Computional Homework17 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Introduction

In this report, we compare two classical numerical ODE solvers:

- The fourth-order Runge–Kutta (RK4) method.
- The second-order Improved Euler (Heun) method.

We apply both methods to the initial-value problem

$$\frac{dy}{dx} = f(x, y), \quad y(0) = 1, \quad x \in [0, 1],$$

whose analytical solution is

$$y(x) = \sqrt{2x + 1}.$$

We investigate accuracy, computational cost, and visualize the performance of both schemes.

2 Runge–Kutta 4 Method

The classical fourth-order Runge–Kutta method updates the solution using

$$\begin{aligned}
k_1 &= f(x_n, y_n), \\
k_2 &= f\left(x_n + \frac{h}{2},\ y_n + \frac{h}{2}k_1\right), \\
k_3 &= f\left(x_n + \frac{h}{2},\ y_n + \frac{h}{2}k_2\right), \\
k_4 &= f\left(x_n + h,\ y_n + hk_3\right),
\end{aligned}$$

and the update formula is

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

The RK4 method has local truncation error $O(h^5)$ and global error $O(h^4)$.

```
def RungeKutta4(self,h):
    N=int(np.ceil((self.xe-self.x0)/h))
```

```
    y_values=np.zeros(N+1)
    y_values[0]=self.y0
    x_values=np.zeros(N+1)
    x_values[0]=self.x0
    x,y=self.x0,self.y0
    t_s=time.time()
    for ii in range(1,N+1):
        y=y_values[ii-1]
        if x+h>self.xe:
            h=self.xe-x
        k1=self.f(x,y)
        k2=self.f(x+h/2,y+h*k1/2)
        k3=self.f(x+h/2,y+h*k2/2)
        k4=self.f(x+h,y+h*k3)
        y_values[ii]=y+h/6*(k1+2*k2+2*k3+k4)
        x_values[ii]=x_values[ii-1]+h
        x+=h
    t_e=time.time()
    return y_values,x_values,t_e-t_s
```

3 Improved Euler Method

The Improved Euler method (Heun method) uses the predictor-corrector form:

$$k_1 = f(x_n, y_n),$$

$$k_2 = f(x_n + h,\ y_n + hk_1),$$

and updates the solution with

$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2).$$

This method has local truncation error $O(h^3)$ and global error $O(h^2)$.

```
def ImprovedEuler(self,h):
    N=int(np.ceil((self.xe-self.x0)/h))
    y_values=np.zeros(N+1)
    y_values[0]=self.y0
    x_values=np.zeros(N+1)
    x_values[0]=self.x0
    x,y=self.x0,self.y0
    t_s=time.time()
    for ii in range(1,N+1):
        y=y_values[ii-1]
```

```
            if x+h>self.xe:
                h=self.xe-x
            k1=self.f(x,y)
            k2=self.f(x+h,y+h*k1)
            y_values[ii]=y+h/2*(k1+k2)
            x_values[ii]=x_values[ii-1]+h
            x+=h
        t_e=time.time()
        return y_values,x_values,t_e-t_s
```

4 Numerical Results

We compare accuracy and computational time using different step sizes.

4.1 Large Step Size

When using relatively large step sizes, such as

$$h_{\text{RK4}} = 0.2, \qquad h_{\text{IE}} = 0.1,$$

both methods run extremely fast, and the total runtime is essentially negligible on modern hardware.

With these coarse step sizes, the accuracy results are:

$$\text{RK4: Max error} = 9.11 \times 10^{-5}, \qquad \text{IE: Max error} = 5.82 \times 10^{-3}.$$

4.2 Small Step Size

When the step size is reduced to $h = 0.002$, the accuracy increases dramatically for both methods, and the computation time begins to show a difference:

$$\text{RK4 time} = 2.00 \times 10^{-3}\,\text{s}, \qquad \text{IE time} = 1.00 \times 10^{-3}\,\text{s}.$$

Accuracy improves to:

$$\text{RK4: Max error} = 8.49 \times 10^{-13}, \qquad \text{IE: Max error} = 2.39 \times 10^{-6}.$$

The higher order of RK4 makes its accuracy far superior, but the cost also increases when the step size becomes small.

5 Figures

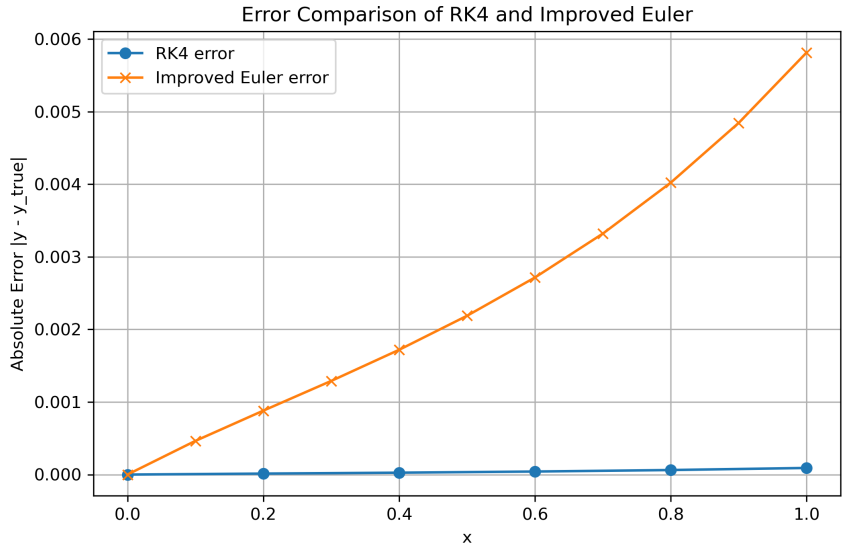

Figure 1 Error comparison between RK4 and Improved Euler at different step sizes.

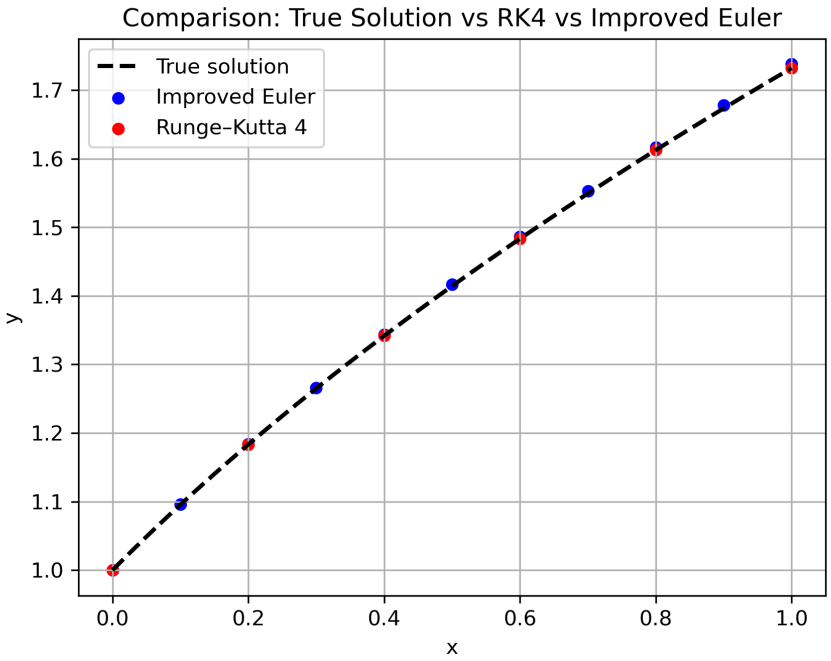

Figure 2 True solution and numerical approximations by RK4 and Improved Euler.