Project: Ordinary Differential Equations with Initial Value Problem

Hieu Bui

November 12, 2022

1 Introduction

The project focus on solving initial value problem numerically. We still go through the overarching procedure of the grid, the scheme, and the solver. Initial value problem only have a start condition, and the next state is described by the governing equation.

2 Methods

2.1 Euler's Method

Given an affine equation, we can reach the next point in the grid through its slope. The slope is in term of the current point, so the equation is explicit. As we march to the right, we use the newly acquired point to get the next one. For the falling body problem, we have an expression for $\dot{v}$ and $\dot{x}$.

$$\dot{v} = \frac{c_1}{m} - \frac{c_2 v}{m}$$
$$\dot{x} = v$$

We have: $v_{i+1} = \dot{v}_i \Delta t + v_i$ and $x_{i+1} = \dot{x}_i \Delta t + x_i$

$$v_{i+1} = (\frac{c_1 \Delta t}{m} - \frac{c_2 v \Delta t}{m}) + v_i$$
$$x_{i+1} = v_i \Delta t + x_i$$

2.2 Backward Euler's Method

The Backward Euler's has the same working principle as the Forward Euler's. However, the slope of the scheme is using the next point in the grid and not the current one, which means the equation is in an implicit form. If we move everything to one side, then we have a root formula, suggesting the use of root finding method. Therefore, we will combined like terms to isolate the point of

interest then use a root finding scheme to solve. Let us convert the implicit form into the explicit form.

For v:

$$v_{i+1} = v_i + (\frac{c_1}{m} - \frac{c_2 v_{i+1}}{m})\Delta t$$
$$v_{i+1} = v_i + \frac{c_1 \Delta t}{m} - \frac{c_2 v_{i+1} \Delta t}{m}$$
$$v_{i+1} + \frac{c_2 v_{i+1} \Delta t}{m} = v_i + \frac{c_1 \Delta t}{m}$$
$$v_{i+1}(1 + \frac{c_2 \Delta t}{m}) = v_i + \frac{c_1 \Delta t}{m}$$
$$v_{i+1} = \frac{v_i + \frac{c_1 \Delta t}{m}}{1 + \frac{c_2 \Delta t}{m}}$$
$$v_{i+1} = \frac{v_i m + c_1 \Delta t}{m + c_2 \Delta t}$$

For x:

$$x_{i+1} = x_i + v\Delta t$$

2.3 Crank Nicolson

Crank Nicolson is an average of Forward Euler and Backward Euler. Therefore, it will have an implicit component, which means we need to convert it to explicit to solve.

For v:

$$v_{i+1} = \frac{1}{2}(\frac{c_1}{m} - \frac{c_2 v_{i+1}}{m} + \frac{c_1}{m} - \frac{c_2 v_i}{m})\Delta t + v_i$$
$$v_{i+1} - \frac{\Delta t c_1}{2m} + \frac{c_2 v_{i+1} \Delta t}{2m} = \frac{c_1 \Delta t}{2m} - \frac{c_2 \Delta t v_i}{2m} + v_i$$
$$v_{i+1}(1 + \frac{c_2 \Delta t}{2m}) = \frac{c_1 \Delta t}{m} + v_i(1 - \frac{c_2 \Delta t}{2m})$$
$$v_{i+1} = \frac{2C_1 \Delta t + v_i(2m - c_2 \Delta t)}{2m + c_2 \Delta t}$$

For x:

$$x_{i+1} - x_i = \frac{\Delta t}{2}\dot{x} + \frac{\Delta t}{2}\dot{x}_{i+1}$$
$$x_{i+1} = \frac{\Delta t v_i}{2} + \frac{\Delta t v_{i+1}}{2} + x_i$$

2.4 Leap Frog

Unlike Backward and Forward Euler's, Leap Frog method requires the solver to know the information about both the previous and future point. Therefore, we need to know 2 different points in total to solve instead of 1. However, we are only given 1 initial point; we need to get one more point to start the solver. We use Forward Euler method to get the second point.

For v:

$$v_{i+1} = 2\Delta t(\frac{c_1}{m} - \frac{c_2 v_i}{m}) + v_{i-1}$$

For x:

$$x_{i+1} = 2\Delta t v_i + v_{i-1}$$

2.5 Runge Kutta

The Runge Kutta method uses Taylor's Series Expansion to reduce error in the slope deviation; it is much more accurate than the other ones we have discussed especially if the governing equation is non linear. The more k terms we add, the lower the order of errors. The scheme for Runge Kutta is explicit, so we do not have to do explicit conversion. In general, the scheme takes the form of $y_{i+1} = y_i +$ Runge-Kutta-estimation.

3 Programming

3.1 Forward Euler's

- Inputs: slope, current point, time step
- Outputs: x,v
- Start loop

- initialize initial point
- calculate slope
- calculate the new point with derived scheme
- equate new point to current point

3.2 Backward Euler's

- Inputs: m,c1,c2,v,x,time step
- Outputs: x,v
- Start loop
- Apply derived explicit equation
- Set input equal to newly calculated output from previous step

3.3 Leap Frog

- Inputs: initial point, Forward Euler solver, time step
- Outputs: next point
- Apply Forward Euler's to compute 2nd point
- Start loop
- Compute derivative for the 2nd point
- Apply the solver for Leap Frog

3.4 Crank Nicolson

- Inputs: current point, constants for governing equation, time step
- Outputs: x,v
- Start loop
- initialize initial point
- calculate the new point with derived solver
- equate new point to current point

3.5 Runge Kutta

- Inputs: initial conditions, time step
- Outputs: x,v
- Start loop
- Compute k values based on the order we want
- Apply appropriate scheme for the order we want
- equate new point to current point

4 Results

The Forward Euler's overshoot the real values while the Backward Euler's undershoot the actual values. The Leap Frog oscillate about the actual values. The Runge Kutta method converges to the actual values with higher orders.