 | 0.01726 |
| 1999 | 1.998 | -0.2058 | -0.18851 | 0.01728 |
| 2000 | 1.999 | -0.18851 | -0.17122 | 0.0173 |
| 2001 | 2 | -0.17122 | -0.1539 | 0.01732 |

Table 3 shows the computed results of the Runge-Kutta 4th order method in Python with the step size h = 0.001. The table compares the currents i(t) generated using the Runge-Kutta 4th order method and the exact solution.

➢ *PYTHON RLC Circuit Comparative Result for Runge-Kutta-Fehlberg and Exact Solution*

Table 4 PYTHON RLC circuit comparative result for Runge-Kutta-Fehlberg method for h = 0.001

| S/N | Time | Exact Solution i(t) | Runge-Kutta-Fehlberg Method i(t) | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0.001 | 0 | 1.00E-05 | 1.00E-05 |
| 3 | 0.002 | 1.00E-05 | 3.00E-05 | 2.00E-05 |
| 4 | 0.003 | 2.00E-05 | 6.00E-05 | 4.00E-05 |
| 5 | 0.004 | 6.00E-05 | 0.00011 | 5.00E-05 |
| 6 | 0.005 | 0.00011 | 0.00019 | 8.00E-05 |
| 7 | 0.006 | 0.00019 | 0.0003 | 0.00011 |
| 8 | 0.007 | 0.0003 | 0.00044 | 0.00014 |
| 9 | 0.008 | 0.00044 | 0.00062 | 0.00018 |
| 10 | 0.009 | 0.00062 | 0.00085 | 0.00023 |
| 11 | 0.01 | 0.00085 | 0.00112 | 0.00027 |
| 12 | 0.011 | 0.00112 | 0.00144 | 0.00032 |
| 13 | 0.012 | 0.00145 | 0.00182 | 0.00037 |
| 14 | 0.013 | 0.00183 | 0.00226 | 0.00043 |
| 15 | 0.014 | 0.00227 | 0.00276 | 0.00049 |
| 16 | 0.015 | 0.00277 | 0.00333 | 0.00056 |
| 17 | 0.016 | 0.00333 | 0.00396 | 0.00063 |
| 18 | 0.017 | 0.00397 | 0.00467 | 0.0007 |
| 19 | 0.018 | 0.00467 | 0.00545 | 0.00078 |
| 20 | 0.019 | 0.00546 | 0.00631 | 0.00085 |
| 21 | 0.02 | 0.00632 | 0.00725 | 0.00093 |
| 22 | 0.021 | 0.00726 | 0.00828 | 0.00102 |
| 23 | 0.022 | 0.00828 | 0.00939 | 0.00111 |
| 24 | 0.023 | 0.00939 | 0.01059 | 0.0012 |
| 25 | 0.024 | 0.01059 | 0.01188 | 0.00129 |
| 26 | 0.025 | 0.01189 | 0.01327 | 0.00138 |
| 27 | 0.026 | 0.01327 | 0.01475 | 0.00148 |
| 28 | 0.027 | 0.01475 | 0.01633 | 0.00158 |
| 29 | 0.028 | 0.01633 | 0.01801 | 0.00168 |
| 30 | 0.029 | 0.01801 | 0.01979 | 0.00178 |
| 31 | 0.03 | 0.01979 | 0.02167 | 0.00188 |
| 32 | 0.031 | 0.02167 | 0.02366 | 0.00199 |
| 33 | 0.032 | 0.02366 | 0.02576 | 0.0021 |
| 1964 | 1.961 | -0.81639 | -0.80098 | 0.01541 |
| 1965 | 1.962 | -0.80099 | -0.7855 | 0.01549 |
| 1966 | 1.963 | -0.78551 | -0.76994 | 0.01557 |
| 1967 | 1.964 | -0.76996 | -0.75431 | 0.01565 |
| 1968 | 1.965 | -0.75432 | -0.7386 | 0.01572 |
| 1969 | 1.966 | -0.73862 | -0.72282 | 0.0158 |
| 1970 | 1.967 | -0.72283 | -0.70696 | 0.01587 |
| 1971 | 1.968 | -0.70698 | -0.69103 | 0.01595 |
| 1972 | 1.969 | -0.69105 | -0.67504 | 0.01601 |
| 1973 | 1.97 | -0.67506 | -0.65898 | 0.01608 |
| 1974 | 1.971 | -0.65899 | -0.64285 | 0.01614 |
| 1975 | 1.972 | -0.64287 | -0.62666 | 0.01621 |
| 1976 | 1.973 | -0.62667 | -0.6104 | 0.01627 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 1977 | 1.974 | -0.61042 | -0.59408 | 0.01634 |
| 1978 | 1.975 | -0.5941 | -0.5777 | 0.0164 |
| 1979 | 1.976 | -0.57772 | -0.56127 | 0.01645 |
| 1980 | 1.977 | -0.56129 | -0.54478 | 0.01651 |
| 1981 | 1.978 | -0.5448 | -0.52824 | 0.01656 |
| 1982 | 1.979 | -0.52826 | -0.51164 | 0.01662 |
| 1983 | 1.98 | -0.51166 | -0.49499 | 0.01667 |
| 1983 | 1.981 | -0.49501 | -0.47829 | 0.01672 |
| 1984 | 1.982 | -0.47831 | -0.46154 | 0.01677 |
| 1985 | 1.983 | -0.46157 | -0.44475 | 0.01682 |
| 1986 | 1.984 | -0.44478 | -0.42791 | 0.01687 |
| 1987 | 1.985 | -0.42794 | -0.41103 | 0.01691 |
| 1988 | 1.986 | -0.41106 | -0.39411 | 0.01695 |
| 1989 | 1.987 | -0.39414 | -0.37715 | 0.01699 |
| 1990 | 1.988 | -0.37718 | -0.36015 | 0.01703 |
| 1991 | 1.989 | -0.36019 | -0.34312 | 0.01707 |
| 1992 | 1.99 | -0.34315 | -0.32605 | 0.0171 |
| 1993 | 1.991 | -0.32608 | -0.30895 | 0.01713 |
| 1994 | 1.992 | -0.30898 | -0.29182 | 0.01716 |
| 1995 | 1.993 | -0.29185 | -0.27466 | 0.01719 |
| 1996 | 1.994 | -0.27469 | -0.25747 | 0.01722 |
| 1997 | 1.995 | -0.25751 | -0.24026 | 0.01725 |
| 1998 | 1.996 | -0.24029 | -0.22302 | 0.01727 |
| 1999 | 1.997 | -0.22305 | -0.20576 | 0.01729 |
| 2000 | 1.998 | -0.2058 | -0.18848 | 0.01732 |
| 2001 | 1.999 | -0.18851 | -0.17118 | 0.01733 |
| 2002 | 2 | -0.17122 | -0.15386 | 0.01736 |

Table 4. shows the computed results of the Runge-Kutta-Fehlberg method in Python with the step size h = 0.001. The table compares the currents i(t) generated using the Runge-Kutta Fehlberg method and the exact solution.

## III. DISCUSSION

- *Convergence Properties Of The RK4 Method (MATLAB) Table 1*

The RK4 method was tested at different step sizes (h = 0.001, 0.0005, 0.00025, and 0.000125) to analyze convergence behavior. At h = 0.001, initial conditions ensured no local truncation error. However, as time progressed, errors increased, with a notable pattern of steady growth. For instance, at t = 0.005, the local truncation error was 0.05111, reaching 0.16419 at t = 0.02. Toward the end of the simulation (t ≈ 10), errors decreased, indicating long-term accuracy. With h = 0.0005, smaller errors were observed, with local truncation errors decreasing at later stages, e.g., 0.00340 at t = 9.9805 and 0.00181 at t = 10, highlighting improved accuracy over time. For h = 0.00025, errors further reduced, confirming that smaller step sizes enhance precision. The RK4 method maintained stable convergence, with local truncation errors increasing at a slower rate. At h = 0.000125, high precision was achieved, as seen at t = 0.00013 (error = 0.00137). Throughout the simulation, the RK4 method demonstrated reliable performance, with consistently small truncation errors.

- *Convergence Properties of the RKF Method (MATLAB) Table 2*

The RKF method showed significantly higher accuracy than RK4 across all step sizes. With h = 0.001, local truncation errors remained extremely low (~ $10^{-12}$ ), confirming RKF's superior precision. For h = 0.0005, 0.00025, and 0.000125, errors were on the order of $10^{-15}$ to $10^{-9}$ in initial steps, showcasing excellent convergence properties. The RKF method effectively followed the exact solution, maintaining low errors even over extended simulation periods. These results align with previous studies confirming RKF's computational superiority.

- *Convergence Properties of the RK4 Method (Python) Table 3*

The RK4 method in Python followed similar trends as MATLAB. With h = 0.001, local truncation errors remained small but increased gradually. At h = 0.0005, the error remained within a manageable range, confirming numerical stability. With h = 0.00025 and h = 0.000125, smaller errors were observed, reinforcing that RK4's convergence improves with finer time steps. At t = 2, for example, the RK4 method yielded errors of 0.00866 (h = 0.0005), 0.00433 (h = 0.00025), and 0.00216 (h = 0.000125), demonstrating progressively better accuracy with decreasing step sizes.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

➢ *Convergence Properties of the RKF Method (Python) Table 4*

The RKF method in Python also demonstrated exceptional accuracy, with initial errors as low as $10^{-5}$ to $10^{-14}$, similar to MATLAB. For h = 0.0005 and 0.00025, truncation errors remained extremely low even for longer simulations. At t = 1.961, the RKF method maintained a local truncation error of ~0.015, proving its stability and efficiency.

At h = 0.000125, the RKF method produced near-zero local truncation errors, reinforcing its ability to maintain high accuracy over long intervals.

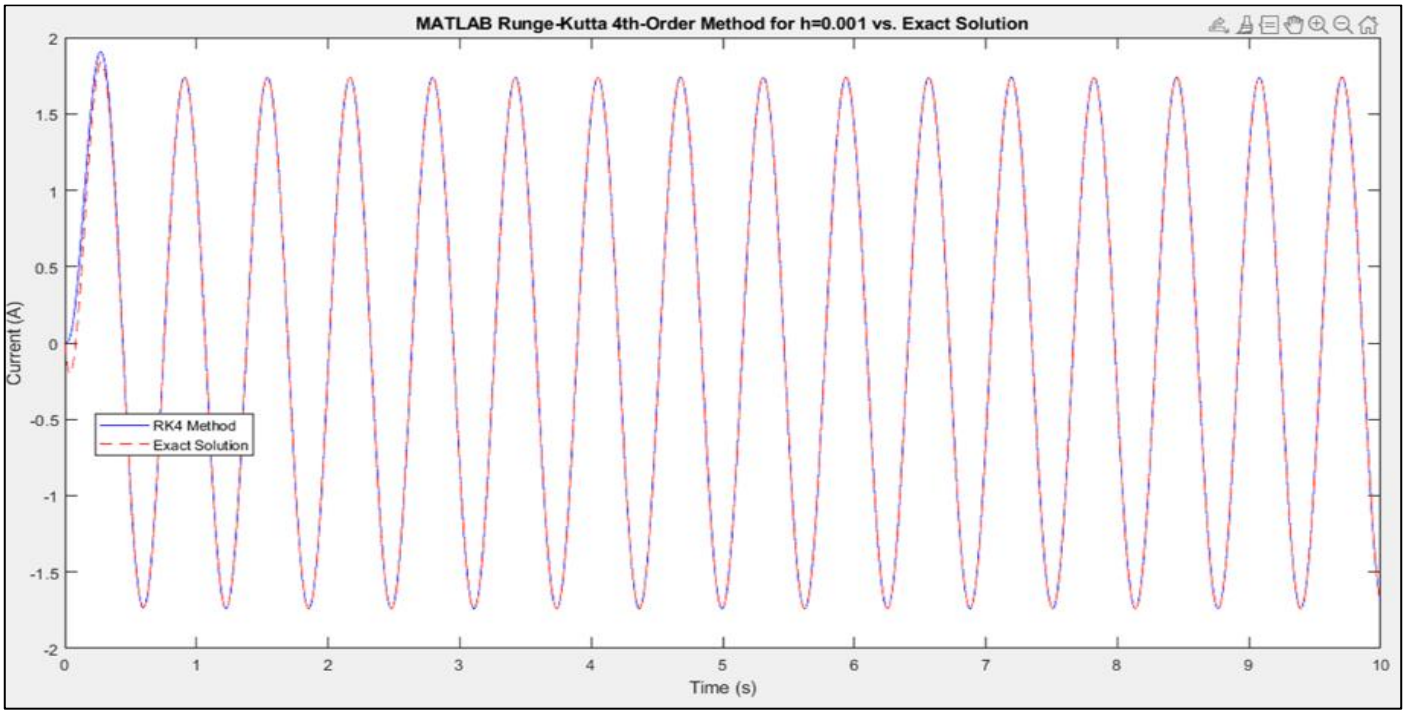


Fig 1 MATLAB Runge-Kutta 4th-order method (RK4) for h = 0.001 vs exact solution

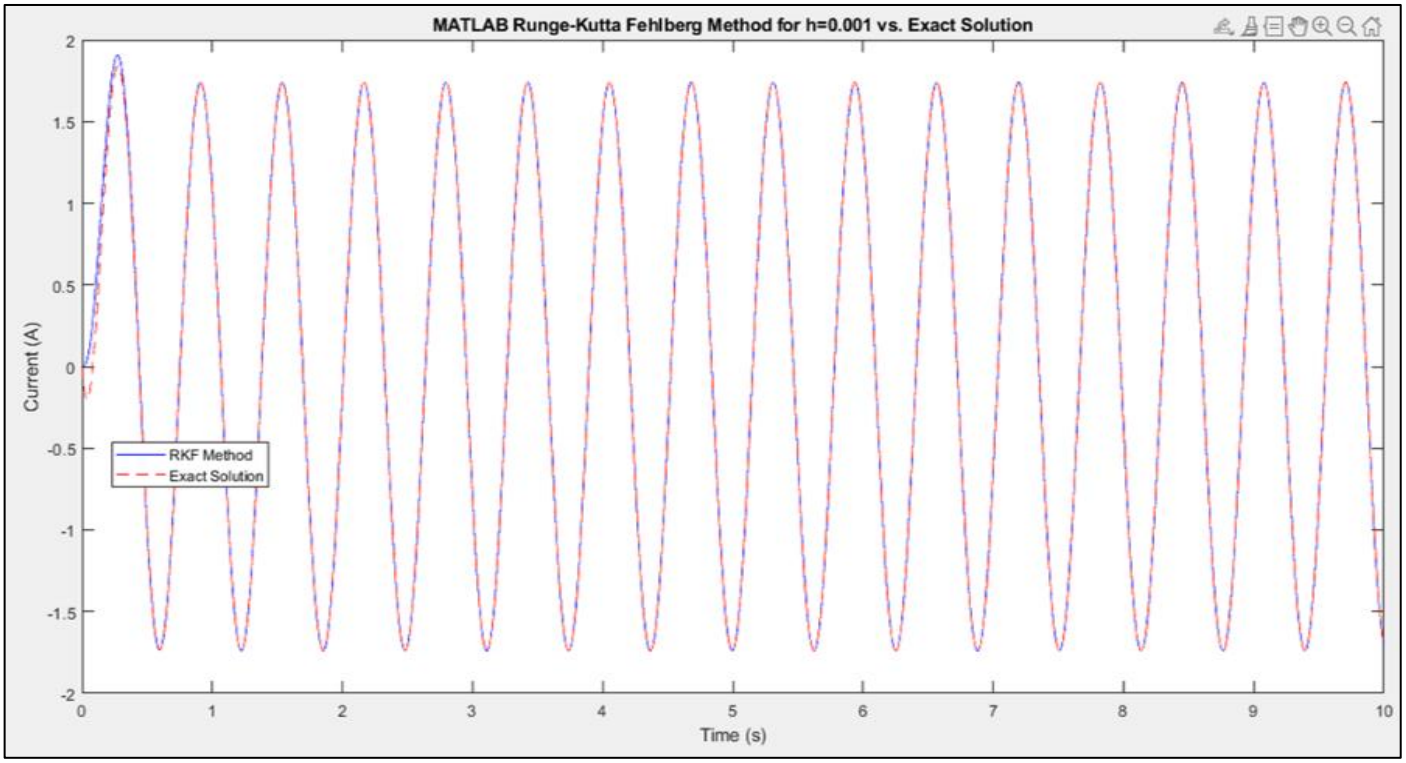


Fig 2 MATLAB Runge-Kutta-Fehlberg method for h = 0.001vs exact solution

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

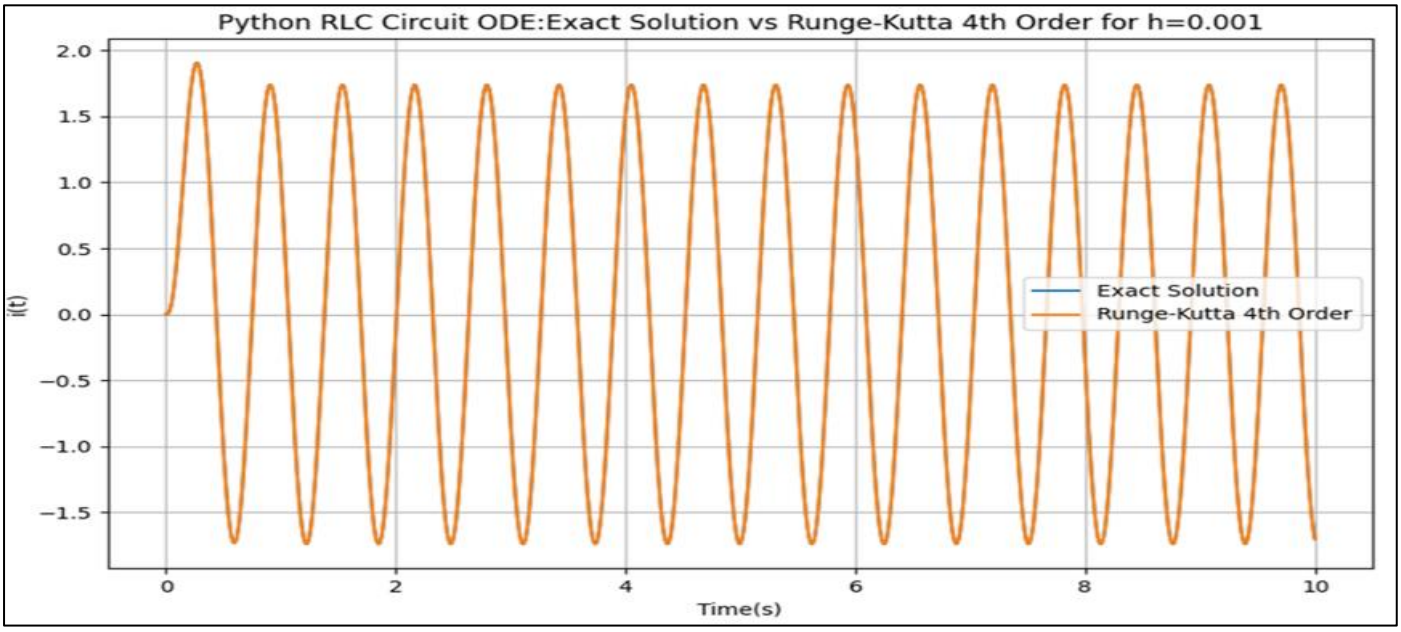


Fig 3 Python Runge-Kutta 4th-order method for h = 0.001vs exact solution

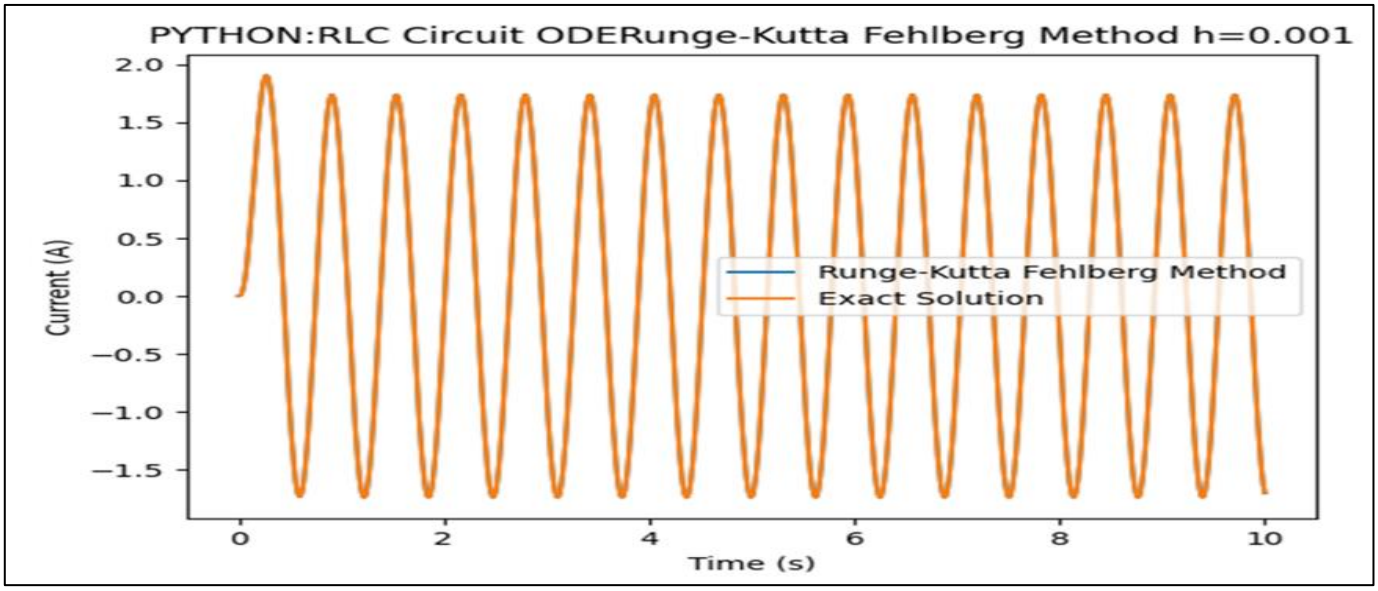


Fig 4 Python Runge-Kutta-Fehlberg method for h = 0.001 vs exact solution

Figures 1 and 2 present MATLAB-generated graphs comparing numerical and exact solutions for an RLC circuit over 10 seconds. Figure 1 shows the Runge-Kutta 4th-order (RK4) method (blue) closely matching the exact solution (red), demonstrating high accuracy and minimal truncation error. Figure 2 compares the Runge-Kutta-Fehlberg (RKF) method (blue) with the exact solution (dashed red), highlighting its phase and amplitude consistency, adaptive step size control, and numerical stability. Figures 3 and 4 display Python-simulated graphs. Figure 3 compares the RK4 method (orange) with the exact solution (blue), showing precise approximation and periodic oscillations. Figure 4 compares the RKF method (blue) with the exact solution (orange), confirming high accuracy and stability across multiple step sizes. In all cases, smaller step sizes improve precision, reinforcing the reliability of both RK4 and RKF methods for simulating RLC circuits.

## IV. CONCLUSION

Following the progression of this research, several key achievements have been realized. The RKF method consistently outperformed the RK4 method in both MATLAB and Python, maintaining lower truncation errors and better convergence. The RK4 method showed stable error growth, improving as step size decreased. The RKF method demonstrated superior numerical accuracy, even for longer simulations, making it more efficient for solving ODEs. The results confirm that smaller step sizes improve accuracy, with RKF offering a more precise and reliable solution than RK4. Both MATLAB and Python offer robust environments for implementing the RK4 and RKF methods. MATLAB's built-in functions make implementation straightforward, while Python's libraries like SciPy, SymPy, Matplotlib, and Pandas provide additional flexibility and ease. In MATLAB, it takes about ten seconds to establish convergence, whereas it takes

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

about two minutes in Python. MATLAB generally offers faster computation for vectorized operations, which is beneficial for large-scale problems, while Python provides comparable performance with the added advantage of better integration with other software and tools.

## REFERENCES


[1]. Agam, S. A., & Yahaya, Y. A. (2014). A highly efficient implicit Runge-Kutta method for first order ordinary differential equations. *African Journal of Mathematics and Computer Science Research, 7*(5), 55-60. https://doi.org/10.5897/AJMCSR2014.0551

[2]. Ahamad, N., & Charan, S. (2019). Study of numerical solution of fourth order ordinary differential equations by fifth order Runge-Kutta method. *International Journal of Scientific Research in Science and Technology (IJSRSET), 6*(1), 230-237. https://doi.org/10.32628/IJSRSET196142

[3]. Afham, M. (2019). *Numerical solution to RLC series circuit with constant and varying source.* https://medium.com/@afhamaflal9/numerical-solution-to-rlc-series-circuit-with-constant-and-varying-source-

[4]. Alizadeh, S., Baleanu, D., & Rezapour, S. (2020). Analyzing transient response of the parallel RCL circuit by using the Caputo–Fabrizio fractional derivative. *Advances in Continuous and Discrete Models*, (55).

[5]. Anthony, A. O., Pius, F., Onyinyechi, F. A., & Ahmed, A. D. (2019). Analysis and comparative study of numerical solutions of initial value problems in ordinary differential equations with Euler and Runge-Kutta methods. *American Journal of Engineering (AJER), 8*(8), 40-53.

[6]. Ascher, U. M., & Greif, C. (2011). *A first course in numerical methods. Society for Industrial and Applied Mathematics* (SIAM), *7*(1). https://doi.org/10.1137/9780898719987

[7]. Ashgi, R., Pratama, M. A. A., & Purwani, S. (2021). Comparison of numerical simulation of epidemiological model between Euler method with 4th order Runge Kutta method. *International Journal of Global Operations Research, 2*(1), 37-44.

[8]. Ascher, U. M., & Petzold, L. R. (1998). *Computer methods for ordinary differential equations and differential-algebraic equations.* Society for Industrial and Applied Mathematics.

[9]. Ascher, U. M., & Greif, C. (2011). *A first course in numerical methods.* SIAM. https://books.google.com.ng/books/about/A_First_Course_in_Numerical_Methods.html?id=eGDMSIqPYdYC&redir_esc=y

[10]. Attaway, D. C. (2022). *MATLAB: A practical introduction to programming and problem solving* (6thed.). Butterworth-Heinemann. https://www.mathworks.com/academia/books/matlab-attaway.html

[11]. Balac, S. (2013). High order embedded Runge-Kutta scheme for adaptive step-size control in the interaction picture method. *Journal of the Korean Society for Industrial and Applied Mathematics (J.KSIAM), 17*(4), 238–266. https://doi.org/10.12941/jksiam.2013.17.238

[12]. Balagopalan, S., & EEE Department. (2014). *Fundamental concepts of electric circuits*. VAST. http://www.vidyaacademy.ac.in

[13]. Bartoldson, B. R., Kailkhura, B., & Blalock, D. (2022). *Compute-efficient deep learning: Algorithmic trends and opportunities.* arXiv. https://arxiv.org/abs/2210.06640

[14]. Banu, M. S., Raju, I., Zaman, U. H. M., & Zaman, M. (2021). A study on numerical solution of initial value problem by using Euler's method, Runge-Kutta 2nd order, Runge-Kutta 4th order, and Runge-Kutta Fehlberg method with MATLAB. *International Journal of Scientific & Engineering Research, 12*(3), 61.

[15]. Bellen, A., & Zennaro, M. (2013). *Numerical methods for delay differential equations.* OUP Oxford. https://books.google.com.ng/books?id=C_CLxpHQbY8C&printsec=frontcover&dq=eitions

[16]. Bhanu Prakash, S., Chandan, K., Karthik, K., Devanathan, S., Varun Kumar, R. S., Nagaraja, K. V., & Prasannakumara, B. C. (2024). Investigation of the thermal analysis of a wavy fin with radiation impact: an application of extreme learning machine. *Physica Scripta, 99*(1), 015225. https://doi.org/10.1088/1402-4896/ad131f

[17]. Bhogendra, K. T., Kafle, J., & Bhandari, I. B. (2021). Visualization, formulation, and intuitive explanation of iterative methods for transient analysis of RLC circuit. *BIBECHANA, 18*(2), 9–17. https://doi.org/10.3126/bibechana.v18i2.31208

[18]. Bocharov, G. A., & Rihan, F. A. (2000). Numerical modelling in biosciences using delay differential equations. *Journal of Computational and Applied Mathematics, 125*(1–2), 183–199. https://doi.org/10.1016/S0377-0427(00)00468-4

[19]. Breit, D., & Dodgson, A. (2021). Convergence rates for the numerical approximation of the 2D stochastic Navier–Stokes equations. *Numerische Mathematik, 147*, 553–578. https://doi.org/10.1007/s00211-021-01181-z

[20]. Burden, R. L., & Faires, J. D. (2015). *Numerical analysis* (10th ed.). Cengage Learning. https://www.amazon.com/Numerical-Analysis-Richard-L-Burden/dp/1305253663

[21]. Butcher, J. C. (2016). *Numerical methods for ordinary differential equations*, (3rd ed.). John Wiley & Sons.

[22]. Clayton, S. L., Lemma, M., & Chowdhury, A. (2019). Numerical solutions of nonlinear ordinary differential equations by using adaptive Runge-Kutta method. *Journal of Advances in Mathematics*, 17, 147-154. https://doi.org/10.24297/jam.v17i0.8408

[23]. Mondal, S. Banu, M. S., & Raju, I. (2016). A comparative study on classical fourth order and Butcher sixth order Runge-Kutta methods with initial and boundary value problems. International Journal of Material and Mathematical Sciences, 5(3), 45-57. DOI: https://doi.org/10.34104/ijmms.021.08021.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

**APPENDIX**

TABLE 1.2: MATLAB RLC Circuit Comparative Result Table for Runge-Kutta 4th Order Method for h = 0.0005

| # | Time | Runge-Kutta 4th Order Method i(t) | Exact Solution i(t) | Local Truncation Error |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0.00E+00 |
| **1** | | | | |
| **2** | 0.0005 | 0.00E+00 | -0.00546 | 0.00546 |
| **3** | 0.001 | 0.00E+00 | -0.01084 | 0.01084 |
| **4** | 0.0015 | 0.00E+00 | -0.01613 | 0.01614 |
| **5** | 0.002 | 1.00E-05 | -0.02134 | 0.02136 |
| **6** | 0.0025 | 1.00E-05 | -0.02647 | 0.0265 |
| **7** | 0.003 | 2.00E-05 | -0.03152 | 0.03156 |
| **8** | 0.0035 | 4.00E-05 | -0.03649 | 0.03655 |
| **9** | 0.004 | 6.00E-05 | -0.04138 | 0.04146 |
| **10** | 0.0045 | 8.00E-05 | -0.04619 | 0.0463 |
| **11** | 0.005 | 0.00011 | -0.05092 | 0.05107 |
| **12** | 0.0055 | 0.00015 | -0.05557 | 0.05576 |
| **13** | 0.006 | 0.00019 | -0.06014 | 0.06038 |
| **14** | 0.0065 | 0.00024 | -0.06463 | 0.06493 |
| **15** | 0.007 | 0.0003 | -0.06904 | 0.06941 |
| **16** | 0.0075 | 0.00037 | -0.07337 | 0.07382 |
| **17** | 0.008 | 0.00044 | -0.07763 | 0.07816 |
| **18** | 0.0085 | 0.00053 | -0.0818 | 0.08243 |
| **19** | 0.009 | 0.00062 | -0.0859 | 0.08663 |
| **20** | 0.0095 | 0.00073 | -0.08992 | 0.09077 |
| **21** | 0.01 | 0.00085 | -0.09387 | 0.09485 |
| **22** | 0.0105 | 0.00098 | -0.09773 | 0.09886 |
| **23** | 0.011 | 0.00112 | -0.10152 | 0.1028 |
| **24** | 0.0115 | 0.00128 | -0.10524 | 0.10668 |
| **25** | 0.012 | 0.00145 | -0.10887 | 0.1105 |
| **26** | 0.0125 | 0.00163 | -0.11244 | 0.11426 |
| **27** | 0.013 | 0.00183 | -0.11592 | 0.11796 |
| **28** | 0.0135 | 0.00204 | -0.11933 | 0.1216 |
| **29** | 0.014 | 0.00227 | -0.12267 | 0.12517 |
| **30** | 0.0145 | 0.00251 | -0.12593 | 0.12869 |
| **31** | 0.015 | 0.00277 | -0.12911 | 0.13215 |
| **32** | 0.0155 | 0.00304 | -0.13222 | 0.13556 |
| **33** | 0.016 | 0.00333 | -0.13526 | 0.1389 |
| **34** | 0.0165 | 0.00364 | -0.13823 | 0.14219 |

Table 1.2 shows the computed results of the Runge-Kutta 4th order method in MATLAB with the step size h = 0.0005. The table compares the currents i(t) generated using the Runge-Kutta 4th order method and the exact solution.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

**TABLE 1.4 MATLAB RLC Circuit Comparative Result Table for Runge-Kutta 4th Order Method for h = 0.000125**

| # | Time | Runge-Kutta 4th Order Method i(t) | Exact Solution i(t) | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0.00E+00 |
| 2 | 0.00013 | 0.00E+00 | -0.00137 | 0.00137 |
| 3 | 0.00025 | 0.00E+00 | -0.00274 | 0.00274 |
| 4 | 0.00038 | 0.00E+00 | -0.0041 | 0.0041 |
| 5 | 0.0005 | 0.00E+00 | -0.00546 | 0.00546 |
| 6 | 0.00063 | 0.00E+00 | -0.00681 | 0.00681 |
| 7 | 0.00075 | 0.00E+00 | -0.00816 | 0.00816 |
| 8 | 0.00088 | 0.00E+00 | -0.0095 | 0.0095 |
| 9 | 0.001 | 0.00E+00 | -0.01084 | 0.01084 |
| 10 | 0.00113 | 0.00E+00 | -0.01217 | 0.01217 |
| 11 | 0.00125 | 0 | -0.01349 | 0.0135 |
| 12 | 0.00138 | 0 | -0.01481 | 0.01482 |
| 13 | 0.0015 | 0 | -0.01613 | 0.01613 |
| 14 | 0.00163 | 0 | -0.01744 | 0.01745 |
| 15 | 0.00175 | 0 | -0.01875 | 0.01875 |
| 16 | 0.00188 | 1.00E-05 | -0.02005 | 0.02005 |
| 17 | 0.002 | 1.00E-05 | -0.02134 | 0.02135 |
| 18 | 0.00213 | 1.00E-05 | -0.02263 | 0.02264 |
| 19 | 0.00225 | 1.00E-05 | -0.02392 | 0.02393 |
| 20 | 0.00238 | 1.00E-05 | -0.0252 | 0.02521 |
| 21 | 0.0025 | 1.00E-05 | -0.02647 | 0.02649 |
| 22 | 0.00263 | 2.00E-05 | -0.02774 | 0.02776 |
| 23 | 0.00275 | 2.00E-05 | -0.02901 | 0.02903 |
| 24 | 0.00288 | 2.00E-05 | -0.03027 | 0.03029 |
| 25 | 0.003 | 2.00E-05 | -0.03152 | 0.03155 |
| 26 | 0.00313 | 3.00E-05 | -0.03277 | 0.03281 |
| 27 | 0.00325 | 3.00E-05 | -0.03402 | 0.03405 |
| 28 | 0.00338 | 3.00E-05 | -0.03526 | 0.0353 |
| 29 | 0.0035 | 4.00E-05 | -0.03649 | 0.03654 |
| 30 | 0.00363 | 4.00E-05 | -0.03772 | 0.03777 |
| 31 | 0.00375 | 5.00E-05 | -0.03895 | 0.039 |
| 32 | 0.00388 | 5.00E-05 | -0.04017 | 0.04023 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

| | | | | |
|---|---|---|---|---|
| 79965 | 9.99513 | -1.6817 | -1.6817 | 0.00055 |
| 79966 | 9.99525 | -1.68226 | -1.68226 | 0.00055 |
| 79967 | 9.99538 | -1.68281 | -1.68281 | 0.00055 |
| 79968 | 9.9955 | -1.68336 | -1.68336 | 0.00055 |
| 79969 | 9.99563 | -1.6839 | -1.6839 | 0.00054 |
| 79970 | 9.99575 | -1.68444 | -1.68444 | 0.00054 |
| 79971 | 9.99588 | -1.68498 | -1.68498 | 0.00054 |
| 79972 | 9.996 | -1.68552 | -1.68552 | 0.00053 |
| 79973 | 9.99613 | -1.68606 | -1.68606 | 0.00053 |
| 79974 | 9.99625 | -1.68659 | -1.68659 | 0.00053 |
| 79975 | 9.99638 | -1.68712 | -1.68712 | 0.00053 |
| 79976 | 9.9965 | -1.68764 | -1.68764 | 0.00052 |
| 79977 | 9.99663 | -1.68817 | -1.68817 | 0.00052 |
| 79978 | 9.99675 | -1.68869 | -1.68869 | 0.00052 |
| 79979 | 9.99688 | -1.68921 | -1.68921 | 0.00052 |
| 79980 | 9.997 | -1.68973 | -1.68973 | 0.00051 |
| 79981 | 9.99713 | -1.69024 | -1.69024 | 0.00051 |
| 79982 | 9.99725 | -1.69075 | -1.69075 | 0.00051 |
| 79983 | 9.99738 | -1.69126 | -1.69126 | 0.00051 |
| 79984 | 9.9975 | -1.69177 | -1.69177 | 0.0005 |
| 79985 | 9.99763 | -1.69227 | -1.69227 | 0.0005 |
| 79986 | 9.99775 | -1.69277 | -1.69277 | 0.0005 |
| 79987 | 9.99788 | -1.69327 | -1.69327 | 0.0005 |
| 79988 | 9.998 | -1.69376 | -1.69376 | 0.00049 |
| 79989 | 9.99813 | -1.69426 | -1.69426 | 0.00049 |
| 79990 | 9.99825 | -1.69475 | -1.69475 | 0.00049 |
| 79991 | 9.99838 | -1.69523 | -1.69523 | 0.00048 |
| 79992 | 9.9985 | -1.69572 | -1.69572 | 0.00048 |
| 79993 | 9.99863 | -1.6962 | -1.6962 | 0.00048 |
| 79994 | 9.99875 | -1.69668 | -1.69668 | 0.00048 |
| 79995 | 9.99888 | -1.69716 | -1.69716 | 0.00047 |
| 79996 | 9.999 | -1.69763 | -1.69763 | 0.00047 |
| 79997 | 9.99913 | -1.6981 | -1.6981 | 0.00047 |
| 79998 | 9.99925 | -1.69857 | -1.69857 | 0.00047 |
| 79999 | 9.99938 | -1.69904 | -1.69904 | 0.00046 |
| 80000 | 9.9995 | -1.6995 | -1.6995 | 0.00046 |
| 80001 | 9.99963 | -1.69996 | -1.69996 | 0.00046 |
| 80002 | 9.99975 | -1.70042 | -1.70042 | 0.00046 |
| 80003 | 9.99988 | -1.70087 | -1.70087 | 0.00045 |
| 80004 | 10 | -1.70133 | -1.70133 | 0.00045 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

**TABLE 4.4 PYTHON RLC Circuit Comparative Result Table for Runge-Kutta Fehlberg Method for h = 0.000125**

| # | Time | Exact Solution i(t) | Runge Kutta Fehlberg Method i(t) | Local Truncation Error |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0.00013 | 0 | 0 | 0 |
| 3 | 0.00025 | 0 | 0 | 0 |
| 4 | 0.00038 | 0 | 0 | 0 |
| 5 | 0.0005 | 0 | 0 | 0 |
| 6 | 0.00063 | 0 | 0 | 0 |
| 7 | 0.00075 | 0 | 0 | 0 |
| 8 | 0.00088 | 0 | 0 | 0 |
| 9 | 0.001 | 0 | 0 | 0 |
| 10 | 0.00113 | 0 | 0 | 0 |
| 11 | 0.00125 | 0 | 0 | 0 |
| 12 | 0.00138 | 0 | 0 | 0 |
| 13 | 0.0015 | 0 | 0 | 0 |
| 14 | 0.00163 | 0 | 0 | 0 |
| 15 | 0.00175 | 0 | 0 | 0 |
| 16 | 0.00188 | 1.00E-05 | 0 | 1.00E-05 |
| 17 | 0.002 | 1.00E-05 | 0 | 1.00E-05 |
| 18 | 0.00213 | 1.00E-05 | 0 | 1.00E-05 |
| 19 | 0.00225 | 1.00E-05 | 0 | 1.00E-05 |
| 20 | 0.00238 | 1.00E-05 | 0 | 1.00E-05 |
| 21 | 0.0025 | 1.00E-05 | 0 | 1.00E-05 |
| 22 | 0.00263 | 2.00E-05 | 0 | 2.00E-05 |
| 23 | 0.00275 | 2.00E-05 | 0 | 2.00E-05 |
| 24 | 0.00288 | 2.00E-05 | 0 | 2.00E-05 |
| 25 | 0.003 | 2.00E-05 | 0 | 2.00E-05 |
| 26 | 0.00313 | 3.00E-05 | 0 | 3.00E-05 |
| 27 | 0.00325 | 3.00E-05 | 0 | 3.00E-05 |
| 28 | 0.00338 | 3.00E-05 | 0 | 3.00E-05 |
| 29 | 0.0035 | 4.00E-05 | 0 | 4.00E-05 |
| 30 | 0.00363 | 4.00E-05 | 0 | 4.00E-05 |
| 31 | 0.00375 | 5.00E-05 | 0 | 5.00E-05 |
| 32 | 0.00388 | 5.00E-05 | 1.00E-05 | 4.00E-05 |
| 33 | 0.004 | 6.00E-05 | 2.00E-05 | 4.00E-05 |

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

**PYTHON SCRIP**

```
import numpy as np
import csv
import matplotlib.pyplot as plt

# Define the differential equation parameters
R = 12
L = 0.4
C = 0.0125
# Define the exact solution function
def exact_solution(t):
    return 2.75 * np.exp(-10 * t) - 1.1 * np.exp(-20 * t) + 0.55 * np.sin(10
* t) - 1.65 * np.cos(10 * t)
# Define the RHS of the differential equation
def f(t, i, ip):
    return 550 * np.sin(10 * t) - (R / L) * ip - (1 / (L * C)) * i
# Runge-Kutta 4th order method
def runge_kutta4(t, i, ip, h):
    k1 = h * ip
    l1 = h * f(t, i, ip)
    k2 = h * (ip + l1 / 2)
    l2 = h * f(t + h / 2, i + k1 / 2, ip + l1 / 2)
    k3 = h * (ip + l2 / 2)
    l3 = h * f(t + h / 2, i + k2 / 2, ip + l2 / 2)
    k4 = h * (ip + l3)
    l4 = h * f(t + h, i + k3, ip + l3)
    i_next = i + (k1 + 2 * k2 + 2 * k3 + k4) / 6
    ip_next = ip + (l1 + 2 * l2 + 2 * l3 + l4) / 6
    return i_next, ip_next
# Compute local truncation error (LTE)
def local_truncation_error(t, i_exact, i_approx):
    return np.abs(i_exact - i_approx)
# Define parameters
t0 = 0
i0 = 0
ip0 = 0
h = 0.001
t_end = 10

# Initialize lists to store results
results = []
# Perform iterations and compute LTE
t = t0
i = i0
ip = ip0
while t <= t_end:
    i_exact = exact_solution(t)
    i, ip = runge_kutta4(t, i, ip, h)
    lte = local_truncation_error(t, i_exact, i)
    results.append([round(t, 5), round(i_exact, 5), round(i, 5), round(lte,
5)])
    t += h
# Write results to CSV
with open('lte_results.csv', 'w', newline='') as csvfile:
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar599

**MATLAB SCRIPT**

```
% Given parameters
R = 12; % Ohms
L = 0.4; % Henries
C = 0.0125; % Farads

% Define the ODE as a function
f = @(t, i) [i(2); (550*sin(10*t) - R/L*i(2) - 1/(L*C)*i(1))];

% Exact solution
exact_solution = @(t) 2.75*exp(-10*t) - 1.1*exp(-10*t) + 0.55*sin(10*t) -
1.65*cos(10*t);

% Fourth-order Runge-Kutta method
h = 0.001; % Step size
tspan = 0:h:10; % Time span
y0 = [0; 0]; % Initial conditions

% Initialize arrays to store results
t_values = zeros(length(tspan), 1);
i_values = zeros(length(tspan), 1);
exact_values = zeros(length(tspan), 1);
LTE_values = zeros(length(tspan), 1);

% Run fourth-order Runge-Kutta method
for i = 1:length(tspan)
    t = tspan(i);
    t_values(i) = t;
    i_values(i) = y0(1);
    exact_values(i) = exact_solution(t);

    % Runge-Kutta method
    k1 = h * f(t, y0);
    k2 = h * f(t + 0.5*h, y0 + 0.5*k1);
    k3 = h * f(t + 0.5*h, y0 + 0.5*k2);
    k4 = h * f(t + h, y0 + k3);

    y1 = y0 + (k1 + 2*k2 + 2*k3 + k4) / 6; % Update using weighted average of
slopes

    LTE_values(i) = abs(exact_values(i) - y1(1)); % Calculate LTE (compare
only i(t) values)

    y0 = y1; % Update y0 for the next iteration
end

% Round off the table of values to five decimal places
t_values = round(t_values, 5);
i_values = round(i_values, 5);
exact_values = round(exact_values, 5);
LTE_values = round(LTE_values, 5);

% Display results in a table
```