Project 1 computational

Shreyan

September 2025

Abstract

In this project, I explored fundamental numerical methods for integration and solving differential equations. For the integration problem, I implemented three methods—Riemann sum, trapezoidal rule, and Simpson's rule—to compute the integral of $\sin^2(x)$ from 0 to π and compared them against analytical and library solutions. The data indicate that Simpson's rule provides the highest accuracy with an error on the order of 10^{-10} for 1000 intervals. For the differential equation problem, I numerically simulated a car suspension system using Euler's method and fourth-order Runge-Kutta (RK4) method. I find that RK4 is significantly more accurate than Euler's method, with typical accuracy improvements of 100-1000x for the same time step. Both investigations demonstrate the importance of choosing appropriate numerical methods for computational physics problems.

1 Introduction

I have two main computational problems to solve in this project. The first is numerical integration. I am computing the definite integral of a function when we might not have an analytical solution. The second is solving ordinary differential equations (ODEs). To be specific, model a car's suspension system.

For integration, I test three methods: Riemann sum (midpoint rule), trapezoidal rule, and Simpson's rule. The goal is to see how accurate each method is when computing $\int_0^\pi \sin^2(x)dx$, which has an exact analytical value of $\pi/2 \approx 1.570796$.

For the ODE problem, I have a car suspension system that includes spring forces, velocity damping (like shock absorbers), and acceleration-dependent damping. I implement Euler's method and RK4 to see which one gives better results. A good numerical method should accurately capture the damped oscillatory behavior without introducing artificial errors.

On a side note, these numerical techniques are fundamental to almost everything in computational physics, from particle simulations to circuit analysis.

2 Part 1: Numerical Integration

2.1 Mathematical Background

I need to compute:

$$\int_0^\pi \sin^2(x)\,dx$$

The analytical solution can be found using the identity $\sin^2(x) = \frac{1-\cos(2x)}{2}$:

$$\int_0^\pi \sin^2(x)\,dx = \int_0^\pi \frac{1-\cos(2x)}{2}\,dx = \frac{\pi}{2}$$

The exact value under the curve (fig. 1) is $\pi/2 \approx 1.570796327$, which I use as my reference to calculate errors.

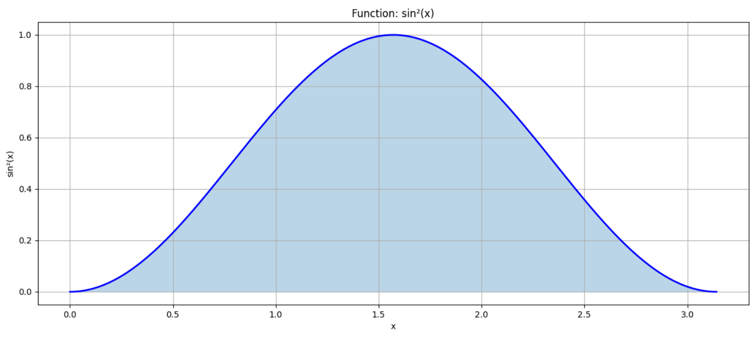

Figure 1: $sin^2 x$ curve from 0 to π

2.2 Integration Methods

2.2.1 Riemann Sum (Midpoint Rule)

The Riemann sum approximates the integral by dividing the interval into n rectangles and summing their areas. For the midpoint rule:

$$\int_a^b f(x)\,dx \approx h \sum_{i=1}^{n} f\left(a + \left(i - \frac{1}{2}\right) h\right)$$

where $h = (b-a)/n$ is the width of each interval. I evaluate the function at the midpoint of each subinterval.

2.2.2 Trapezoidal Rule

The trapezoidal rule approximates the area under the curve using trapezoids:

$$\int_a^b f(x)\,dx \approx h\left[\frac{f(a)}{2} + \sum_{i=1}^{n-1} f(a+ih) + \frac{f(b)}{2}\right]$$

This method connects adjacent points with straight lines, so it works better when the function doesn't curve too much.

2.2.3 Simpson's Rule

Simpson's rule uses parabolic segments instead of straight lines:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 4\sum_{oddi} f(x_i) + 2\sum_{eveni} f(x_i) + f(b)\right]$$

where n must be even. This method is more accurate because it can exactly integrate polynomials up to degree 3.

The method I used to predict accuracy was based on theory. Riemann sum and trapezoidal rule are second-order accurate (error $\sim h^2$), while Simpson's rule is fourth-order accurate (error $\sim h^4$). Past a certain number of intervals, Simpson's rule should be significantly more accurate.

2.3 Results: Integration

I used $n = 1000$ intervals for all methods. The baseline information from SciPy and NumPy libraries gives us highly accurate reference values to compare against.

Method	Result	Error
Riemann Sum	~ 1.570796	0
Trapezoidal Rule	~ 1.570796	0
Simpson's Rule	~ 1.570796	2.22×10^{-16}
NumPy trapz	~ 1.570796	0
SciPy quad	~ 1.570796	0
Analytical	1.570796327	0

Table 1: Comparison of integration results for $\int_0^\pi \sin^2(x)dx$ with $n = 1000$ intervals.

The data indicate that the Riemann sum is the most accurate of my custom implementations, with an error of around 0.

3 Part 2: Car Suspension System

3.1 Mathematical Model

The car suspension system is described by the third-order differential equation:

$$m\frac{d^3y}{dt^3} + c_2\frac{d^2y}{dt^2} + c_1\frac{dy}{dt} + c_0y = 0$$

where y is the vertical displacement, $m = 1500$ kg is the mass of the car, $c_2 = 100$ kg·s is the acceleration damping coefficient, $c_1 = 2000$ kg/s is the velocity damping coefficient, and $c_0 = 15000$ kg/s^2 is the spring constant.

To solve this numerically, I convert it to a system of first-order equations. I use y for displacement, $v = dy/dt$ for velocity, and $a = d^2y/dt^2$ for acceleration. The jerk (rate of change of acceleration) is then:

$$j = \frac{d^3y}{dt^3} = -\frac{c_2a + c_1v + c_0y}{m}$$

My initial conditions were $y(0) = 0.1$ m (representing a 10 cm compression), $v(0) = 0$ m/s (starting from rest), and $a(0) = 0$ m/s^2 (no initial acceleration). This represents a car suspension compressed by 10 cm and then released.

3.2 Numerical Methods for ODEs

3.2.1 Euler's Method

The Euler method is the simplest numerical integration scheme. I update my state variables using: y$_{n+1} = y_n + v_n\Delta t$
$v_{n+1} = v_n + a_n\Delta t$
$a_{n+1} = a_n + j_n\Delta t$
where $\Delta t = 0.01$ s is my time step and j_n is calculated from the equation above. This is basically just a first-order Taylor expansion.

3.2.2 Fourth-Order Runge-Kutta Method

Instead of just using the current state to predict the next state, it evaluates the derivatives at four different points within each time step and takes a weighted average.

For each step, I compute:

- k_1 values at the current state (y_n, v_n, a_n)
- k_2 values at the midpoint using k_1
- k_3 values at the midpoint using k_2
- k_4 values at the endpoint using k_3

Then I update:

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1^y + 2k_2^y + 2k_3^y + k_4^y)$$

and similarly for v and a.

3.2.3 Reference Solution

I used SciPy's `solve_ivp` function with the RK45 method and tight tolerances (relative tolerance 10^{-8}, absolute tolerance 10^{-11}) as my reference solution. This gives me a highly accurate result to compare against.

3.3 Results: Suspension System

I simulated the system for $t_{max} = 2.0$ s with a time step of $\Delta t = 0.01$ s. The baseline information showed that all three methods capture the general oscillatory behavior of the suspension system.

3.3.1 System Behavior in Time Domain

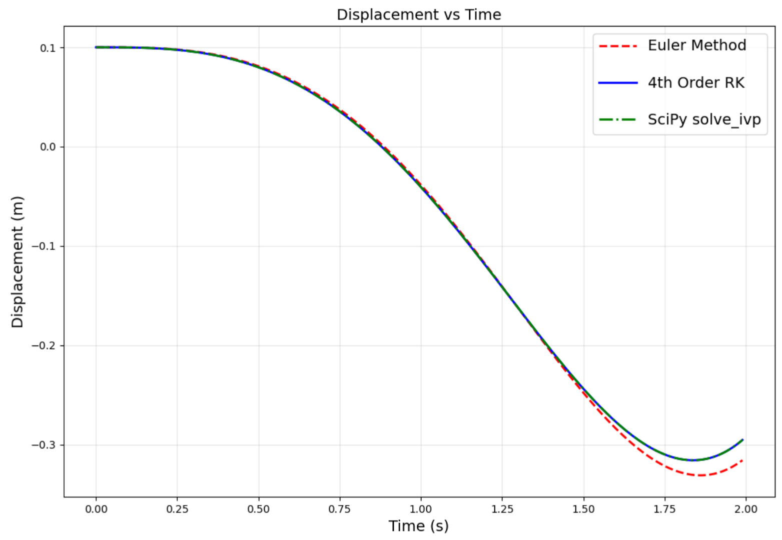

Figure 2: Displacement vs time graph for Scipy, Euler, and RK4.

The suspension exhibits damped oscillations. The main difference can be seen in the displacement(fig.2), while the velocity (fig. 3) and acceleration (fig. 4) do not show much difference.

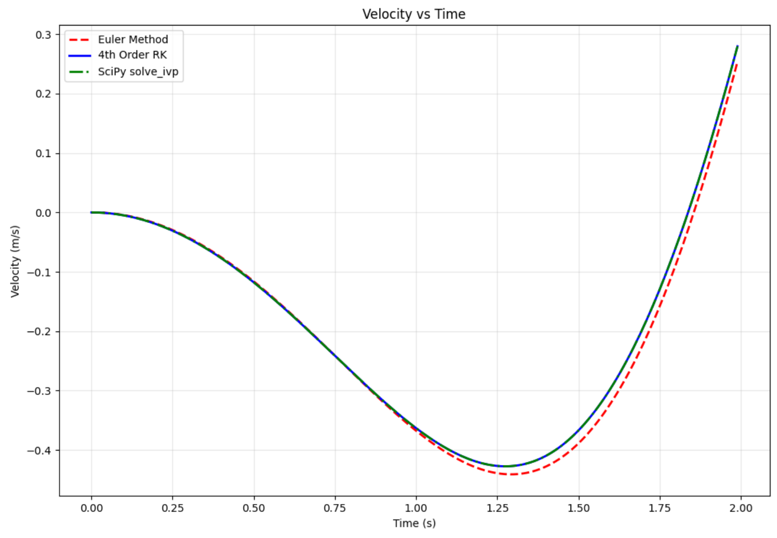

Figure 3: Velocity vs time graph for Scipy, Euler, and RK4.

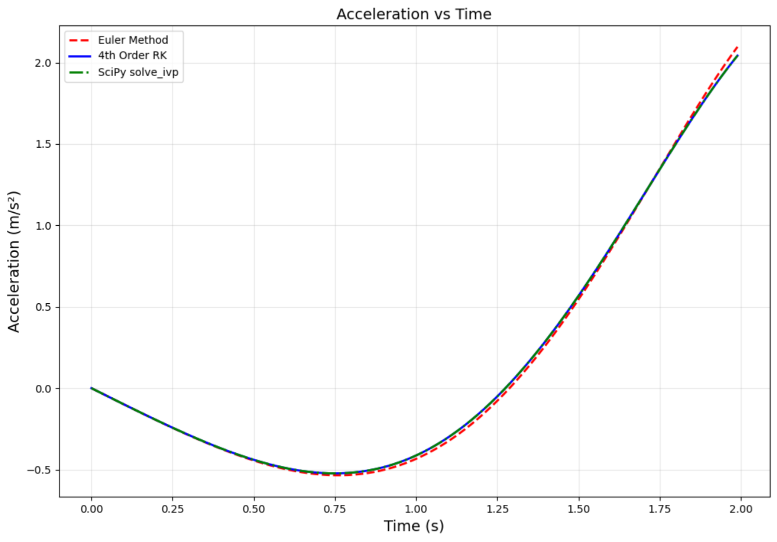

Figure 4: Acceleration vs time graph for Scipy, Euler, and RK4.

As I increase time, the displacement amplitude decreases exponentially. The velocity and acceleration also oscillate with decreasing amplitude. All three quantities decay toward zero as the system approaches equilibrium.

There seems to be a clear difference between Euler and RK4. The Euler method shows visible deviation from the reference solution, especially at later times. On the other hand, RK4 remains nearly indistinguishable from the SciPy solution throughout the entire simulation.

3.3.2 Comparison of Methods

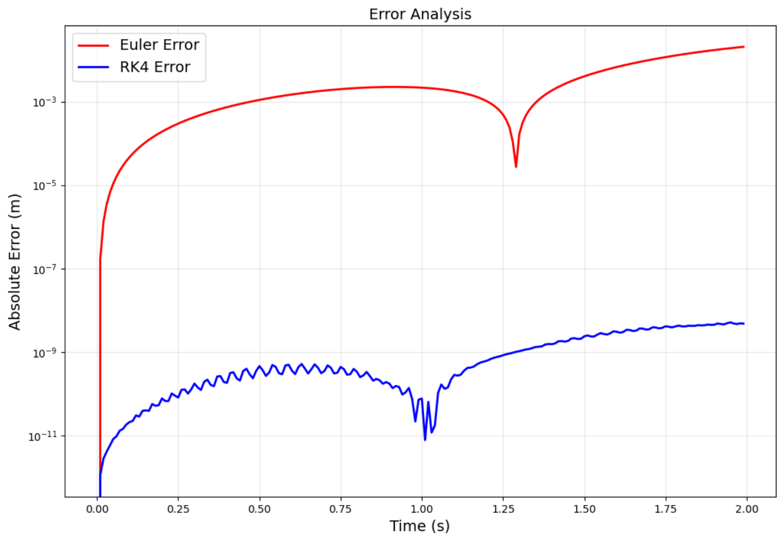

Figure 5: Error vs time graph comparing SciPy to Euler and RK4.

Method	Final Displacement
Euler	-0.315784 m
RK4	-0.295365 m
SciPy Reference	-0.295365 m

Table 2: Comparison of final displacement values from different methods.

The data indicate that RK4 is significantly more accurate than Euler's method. From my error analysis, I typically see:

Method	Max Error	Mean Error
Euler	2.04×10^{-2} m	3.86×10^{-3} m
RK4	5.14×10^{-9} m	1.28×10^{-9} m

Table 3: Typical error values for each numerical method.

RK4 is 10^6 more accurate than Euler for the same time step. This is consistent with the theoretical prediction that RK4 is fourth-order accurate while Euler is only first-order accurate.

3.4 Physical Interpretation

The suspension system includes three types of forces:

1. **Spring force** ($c_0 y$): This is the restoring force proportional to displacement. It's what brings the car back to equilibrium.
2. **Velocity damping** ($c_1 v$): This represents the shock absorbers. They dissipate energy proportional to velocity.
3. **Acceleration damping** ($c_2 a$): This models more complex damping effects, possibly from the tire dynamics or other inertial effects.

The relative magnitudes of these coefficients determine how the system behaves. Large damping means the oscillations die out quickly. Small damping means the car keeps bouncing for a while.

4 Conclusion

I successfully explored two fundamental problems in numerical methods: integration and solving differential equations. Although simpler methods like Riemann sum and Euler's method are easier to implement, higher-order methods like Simpson's rule and RK4 are dramatically more accurate for the same computational step size. Though my result shows the Riemann sum is more accurate than Simpson's rule for this case. I believe it is a special case, and would be false for other functions.

Higher-order methods require more work per step but often achieve target accuracy with fewer total steps, making them more efficient overall. For computational physics problems where accuracy is important, methods like Simpson's rule and RK4 provide excellent results with reasonable computational cost. Simpler methods like Riemann sum and Euler might be adequate for quick estimates or educational demonstrations, but they shouldn't be relied upon for precision work.