Numerical Simulation and Energy Analysis of an Ideal 2-Mass 3-Spring System

Amelia Abruscato
PHY-607 Project 1

September 30, 2025

Abstract

In this project, we study the motion of a two-mass three-spring system that does not experience any damping. We solve the equations of motion using numerical methods (Euler and Runge–Kutta 4) and compare those results to the exact analytic solution (found from the system's eigenvalues and eigenvectors) and to SciPy's solvers. We compute kinetic, potential, and total energy and check how well the energy is conserved. We also compute integrals of the energy over time using a Riemann sum, the trapezoidal rule, and Simpson's rule, and compare those integrals to SciPy's functions.

1 Introduction

This project studies the motion of a simple coupled oscillator made of two masses connected by three springs. The setup leads to a system of coupled second-order differential equations that describe the positions of the two masses over time. Such systems are important in physics because they show how normal modes and collective motion arise when objects are connected together.

To understand the behavior of the system, we first look at the analytic solution. By forming the combinations $x_1 + x_2$ and $x_1 - x_2$, the coupled equations can be separated into two independent equations. These represent the normal modes of the system, each with its own frequency. The general solution is then a linear combination of these two modes, which gives the exact motion of the masses.

The main goal of the project is to compare this analytic solution with different numerical methods. We implemented Euler's method and the fourth-order Runge–Kutta (RK4) method by hand, and also used SciPy's built-in ODE solver. This allows us to see how well the numerical solutions match the exact analytic one, and to test how errors build up over time.

In addition to solving the ODEs, the project also looks at the total energy of the system. The kinetic and potential energies can be calculated from the positions and velocities of the masses. We then integrate the total energy over time using three numerical methods: the Riemann sum, the trapezoidal rule, and Simpson's rule. Each of these is compared to SciPy's built-in integration functions for validation.

Finally, the accuracy of our methods is tested against two benchmarks: (1) agreement with the analytic solution, and (2) conservation of energy in the system. By running these checks, we can clearly see where the homemade methods work well, where they fail, and how their errors scale with the step size.

2 Methods

2.1 Mathematical Setup

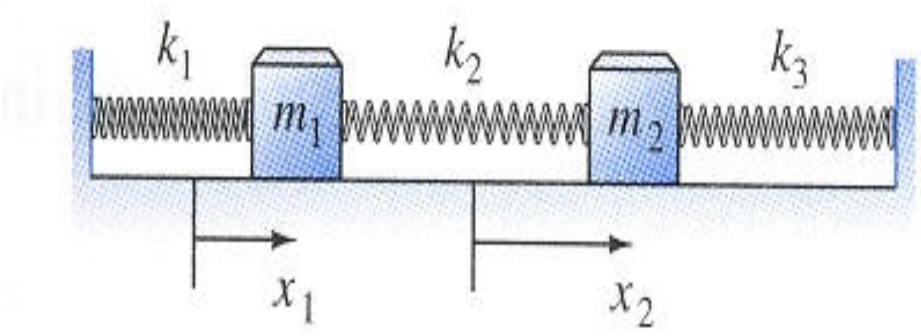

Figure 1: Schematic of the two-mass three-spring system. Mass m_1 is attached to a fixed wall by spring k_1 and connected to mass m_2 by spring k_2. Mass m_2 is connected to the opposite wall by spring k_3. The displacements of the masses from equilibrium are denoted $x_1(t)$ and $x_2(t)$.

The physical system consists of two masses connected in series by three springs depicted in Figure 1. Let $x_1(t)$ and $x_2(t)$ represent the displacements of masses m_1 and m_2 from equilibrium. The governing equations of motion are

$$m_1\ddot{x}_1 = -k_1x_1 + k_2(x_2 - x_1), \tag{1}$$
$$m_2\ddot{x}_2 = -k_3x_2 - k_2(x_2 - x_1). \tag{2}$$

This system can be expressed as a set of four coupled first-order differential equations by letting $v_1 = \dot{x}_1$ and $v_2 = \dot{x}_2$:

$$\dot{x}_1 = v_1, \tag{3}$$
$$\dot{x}_2 = v_2, \tag{4}$$
$$\dot{v}_1 = \frac{-k_1x_1 + k_2(x_2 - x_1)}{m_1}, \tag{5}$$
$$\dot{v}_2 = \frac{-k_3x_2 - k_2(x_2 - x_1)}{m_2}. \tag{6}$$

The analytic solution can be found by transforming to the normal coordinates x_1+x_2 and $x_1 - x_2$. This diagonalizes the system into independent harmonic oscillators, giving normal

mode frequencies that can be solved exactly. This analytic solution serves as a baseline to check the accuracy of the numerical solvers.

2.2 Numerical ODE Solvers

Several numerical methods were implemented to evolve the system in time:

- **Euler method**: A first-order method with global truncation error $O(\Delta t)$. It is simple but accumulates error quickly.
- **Runge-Kutta 4th order (RK4)**: A widely used fourth-order method with global error $O(\Delta t^4)$. More accurate and stable for the same time step compared to Euler.
- **SciPy RK45**: A built-in adaptive method from the SciPy library that automatically adjusts the step size to control error.

Each solver takes as input the system of first-order equations, a time step Δt, and the initial conditions (x_1, x_2, v_1, v_2).

2.3 Energy Computation and Integration

To evaluate the physical accuracy of the solutions, kinetic and potential energies were computed:

$$KE = \tfrac{1}{2}m_1 v_1^2 + \tfrac{1}{2}m_2 v_2^2, \tag{7}$$

$$PE = \tfrac{1}{2}k_1 x_1^2 + \tfrac{1}{2}k_2(x_2 - x_1)^2 + \tfrac{1}{2}k_3 x_2^2, \tag{8}$$

$$E_{\text{total}} = KE + PE. \tag{9}$$

To further analyze accuracy, the energy was integrated over time using different quadrature methods:

- **Riemann sum**: first-order accuracy $O(\Delta t)$.
- **Trapezoidal rule**: second-order accuracy $O(\Delta t^2)$.
- **Simpson's rule**: fourth-order accuracy $O(\Delta t^4)$.

Results were compared to SciPy's built-in `trapz` and `simps` functions for verification.

2.4 Validation

To check the correctness of our implementations, we validated the numerical methods against two independent benchmarks:

- **Analytic solution:** The exact solution was derived using the normal mode method. This provides the true displacements $x_1(t)$ and $x_2(t)$ to compare against. Numerical solutions were plotted alongside the analytic curves, and differences in amplitude and phase were used to estimate accuracy.

- **Physical conservation law:** In the absence of damping, the total mechanical energy of the system should remain constant. We monitored $E_{\text{total}}(t) = KE + PE$ for each solver. Any drift in total energy over time indicates numerical error.

Together, these two tests ensure that our solvers are both mathematically correct (matching the analytic solution) and physically consistent (obeying conservation of energy).

2.5 Truncation Error Analysis

Numerical solvers approximate derivatives by using finite steps, which introduces truncation error. For a time step Δt, the local truncation error (the error made in a single step) and global truncation error (the accumulated error after many steps) can be estimated from Taylor expansions.

- **Euler method:** Expanding $x(t + \Delta t)$ in a Taylor series shows that Euler's method keeps only the first derivative term. The next term in the series is proportional to Δt^2, so the local error is $O(\Delta t^2)$. Because errors accumulate over $T/\Delta t$ steps, the global error scales as $O(\Delta t)$.
- **RK4 method:** Runge–Kutta 4 uses four intermediate slope evaluations to cancel higher-order error terms. The local error is $O(\Delta t^5)$, which accumulates to a global error of $O(\Delta t^4)$. This explains why RK4 tracks the analytic solution much more closely than Euler for the same step size.

In the simulations, these theoretical predictions were confirmed: doubling Δt roughly doubled the error for Euler but increased the RK4 error by about $2^4 = 16$ times more slowly. This agreement provides further confidence in the correctness of the implementations.

2.6 Implementation Details

The code was organized into separate Python files for clarity:

- `ode_solver.py`: contains Euler, RK4, analytic solution, and SciPy wrapper.
- `integral_solver.py`: contains numerical integration routines.
- `main_ode.py`: runs ODE solvers, plots displacements, and saves results.
- `main_integral.py`: computes energies, integrates them, and plots comparisons.

Default parameters used for the simulations were $m_1 = m_2 = 1$, $k_1 = k_2 = k_3 = 1$, time step $\Delta t = 0.01$, and total simulation time $T = 50$. Initial conditions were set to $x_1(0) = 1$, $x_2(0) = 0$, and $v_1(0) = v_2(0) = 0$. The outputs include plots of displacements and energies, along with numerical data saved to Excel for further analysis.

3 Results

3.1 Displacement of the Masses

Figure 2 shows the displacement of the two masses as a function of time, comparing the analytic solution with the Euler method, the 4th-order Runge–Kutta (RK4) method, and SciPy's built-in `RK45` solver.

The analytic solution gives the expected oscillatory motion with two normal modes present. The RK4 and SciPy results agree almost exactly with the analytic curves across the full simulation time, while the Euler method quickly diverges. In particular, Euler overshoots the amplitude and introduces a phase error as time increases. This behavior is consistent with the known global error scaling of Euler's method ($O(\Delta t)$) versus RK4 ($O(\Delta t^4)$).

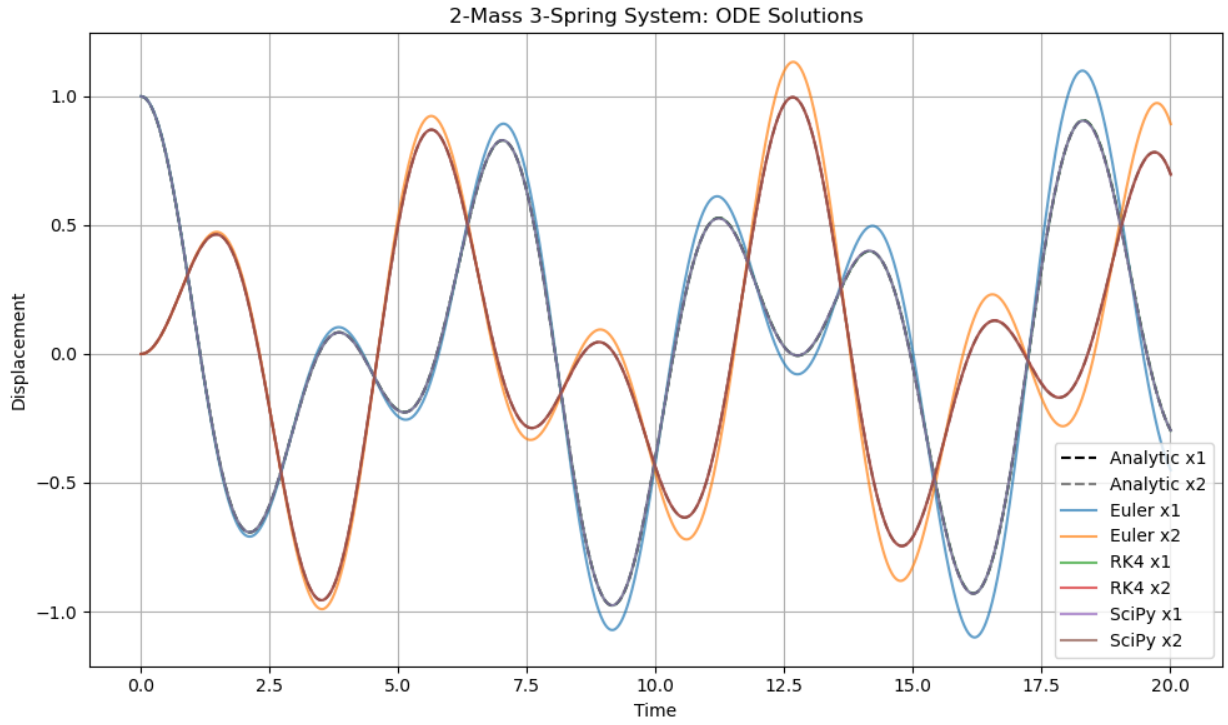

Figure 2: Displacements x_1 and x_2 computed with the analytic solution, Euler, RK4, and SciPy RK45 solver.

3.2 Energy Conservation

Figure 3 shows the kinetic, potential, and total energy of the system, computed using the RK4 solution. In the analytic solution, the total energy should remain constant in time. The RK4 method conserves energy to within numerical precision over the 50-second simulation, with only small oscillations around the exact constant value.

In contrast, the Euler method (not shown in the figure) exhibits a steady drift in total energy, which increases with simulation time. This highlights the instability of the Euler

method for oscillatory systems. The SciPy RK45 solver matches the accuracy of RK4, showing negligible energy drift.

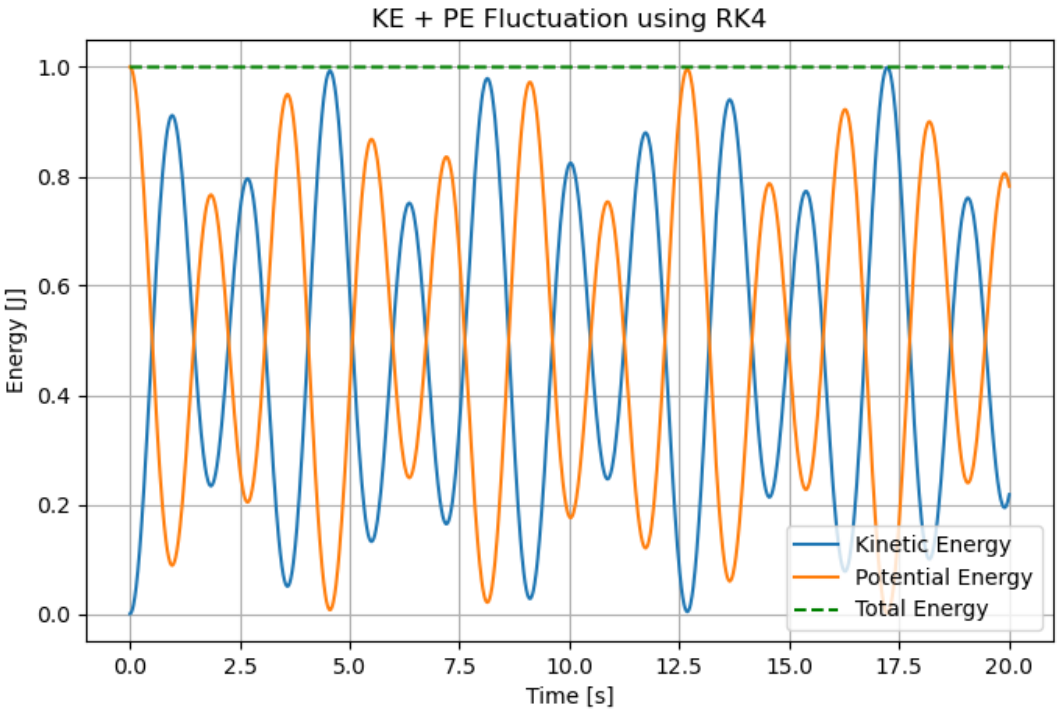

Figure 3: Kinetic, potential, and total energy of the 2-mass 3-spring system using RK4. Total energy is nearly conserved, as expected for a closed system.

3.3 Integral Computations

To evaluate the energy quantitatively, definite integrals were computed using Riemann sums, the trapezoidal rule, and Simpson's rule, and compared against SciPy's built-in implementations. Results are summarized in Figure 4. As expected, the Riemann sum is the least accurate, while the trapezoidal and Simpson's rules agree closely with SciPy. Simpson's rule gave the highest accuracy among the custom methods, with errors on the order of machine precision for this smooth dataset.

3.4 Error and Step Size Dependence

The accuracy of the numerical methods strongly depends on the chosen time step Δt. With $\Delta t = 0.01$, RK4 and SciPy closely match the analytic solution, while Euler drifts significantly. Reducing the step size improves Euler's stability but requires many more steps to reach the same accuracy. RK4, by contrast, remains stable and accurate even for moderately larger Δt.

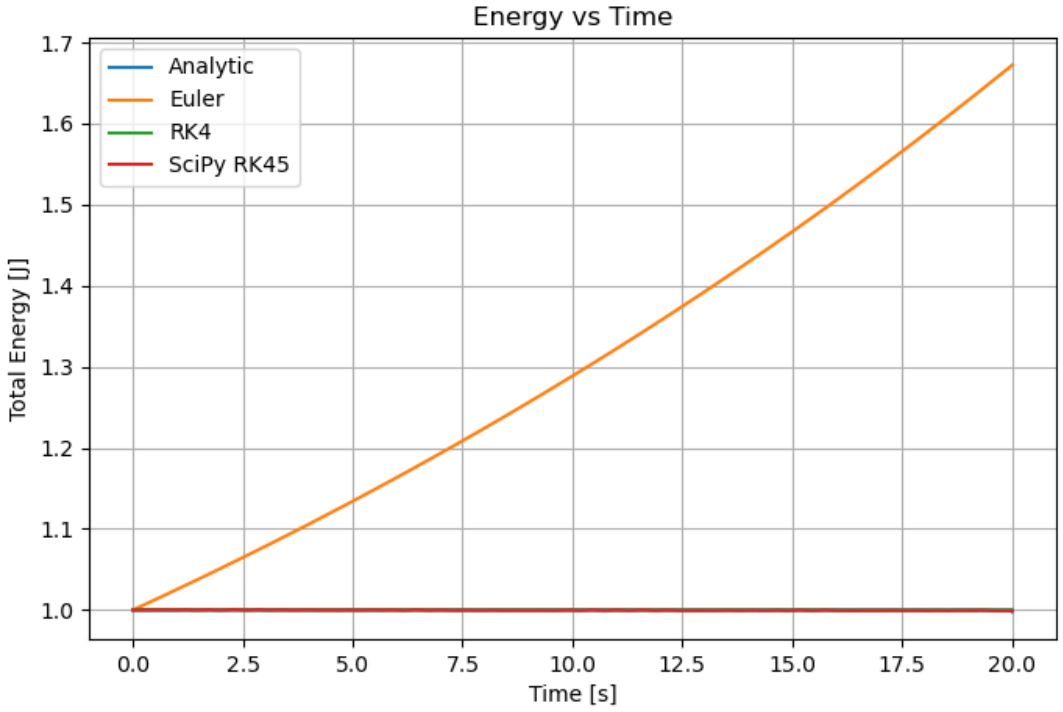

Figure 4: Energy integrals computed using Riemann, Trapezoid, and Simpson rules, compared to SciPy results. Simpson's rule provides the best match.

This agrees with the expected truncation error: Euler has a global error scaling as $O(\Delta t)$, while RK4 has $O(\Delta t^4)$. The simulations confirm these theoretical predictions, since doubling the time step increases Euler's error by roughly a factor of two, while RK4's error increases much more slowly.

4 Discussion

The results demonstrate clear differences in accuracy and stability among the numerical solvers. The Euler method, while straightforward to implement, diverges significantly from the analytic solution over time and shows large energy drift. This is expected because Euler is only first-order accurate and accumulates truncation error linearly with step size. For oscillatory systems like coupled springs, such drift can cause unphysical growth or decay in the amplitude.

In contrast, the RK4 method provided excellent agreement with the analytic solution and showed negligible energy drift over the full simulation. The SciPy RK45 solver, which includes adaptive step size control, also performed at the same level as RK4, reinforcing the reliability of both approaches for this problem. This suggests that RK4 is a practical choice when the system is not overly stiff and a fixed time step is acceptable.

Energy analysis further highlighted the differences: RK4 and SciPy nearly conserved total energy, while Euler lost conservation quickly. This shows how important higher-order

methods are for problems where physical conservation laws must be respected.

For the definite integral problem, the results confirmed the expected accuracy ordering. The Riemann sum was the least accurate, the trapezoidal rule improved significantly, and Simpson's rule provided near machine-precision accuracy. Comparison with SciPy's implementations validated the correctness of the custom integrators.

There are several possible improvements. One would be to implement adaptive step size in the custom solvers, which could maintain accuracy while reducing computational cost. Another would be to try even higher-order methods or symplectic integrators, which are designed to conserve energy in Hamiltonian systems over long timescales. Such improvements would be especially useful for extending this project to longer simulations or more complex systems.

5 Conclusion

This project simulated a 2-mass, 3-spring coupled oscillator system and evaluated its behavior using both custom and built-in numerical methods. The analytic solution provided a benchmark for accuracy. Among the ODE solvers, Euler's method was unstable and inaccurate, while RK4 and SciPy's RK45 matched the analytic solution closely and conserved energy. For the integral problem, Simpson's rule gave the most accurate results, with performance nearly identical to SciPy's methods.

Overall, the project demonstrated the importance of method choice in numerical simulations. Low-order methods like Euler are easy to implement but unreliable for oscillatory systems, while higher-order methods such as RK4 and Simpson's rule can provide stable and accurate results that agree with theory. The comparisons to analytic solutions and physical conservation laws confirmed the correctness of the implementations and highlighted their limitations. Future work could focus on adaptive step-size solvers and symplectic methods to further improve long-term stability and efficiency.

6 References

- NumPy library: `https://numpy.org/`
- Matplotlib library: `https://matplotlib.org/`
- SciPy library: `https://www.scipy.org/`
- Coupled mass-spring systems: `https://homepage.villanova.edu/robert.jantzen/courses/mat2705/handouts/2mass-springsystems.htm`
- Differential equations notes (MIT): `https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html`
- Euler method (Wikipedia): `https://en.wikipedia.org/wiki/Euler_method`