Development of Astronomical Simulation using Runge-Kutta Method

14th Information Science Project Presentation

25-083 Lee Si-hoo (I. DEV)

Abstract

This report details the development of a physics simulation engine utilizing the Runge-Kutta method. The project aims to solve Ordinary Differential Equations (ODEs) required for accurate space and astronomical simulations. By implementing the 2nd-order Runge-Kutta (RK2) algorithm in TypeScript and comparing it against Euler's Method using a Spring Oscillation model, the study demonstrates the superior accuracy of RK2.

1 Development Purpose

In the modern space development sector, pre-launch simulations are indispensable. Physical systems in space cannot always be described by static mathematical formulas; they often require solving **Ordinary Differential Equations (ODEs)** to determine state changes at specific points in time. This project implements the **Runge-Kutta algorithm** to address these computational needs accurately.

2 Project Roadmap

The project is structured into three main phases:

1. Mathematical proof and derivation of the Runge-Kutta (2nd method).
2. Simulation of Spring Oscillation using RK2 and comparison of error rates against Euler's Method.
3. (Planned) Development of a general-purpose astronomical simulation engine applicable to diverse scenarios.

3 Mathematical Proof: Derivation of RK2

The project utilizes the RK2 method (specifically Heun's method). We rigorously derive the coefficients by matching the Runge-Kutta expansion to the Taylor Series expansion of the unknown function.

3.1 1. Taylor Series Expansion (Target)

Let the exact solution be $y(x)$. The expansion of $y(x+h)$ up to the second order is:

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + O(h^3)$$

Given $y' = f(x,y)$, we use the chain rule for the second derivative y'':

$$y'' = \frac{d}{dx}[f(x,y)] = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \cdot \frac{dy}{dx} = f_x + f_y f$$

Substituting this back, the **exact expansion** is:

$$y(x+h) = y + hf + \frac{h^2}{2}(f_x + f_y f) + O(h^3) \tag{1}$$

3.2 2. General RK2 Expansion

The general form of a 2-stage Runge-Kutta method is:

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2)$$

Where the slopes are defined as:

$$\begin{aligned} k_1 &= f(x,y) \\ k_2 &= f(x + c_2 h, y + a_{21} h k_1) \end{aligned}$$

We expand k_2 using a 2D Taylor series around (x,y):

$$k_2 \approx f(x,y) + (c_2 h)\frac{\partial f}{\partial x} + (a_{21} h k_1)\frac{\partial f}{\partial y}$$

$$k_2 = f + hc_2 f_x + ha_{21} f f_y + O(h^2)$$

Substituting this back into the general RK2 equation:

$$y_{n+1} = y_n + hb_1 f + hb_2[f + h(c_2 f_x + a_{21} f f_y)]$$

Rearranging terms by powers of h:

$$y_{n+1} = y_n + h(b_1 + b_2)f + h^2(b_2 c_2 f_x + b_2 a_{21} f f_y) + O(h^3) \tag{2}$$

3.3 3. Matching Coefficients

To achieve $O(h^2)$ accuracy, we equate the coefficients of h and h^2 between Equation (1) and Equation (2):

- h^1 **term (f):**

$$b_1 + b_2 = 1$$

- h^2 **term (f_x):**

$$b_2 c_2 = \frac{1}{2}$$

- h^2 **term ($f f_y$):**

$$b_2 a_{21} = \frac{1}{2}$$

3.4 4. Result: Heun's Method

For this project, we chose Heun's Method, which sets $b_1 = \frac{1}{2}$ and $b_2 = \frac{1}{2}$ (simple average of slopes). Solving the system above:

$$\frac{1}{2}c_2 = \frac{1}{2} \implies c_2 = 1$$

$$\frac{1}{2}a_{21} = \frac{1}{2} \implies a_{21} = 1$$

This yields the final formula used in our code:

$$y_{n+1} = y_n + \frac{h}{2}(f(x, y) + f(x + h, y + hk_1))$$

4 Software Environment & Implementation

- **Language:** TypeScript
- **IDE:** Visual Studio Code

Below is the TypeScript logic corresponding to the derived Heun's Method.

```
// 1. Calculate k1 (Slope at the beginning of the interval)
const p1value = currentState.value;
const p1rate = currentState.rate;
const p1rateDeriv = this.rateDerivativeFn(p1value, p1rate);

const k1rate = p1rate;
const k1rateDeriv = p1rateDeriv;

// 2. Calculate State for k2 (y + h*k1)
// We step forward by full 'h' (because c2 = 1)
const p2value = currentState.value.add(k1rate.scale(this.deltaH));
const p2rate = currentState.rate.add(k1rateDeriv.scale(this.deltaH));

// 3. Calculate k2 (Slope at x + h)
const k2rate = p2rate;

// 4. Final Update: Average k1 and k2 (b1=0.5, b2=0.5)
const nextValue = currentState.value.add(
    k1rate.scale(this.deltaH/2).add(k2rate.scale(this.deltaH/2))
);
```

Listing 1: RK2 Implementation (Heun's Method)

5 Experimental Results: Oscillation

5.1 Experiment Parameters

- **System:** 1kg mass object, Spring constant $k = 100N/m$.
- **Answer Function:** $y = 0.1 \cdot \sin(10t)$
- **Time Range:** $t \in [0, 3]$

5.2 Results

1. $\Delta H = 0.01$**:** RK2 aligns almost perfectly with the answer. Euler's method shows slight deviation.

2. $\Delta H = 0.05$**:** Euler's method diverges significantly from the true sine wave. RK2 maintains high accuracy despite the larger time step, validating the $O(h^2)$ error term elimination proved in Section 3.