The sixth schoolwork of Computational Physics

万炫均 物理 1701 U201710170

Description of this chapter:

Solving ordinary derivative equations is a very common problem in physics. Here we use basic Euler method and Runge-Kutta method to solve ordinary equations by iterative methods.

Description of the problem

Homework

Write a program to solve the following ordinary differential equation by 1) basic *Euler* 2) improved *Euler* method

$$\begin{cases} y' = -x^2 y^2 & (0 \le x \le 1.5) \\ y(0) = 3 \end{cases}$$

and calculate $y(1.5)$ with stepsize=0.1, 0.1/2, 0.1/4, 0.1/8

Compare it with analytic solution (in figure)

$$y(x) = 3/(1+x^3)$$

CPCM

Homework

Write a program to solve the following ordinary differential equation by four-order *Runge-Kutta* method

$$\begin{cases} y' = -x^2 y^2 & (0 \le x \le 1.5) \\ y(0) = 3 \end{cases}$$

and calculate $y(1.5)$ with stepsize=0.1, 0.1/2, 0.1/4, 0.1/8

Compare it with analytic solution (in figure)

$$y(x) = 3/(1+x^3)$$

CPCM

42

- **Formula to use**
  - Euler Method

2 *Euler* Method

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

Integrate the equation

$$\begin{cases} y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(x, y(x))dx \\ y(x_0) = y_0 \end{cases}$$

differential equation $\Rightarrow$ **integral equation**

Improved *Euler* method

$$\begin{cases} \bar{y}_{n+1} = y_n + hf(x_n, y_n) & \text{rectangular integration} \\ y_{n+1} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})] & \\ & \text{trapezoidal integration} \\ y_0 = y(x_0) & \end{cases}$$

$$\begin{array}{ccccccc} x_1 & \to & x_2 & \to & \cdots\cdots & \to & x_n \\ \bar{y}_1 & \to & \bar{y}_2 & \to & \cdots\cdots & \to & \bar{y}_n \\ y_1 & \to & y_2 & \to & \cdots\cdots & \to & y_n \end{array} \qquad \begin{array}{l} x_n = x_0 + hi \\ y_n = ? \end{array}$$

- Runge-Kutta Method

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2!}\Big(f'_x(x_n, y_n) + f'_y(x_n, y_n)\cdot f(x_n, y_n)\Big) + \cdots$$

$$y_{n+1} = y_n + hF \qquad \textit{Runge-Kutta}\ \text{Method}$$

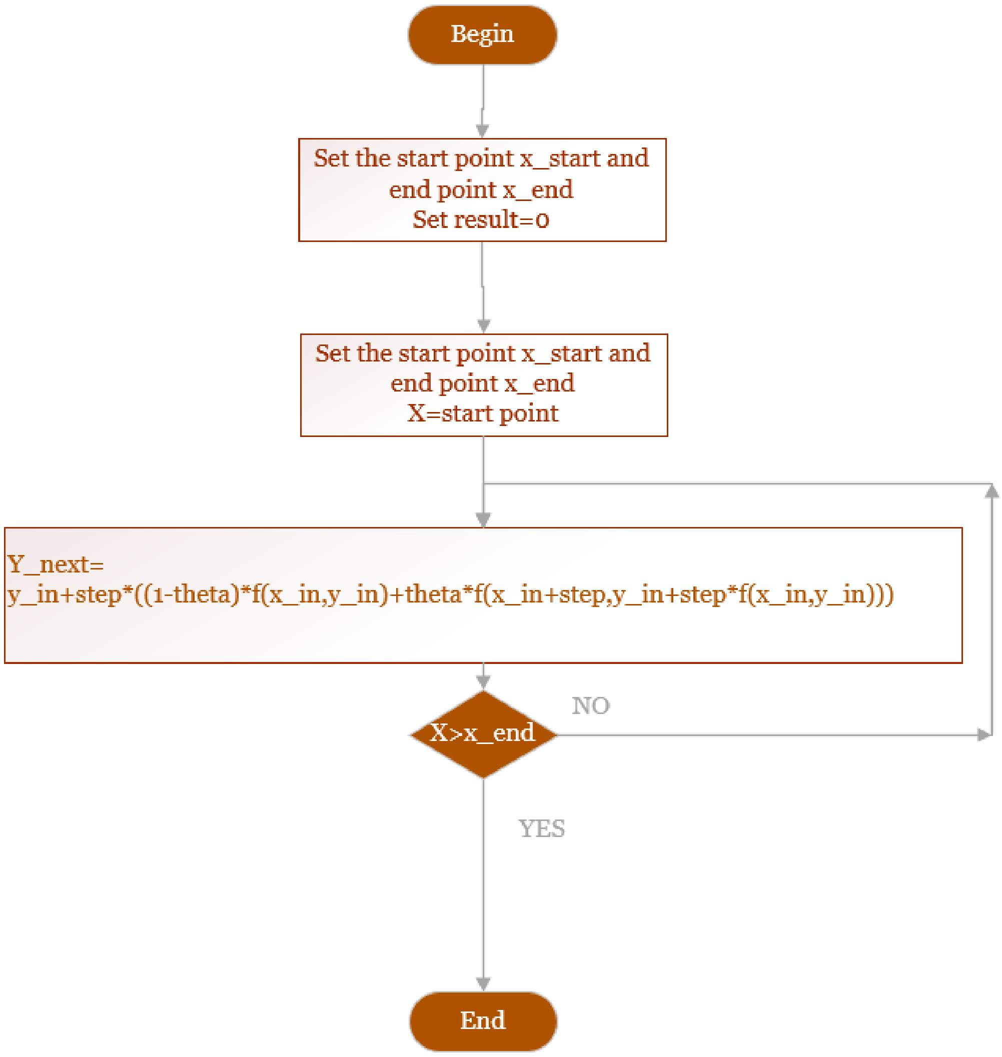
$$\Downarrow$$

$$\begin{pmatrix} c_1 f(x_n, y_n) \\ +c_2 f(x_n + \delta x, y_n + \delta y) + \cdots \end{pmatrix}$$

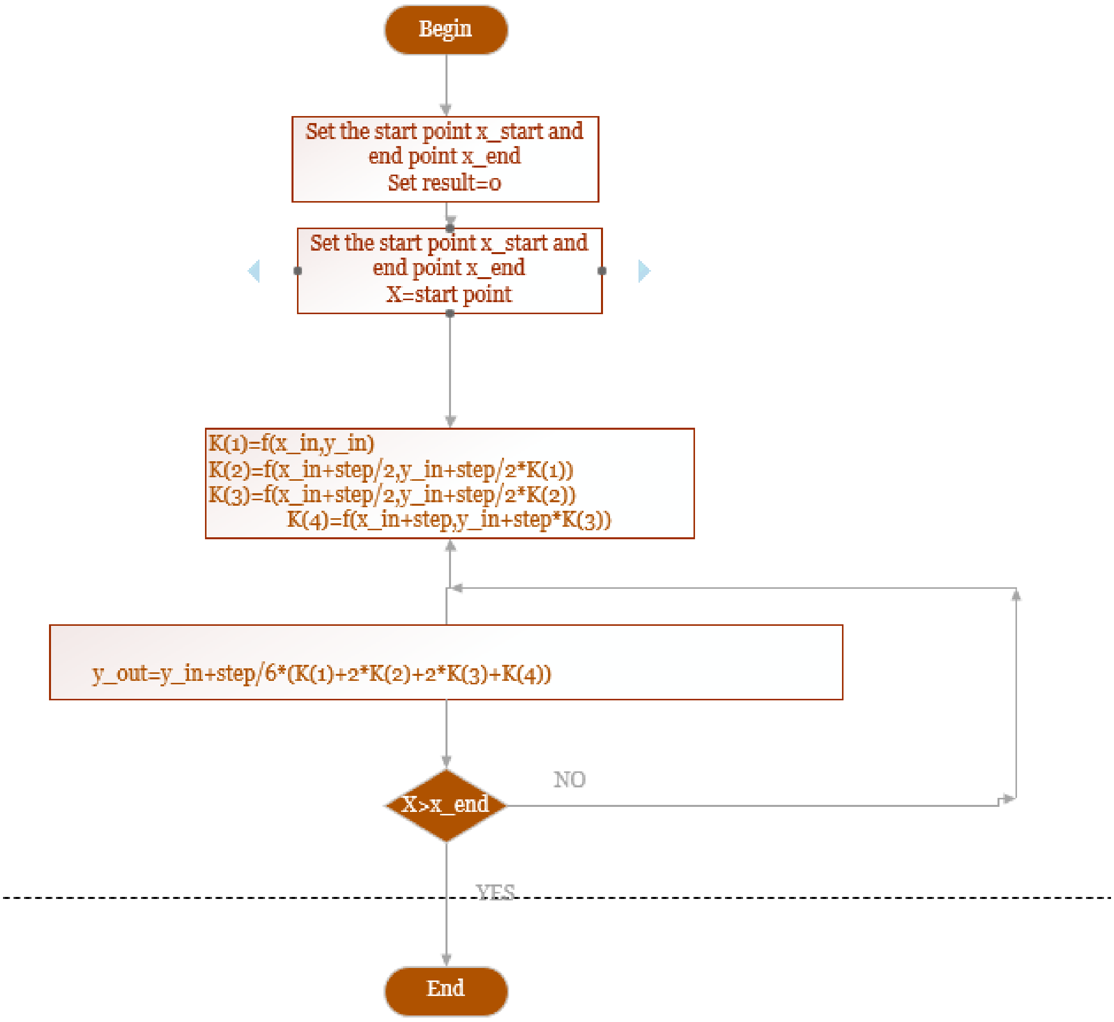
Runge-Kutta Method with 4-order precision

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_1) \\ K_3 = f(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_2) \\ K_4 = f(x_n + h, y_n + hK_3) \end{cases}$$

- **Flow chart**
  - **Euler Method Flowchart**



- **Runge-Kutta Method Flowchart**



- **Source code**

```
program MainProgram

    do i=1,4
        call Euler0(0.1/dble(i))
        call Euler5(0.1/dble(i))
        call RK4(0.1/dble(i))
    end do
end program MainProgram

subroutine Euler0(step)

    !TODO_add_body
    use ODE_Solver
    use IO


    real*8,intent(in)  :: step
```

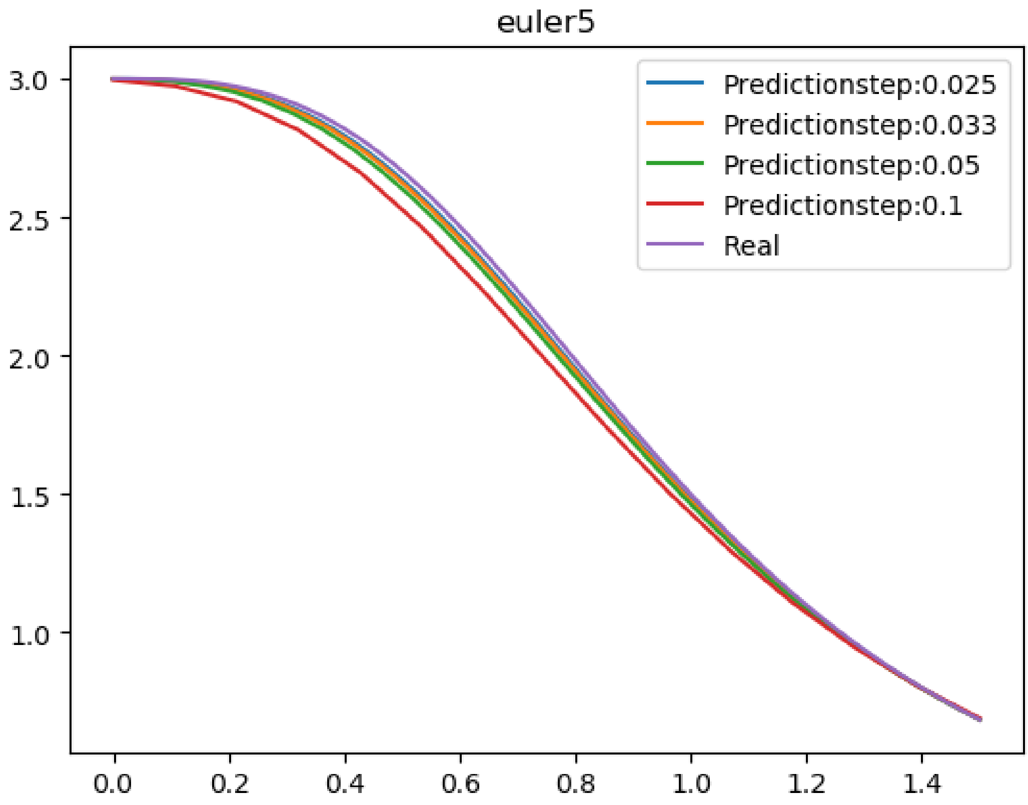
```
    real*8 :: x,y,y_next
    character(len=64) :: path
    procedure (func), pointer :: f_ptr => null()
    external f
    f_ptr => f
    x=0
    y=3
    write (path,"(a,f4.3,a)")"./data/euler0_",step,".txt"
    call ClearFile(path)
    do while(x<=1.5)
        call EulerSolver(x,y,f_ptr,step,dble(0),y_next)!0 for explicit Euler solver
        call WriteNumToFile(path,y_next,.true.)
        y=y_next
        x=x+step
    end do
end subroutine Euler0

subroutine Euler5(step)

    !TODO_add_body
    use ODE_Solver
    use IO

    real*8,intent(in)  :: step
    real*8 :: x,y,y_next
    character(len=64) :: path
    procedure (func), pointer :: f_ptr => null()
    external f
    f_ptr => f
    x=0
    y=3

    write (path,"(a,f4.3,a)")"./data/euler5_",step,".txt"

    call ClearFile(path)

    do while(x<=1.5)
        call EulerSolver(x,y,f_ptr,step,dble(0.5),y_next)!0 for explicit Euler solver
        call WriteNumToFile(path,y_next,.true.)
        y=y_next
```

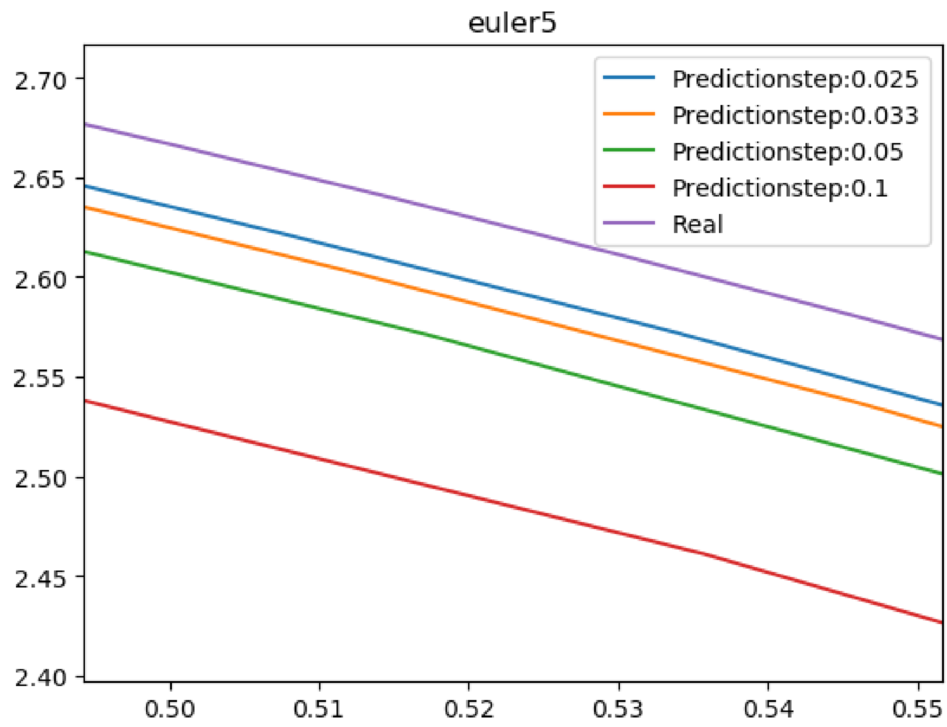
```fortran
        x=x+step
    end do
end subroutine Euler5

subroutine RK4(step)

    !TODO_add_body
    !TODO_add_body
    use ODE_Solver
    use IO

    real*8,intent(in)  :: step
    real*8 :: x,y,y_next
    character(len=64) :: path
    procedure (func), pointer :: f_ptr => null()
    external f
    f_ptr => f
    x=0
    y=3

    write (path,"(a,f4.3,a)")"./data/rk4_",step,".txt"

    call ClearFile(path)
    do while(x<=1.5)
        call RungeKutta4(x,y,f_ptr,step,y_next)!0 for explicit Euler solver
        call WriteNumToFile(path,y_next,.true.)
        y=y_next
        x=x+step
    end do
end subroutine RK4

function f (x,y)
    real*8, intent (in) :: x,y
    real*8 :: f
    f=-(x**2)*(y**2)
end function f
```

```
module ODE_Solver
    implicit none
    abstract interface
        !Function form definition of function f to define the
function pointer
        function func (x,y)
            real*8, intent (in) :: x,y
            real*8 :: func
        end function func
    end interface
contains
    subroutine EulerSolver(x_in,y_in,f_ptr,step,theta,y_out)
        !y_in:The current inputted y value
        !x_in:Thecurrent x value
        !f_ptr:The function pointer to f(x,y)
        !step:The step interval
        !theta:0 for the explicit euler function,1 for the imp
licit euler function
        !y_out:The out put of the next y value
        procedure (func), pointer,intent(in) :: f_ptr
        real*8,intent(in) :: y_in,x_in,step
        real*8,optional::theta
        real*8,intent(out) :: y_out
        if (.not.present(theta))then
            theta = 0.5
        end if

        y_out=y_in+step*((dble(1)-
theta)*f_ptr(x_in,y_in)+theta*f_ptr(x_in+step,y_in+step*f_ptr(
x_in,y_in)))

    end subroutine

    subroutine RungeKutta4(x_in,y_in,f_ptr,step,y_out)
        !y_in:The current inputted y value
        !x_in:Thecurrent x value
        !f_ptr:The function pointer to f(x,y)
        !step:The step interval
        !y_out:The out put of the next y value
        procedure (func), pointer :: f_ptr
        real*8,intent(in) :: y_in,x_in,step
        real*8,intent(out) :: y_out
```

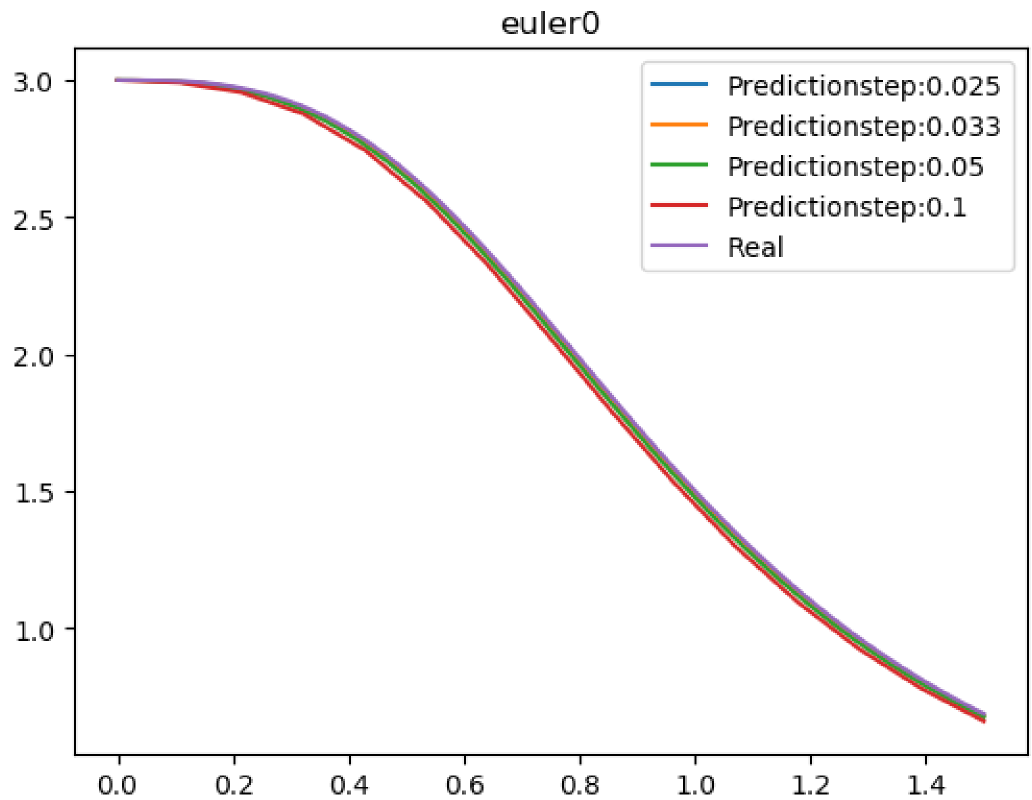
```
        !Local vars
        real*8 :: K(4)
        !Setup K values
        K(1)=f_ptr(x_in,y_in)
        K(2)=f_ptr(x_in+step/2,y_in+step/2*K(1))
        K(3)=f_ptr(x_in+step/2,y_in+step/2*K(2))
        K(4)=f_ptr(x_in+step,y_in+step*K(3))

        y_out=y_in+step/6*(K(1)+2*K(2)+2*K(3)+K(4))

    end subroutine
end module ODE_Solver
```

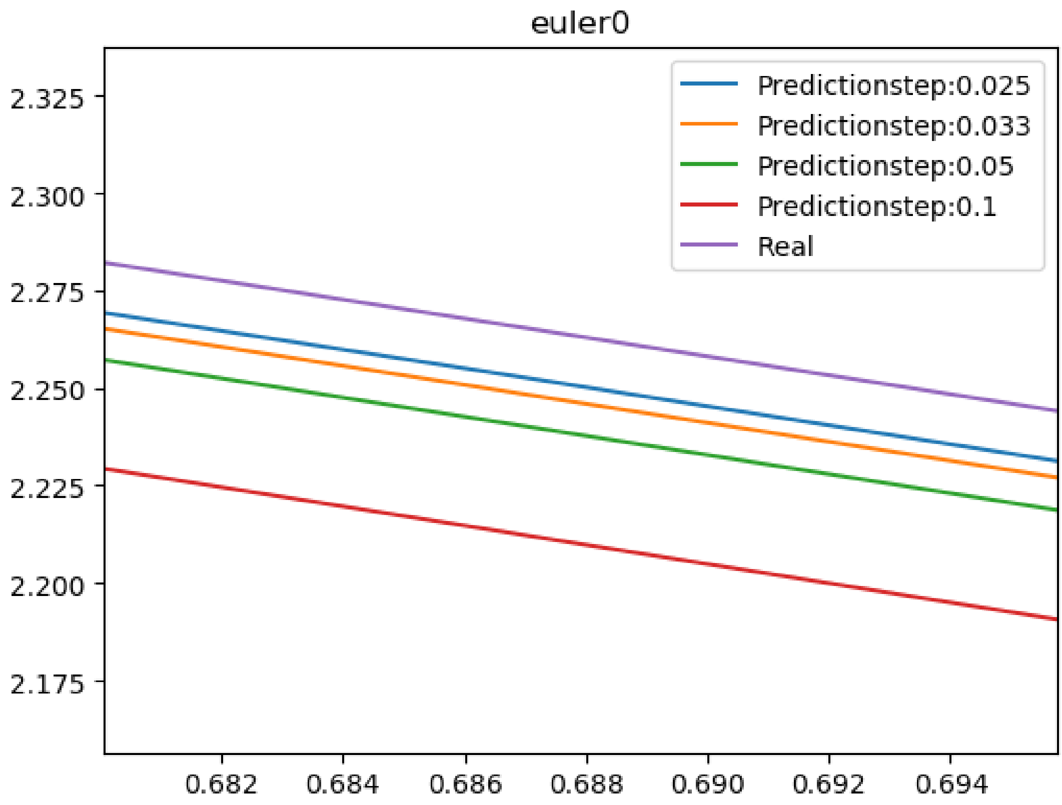
- **Example and Result**
  - **Improved Euler Method**



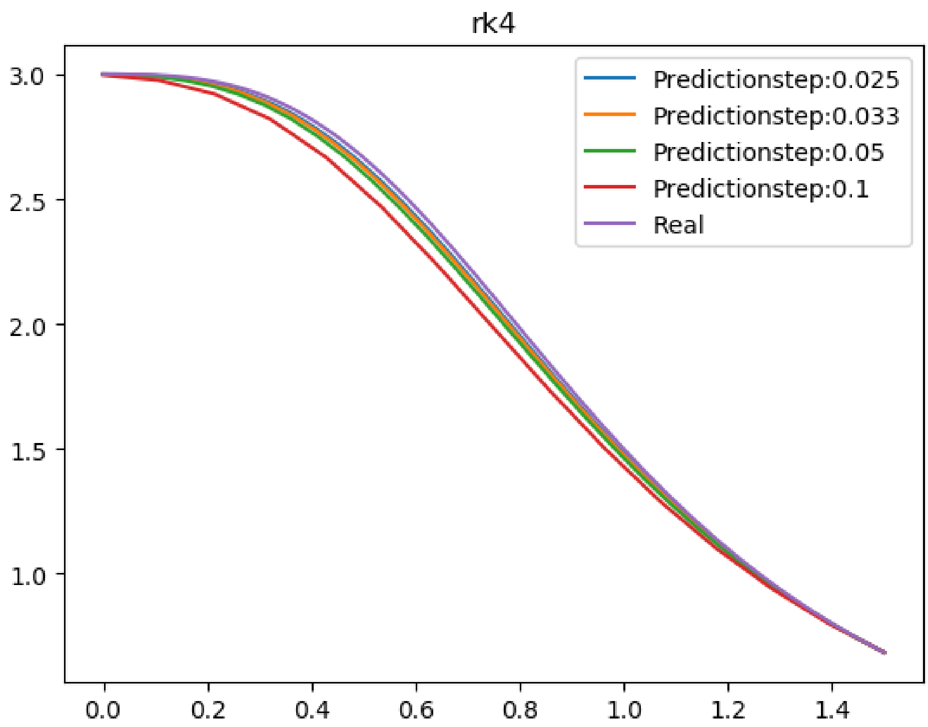


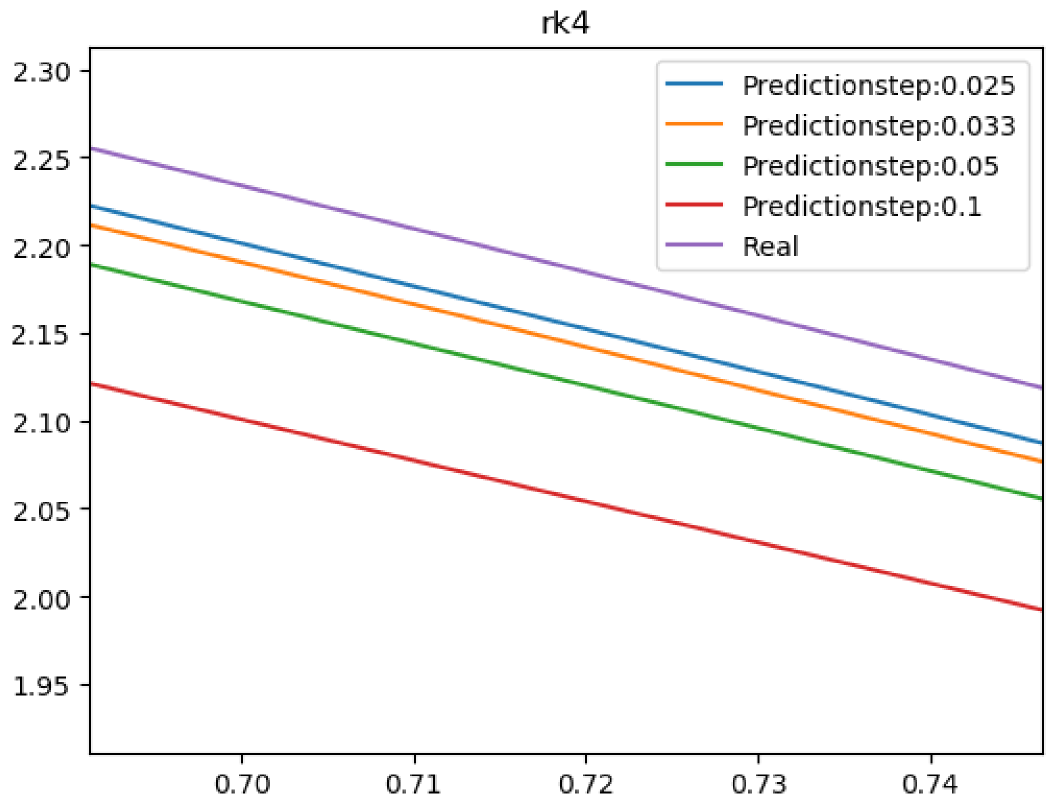
- **Explicit Euler Method**





- **4-order Runge-Kutta method**





- **Demo**
  Check the folder ”ODE” in the directory and follow the instruction to set up the matrices and vector