Project 1: Algorithms to numerically simulate RLC-series circuits and electric fields

Jada Garofalo
Physics Department at Syracuse University, Syracuse, NY 13210

September 30, 2025

1 Introduction

1.1 Motivation

When dealing with physical systems, it is often unreasonable to assume one would be able to spend hours to days to years with a pen and paper, tabulating how they evolve over a set parameter. To avoid this, we let computers do these calculations at speeds that would otherwise be unfathomable, saving time and giving us valuable insights into the systems we wish to simulate.

This semester, I am a teaching assistant for PHY212, the introductory electromagnetism course for engineering students. This class has a curriculum containing quite a few interesting situations that may be valuable to simulate via numerical methods. The first that came to mind is the ordinary differential equation (ODE) that describes current as a function of time for an RLC-series circuit (that is, a circuit in which one resistor, inductor, and capacitor are in series). The second order ODE that describes this behavior follows

$$\ddot{I} + 2\alpha\dot{I} + \omega_0^2 I(t) = 0 \tag{1}$$

where $\alpha = \frac{R}{2L}$ and $\omega_0 = \frac{1}{\sqrt{LC}}$. Like many other interesting physical phenomena, this is an oscillator! By adjusting α, we can damp the oscillator, and one could even add a driving force, just as was first taught with spring systems. Undamped, this takes the analytic form

$$I(t) = \frac{\dot{I}_0}{\omega_0} \sin \omega_0 t \tag{2}$$

The next scenario that came to mind is that of an electric field due to a linear charge distribution. To simplify the problem, I decided to limit the degrees of freedom such that I only calculate the electric field at various points along the x-axis for a linear charge distribution that also lies on the x-axis. This takes the form

$$E(x_{test}) = \frac{\lambda}{4\pi\epsilon_0} \int_0^l \frac{x_{test} - x'}{\|x_{test} - x'\|^3} dx' \tag{3}$$

with λ being the linear charge density, l being the length of the distribution (defined to be from $x = 0$ to $x = l$), ϵ_0 being the vacuum permittivity, and x_{test} being the point at which we wish to find the electric field of.

This can then be solved numerically as a discrete summation of areas, with widths of dx', following the form of the classic Riemann sum. This integral has the analytic solution

$$E(x_{test}) = \frac{\lambda}{4\pi\epsilon_0}\left(\frac{1}{x_{test} - l} - \frac{1}{x_{test}}\right) \tag{4}$$

2 Methods

2.1 ODE Algorithms

To solve the ODE for the RLC-series circuit problem, I used four stepwise methods, and for error comparison one method from SciPy, namely RK45[1].

The stepwise methods I implemented included the Euler-explicit, Symplectic, RK2, and RK4 methods we discussed in class (with RK4 being the exception here, but this essentially operates as a fourth-order version of RK2 with weighted constants determining the contribution of higher levels of approximations). Euler-explicit is expected to scale at first-order, Symplectic and RK2 are expected to scale at second-order, and RK4 is expected to scale at fourth-order. SciPy's RK45 does not scale with stepsize, and is expected to be constant, performing slightly better than RK4 (as per the documentation, RK45 follows RK4, but with steps being taken at fifth-order accuracy).

2.2 Integration Algorithms

To solve the integral for the linear charge distribution problem, I used four stepwise methods, and for error comparison one method from SciPy, namely the QUAD[2] integration method.

The stepwise methods I implemented included the Left-hand Riemann, Midpoint Riemann, Trapezoid Riemann, and Simpson's rule methods we discussed in class. Left-hand Riemann is expected to scale at first-order, Midpoint Riemann and Trapezoid Riemann are expected to scale at second-order, and Simpson's rule is expected to scale at fourth-order. SciPy's QUAD does not scale with stepsize, and is expected to be constant. The documentation did not state how comparatively accurate I should expect for this to operate at, but did mention the technique used comes from Fortran's QUADPACK library.

2.3 Expectations and Analysis

For the RLC-series circuit, I expected to have presentable plots for both underdamped and overdamped cases. The underdamped case should exhibit sinusoidal motion, decaying over time. The overdamped case should exponentially fall off to zero.

I also expected that the Euler-explicit method will perform the worst, the RK2 and Symplectic methods would be better and comparable to one another, and RK4 would perform best.

For the linear charge distribution, I expected the electric field to fall off like $1 \,/\, x'$ outside of the charge distribution, and for it to sharply spike upwards near the ends of the distribution. I also expected that Left-hand Riemann summation would perform the worst, Midpoint and Trapezoid Riemann summation would preform better and be comparable to one another, and Simpson's rule summation would perform best.

To compare methods and verify efficacy of the above algorithms, I employed a simple root mean square error (RMSE) calculation for various stepsizes for each method. As mentioned above, SciPy methods were also included to see how these wide-release methods compare to my own.

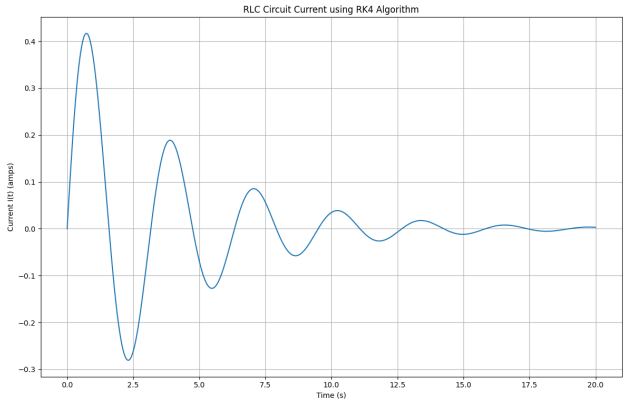
3 Results

Method	$\Delta t = 10^{-1}$	$\Delta t = 10^{-2}$	$\Delta t = 10^{-3}$	$\Delta t = 10^{-4}$
Euler-explicit	2.28×10^{-1}	1.92×10^{-2}	1.89×10^{-3}	1.89×10^{-4}
Symplectic	2.00×10^{-3}	1.99×10^{-5}	1.99×10^{-7}	1.99×10^{-9}
RK2	7.29×10^{-3}	7.27×10^{-5}	7.27×10^{-7}	7.27×10^{-9}
RK4	3.64×10^{-6}	3.64×10^{-10}	3.77×10^{-14}	2.40×10^{-15}

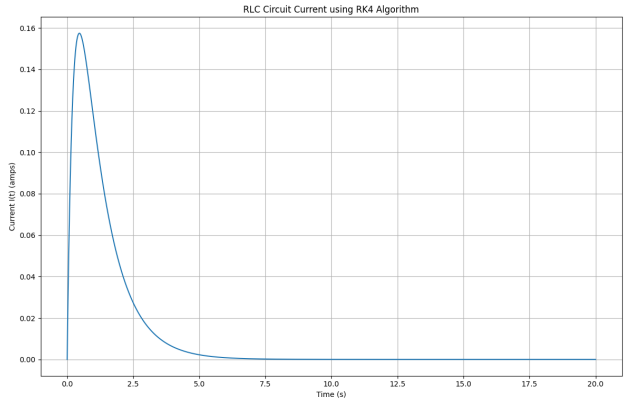
Table 1: RLC-series circuit RMSE per stepsize (Δt), parameters from Figure 2

Method	$\Delta x = 10^{-1}$	$\Delta x = 10^{-2}$	$\Delta x = 10^{-3}$	$\Delta x = 10^{-4}$
Left-hand Riemann	8.06×10^{1}	8.06	8.06×10^{-1}	8.06×10^{-2}
Midpoint	1.32	1.33×10^{-2}	1.33×10^{-4}	1.33×10^{-6}
Trapezoid	2.65	2.66×10^{-2}	2.66×10^{-4}	2.66×10^{-6}
Simpson's rule	1.18×10^{-3}	1.19×10^{-7}	1.21×10^{-11}	5.18×10^{-12}

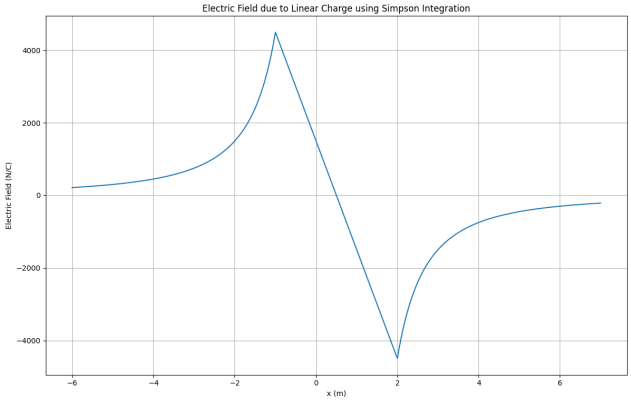
Table 2: Linear charge distribution RMSE per stepsize (Δx), parameters from Figure 2



(a) Evolution of current over time for an underdamped RLC-circuit, using the RK4 method and $\Delta t = 0.01$ (specifically, $L = 1$, $R = 0.5$, $C = 0.25$, $\dot{I}_0 = 1$)

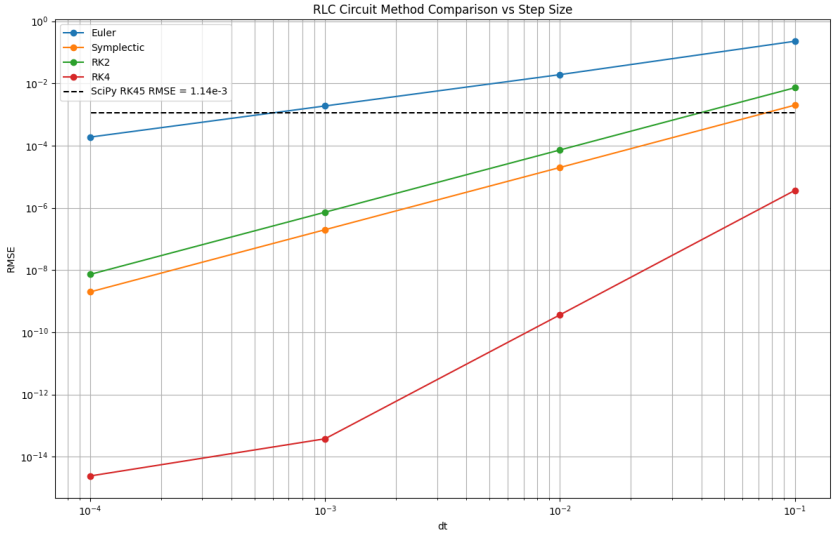


(b) Evolution of current over time for an overdamped RLC-circuit, using the RK4 method and $\Delta t = 0.01$ (specifically, $L = 1$, $R = 5$, $C = 0.25$, $\dot{I}_0 = 1$)

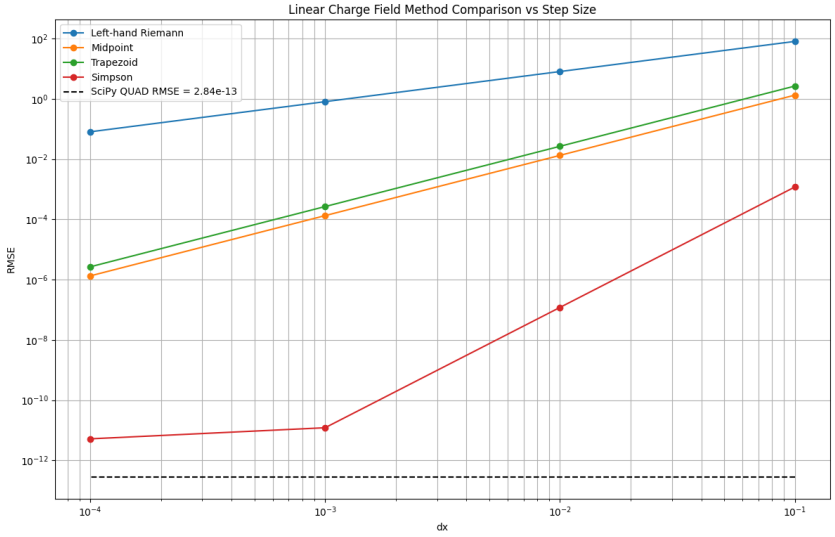


(c) Electric field as a function of test position using Simpson's rule ($\lambda = 10^{-6}$, $l = 1$, evaluated about `np.linspace(-6.0, -1.0, 100)` and `np.linspace(2.0, 7.0, 100)`)

Figure 1



(a) Evolution of RMSE over stepsize for undamped RLC-series circuit methods, along with SciPy RK45 method for comparison ($L = 1$, $C = 1$, $R = 0$, $\dot{I}_0 = 1$)



(b) Evolution of RMSE over stepsize for linear charge distribution methods, along with SciPy QUAD method for comparison ($\lambda = 10^{-6}$, $l = 1$, evaluated about `np.linspace(-6.0, -1.0, 100)` and `np.linspace(2.0, 7.0, 100)`)

Figure 2

4 Discussion

Generally, I found the results to be in agreement with my expectations, barring a few key surprises. As shown in Fig. 1(a) and Fig. 1(b), the overdamped and underdamped cases for the RLC-series circuit behave exactly as expected, namely the underdamped case oscillates, with smaller and smaller amplitudes tending towards zero over time, and the overdamped case exponentially dips to zero. These cases indicate that the algorithms are respecting the input parameters of the physical system, and showcase fundamental principles of oscillatory motion. When comparing their RMSE over varying stepsizes (see Fig. 2(a)), the algorithms exhibit their expected orders, namely Euler-explicit acts at first-order, Symplectic and RK2 act at second-order, and RK4 acts at fourth-order. Interestingly, SciPy's RK45 initially gives off results similar to that of a second-order algorithm, but as the stepsizes dwindle it quickly gets passed by even the Euler-explicit method. Based on SciPy's documentation, I would expect for this function to be making a much stronger estimate than what it did here. This is perhaps something that should be further examined in case there was an error in implementation.

One primary issue with this project that took a decent amount of time to reconcile was plotting the linear charge distribution plot featured in Fig. 1(c). Due to the rapid increase in electric field, both the analytic and numeric solutions fail rather spectacularly if allowed to run right up to and through the interior region of the charge distribution. To tackle this truncation error, I instead focused on evaluation points well outside of these boundary conditions. This made a plot that readily shows the sharp increase as the evaluation points approach either end of the distribution, while also showing the expected tendency to fall off like $1 \,/\, x'$. These two factors indicate the integration techniques are working as expected, with respect to the imposed boundary conditions. I would have liked to get the plot to not include the center line connecting the two regions of evaluation, but budgeted my time elsewhere. When comparing their RMSE over varying stepsizes (see Fig. 2(b)), the integration methods exhibit their expected orders, namely Left-hand Riemann acts at first-order, Midpoint Riemann and Trapezoid Riemann act at second-order, and Simpson's rule acts at fourth-order. Here, SciPy's QUAD method actually manages to beat out all of my methods across the tested stepsizes. This makes me curious as to what exact technique from Fortran's QUADPACK library is being employed here to get such good results, and I will definitely look into this more.

5 Conclusion

Through Project 1, I got to explore different algorithms to solve ODEs and integrals in physical systems. Constructing these from the ground up let me get more hands-on experience with programming discrete problems, and with learning how developing more complex approaches can lead to vastly more accurate results. This project also got me more familiar with working with multiple files spanning the same directory using Python imports, which makes modular tasks a lot easier and saves a lot of time from reusing the same code in multiple files. I think the biggest takeaway from this project for me was mostly reinforcing good coding techniques, and introducing new ones when necessary. It also allowed for me to ultimately gain a greater appreciation for these systems which I have taught/will be teaching this semester as a teaching assistant.

6 References

[1] SciPy RK45 method – https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.RK45.html
[2] SciPy QUAD integration method – https://docs.scipy.org/doc/scipy-1.7.0/reference/reference/generated/scipy.integrate.quad.html