TASK1.

XIANG ZHANG

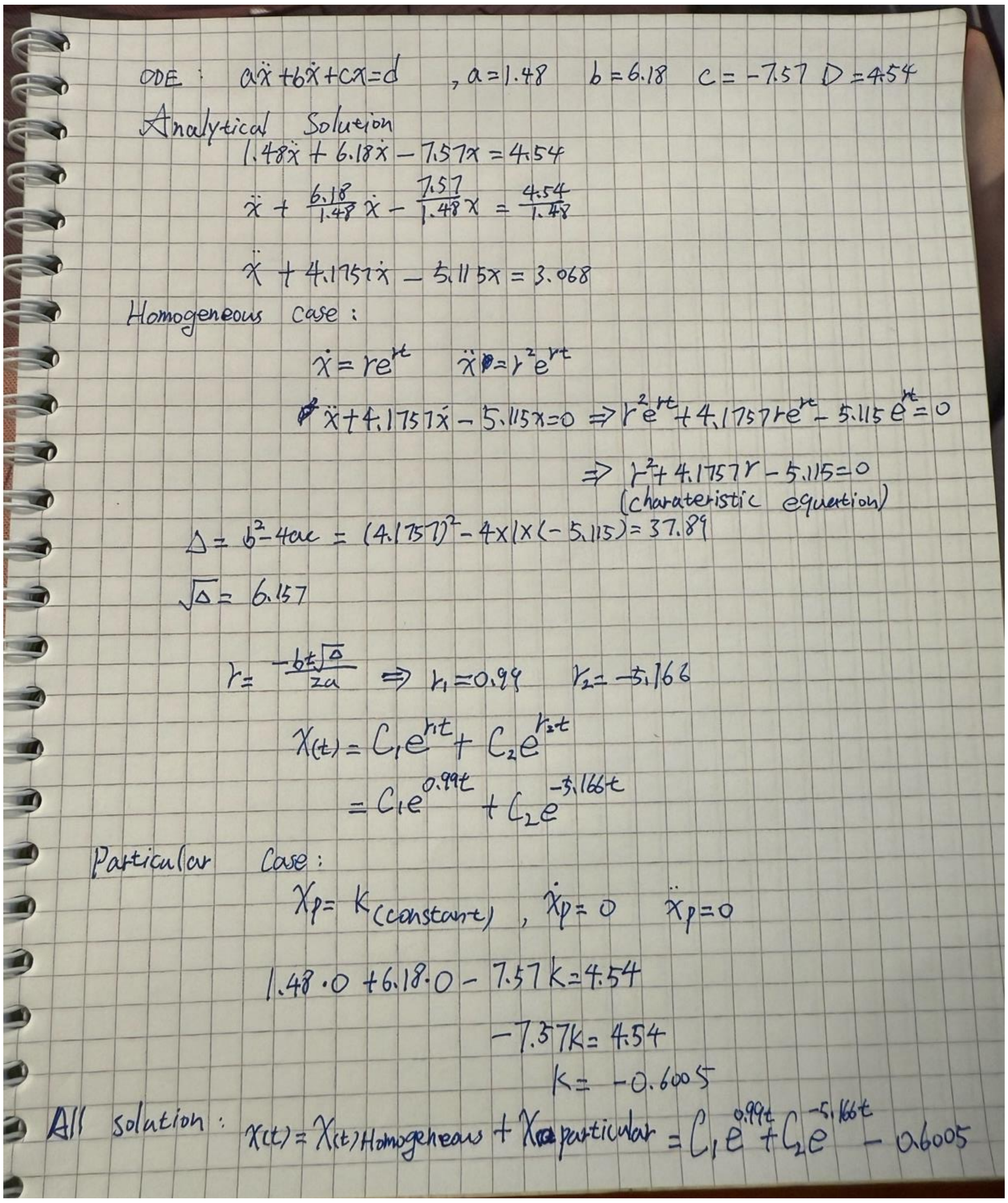
ISU:508513

1. solve analytically the ODE in the form of:

$$a \cdot \ddot{x} + b \cdot \dot{x} + c \cdot x = d$$

My coefficient: a=1.48, b=6.18, c=-7.57, d=4.54

Handwritten analysis:



ODE : $a\ddot{x} + b\dot{x} + cx = d$, $a = 1.48$ $b = 6.18$ $c = -7.57$ $D = 4.54$

Analytical Solution

$$1.48\ddot{x} + 6.18\dot{x} - 7.57x = 4.54$$

$$\ddot{x} + \frac{6.18}{1.48}\dot{x} - \frac{7.57}{1.48}x = \frac{4.54}{1.48}$$

$$\ddot{x} + 4.1757\dot{x} - 5.115x = 3.068$$

Homogeneous case :

$$\dot{x} = re^{rt} \qquad \ddot{x} = r^2 e^{rt}$$

$$\ddot{x} + 4.1757\dot{x} - 5.115x = 0 \Rightarrow r^2 e^{rt} + 4.1757 r e^{rt} - 5.115 e^{rt} = 0$$

$$\Rightarrow r^2 + 4.1757r - 5.115 = 0$$

(charateristic equation)

$$\Delta = b^2 - 4ac = (4.1757)^2 - 4 \times 1 \times (-5.115) = 37.89$$

$$\sqrt{\Delta} = 6.157$$

$$r = \frac{-b \pm \sqrt{\Delta}}{2a} \Rightarrow r_1 = 0.99 \quad r_2 = -5.166$$

$$X(t) = C_1 e^{r_1 t} + C_2 e^{r_2 t}$$

$$= C_1 e^{0.99t} + C_2 e^{-5.166t}$$

Particular Case :

$$X_p = k_{(constant)} \, , \quad \dot{x}_p = 0 \quad \ddot{x}_p = 0$$

$$1.48 \cdot 0 + 6.18 \cdot 0 - 7.57k = 4.54$$

$$-7.57k = 4.54$$

$$k = -0.6005$$

All solution : $x(t) = X(t)_{Homogeneous} + X_{particular} = C_1 e^{0.99t} + C_2 e^{-5.166t} - 0.6005$

Then we solve them with three integrators:
Explicit Euler, Implicit Euler, Rund-Kutta methods.

1. define system dynamics and input coeffiicients :

```python
def system_dynamics(x):
    """
    Dynamics for a*¨x + b*˙x + c*x = d
    where x = [position, velocity]
    """
    a = 1.48
    b = 6.18
    c = -7.57
    d = 4.54

    x1 = x[0]  # position
    x2 = x[1]  # velocity

    # Compute acceleration from the equation
    x2_dot = (d - b * x2 - c * x1) / a

    return np.array([x2, x2_dot])
```

2. Use these functions of correspond methods:
   The principle of those integrators methods:
   (1) Explicit Euler method (Forward Euler): According to the definition of the derivative, the slope at the current time step can be used to estimate the value at the next time step.

$$x_{n+1} = x_n + hf(t_n, x_n)$$

Advantages: Simplest algorithm structure, easy to implement; Low computational cost, fast execution speed;
Constraints: Poor stability, may diverge numerically on rigid or unstable systems; Requires a small step size "h" to obtain accurate results.

(2) Implicit Euler method (Backward Euler): Use the slope at the next time step to estimate the change in the current step size.

$$x_{n+1} = x_n + hf(t_{n+1}, x_{n+1})$$

Advantages: Strong numerical stability, performs well for rigid systems;
Can maintain convergence with large step sizes;
Constraints : Each step requires solving equations, resulting in a larger computational load than explicit methods;

(3) Runge - Kutta : Within a single integration step, the slope is calculated multiple times at different intermediate points (usually 4 times), and a weighted average is taken to obtain higher-order precision.

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Advantages:High accuracy (fourth-order accurate), error decreases rapidly with step size;Balances stability and accuracy;

Constraints:Calculates the function 4 times per step, taking more time than the explicit Euler method.

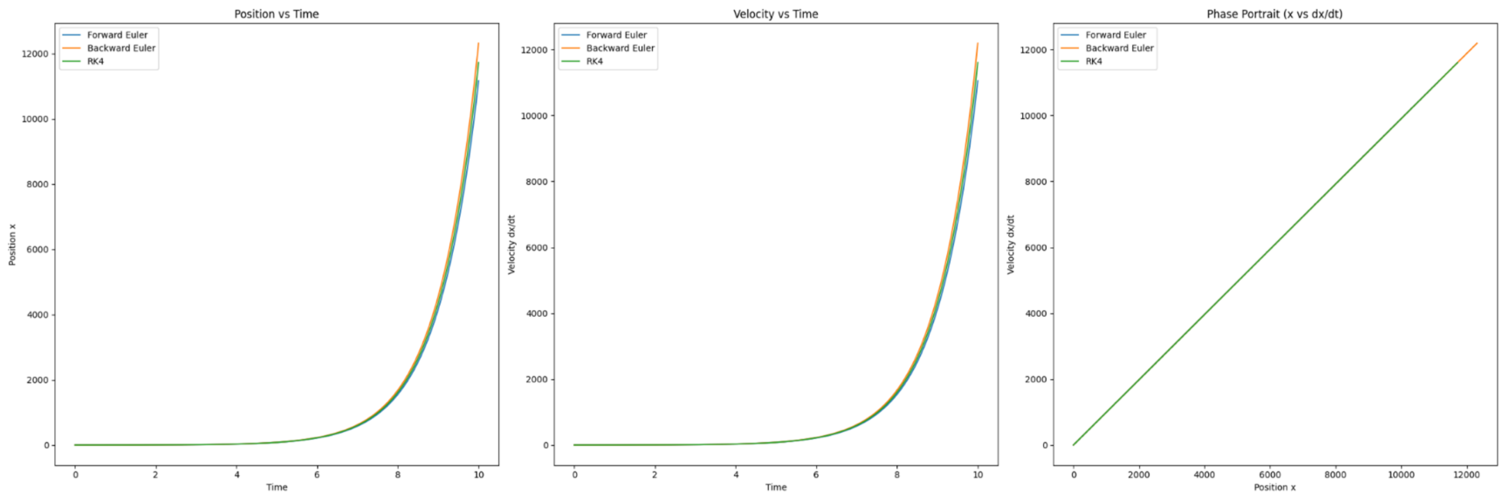
```
3. def forward_euler(fun, x0, Tf, h):
       """
       Explicit Euler integration method
       """
       t = np.arange(0, Tf + h, h)
       x_hist = np.zeros((len(x0), len(t)))
       x_hist[:, 0] = x0

       for k in range(len(t) - 1):
           x_hist[:, k + 1] = x_hist[:, k] + h * fun(x_hist[:, k])

       return x_hist, t


   def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
       """
       Implicit Euler integration method using fixed-point iteration
       """
       t = np.arange(0, Tf + h, h)
       x_hist = np.zeros((len(x0), len(t)))
       x_hist[:, 0] = x0

       for k in range(len(t) - 1):
           x_hist[:, k + 1] = x_hist[:, k]  # Initial guess

           for i in range(max_iter):
               x_next = x_hist[:, k] + h * fun(x_hist[:, k + 1])
               error = np.linalg.norm(x_next - x_hist[:, k + 1])
               x_hist[:, k + 1] = x_next

               if error < tol:
                   break

       return x_hist, t
```

```python
def runge_kutta4(fun, x0, Tf, h):
    """
    4th order Runge-Kutta integration method
    """
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        k1 = fun(x_hist[:, k])
        k2 = fun(x_hist[:, k] + 0.5 * h * k1)
        k3 = fun(x_hist[:, k] + 0.5 * h * k2)
        k4 = fun(x_hist[:, k] + h * k3)

        x_hist[:, k + 1] = x_hist[:, k] + (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)

    return x_hist, t
```

4. Test all integrators and plot result:

```python
5. x0 = np.array([0.1, 0.0])  # Initial state: [angle, angular_velocity]
   Tf = 10.0
   h = 0.01

   # Forward Euler
   x_fe, t_fe = forward_euler(system_dynamics, x0, Tf, h)

   # Backward Euler
   x_be, t_be = backward_euler(system_dynamics, x0, Tf, h)

   # Runge-Kutta 4
   x_rk4, t_rk4 = runge_kutta4(system_dynamics, x0, Tf, h)

   # Plot results
   plt.figure(figsize=(24, 8))

   plt.subplot(1, 3, 1)
   plt.plot(t_fe, x_fe[0, :], label='Forward Euler')
   plt.plot(t_be, x_be[0, :], label='Backward Euler')
   plt.plot(t_rk4, x_rk4[0, :], label='RK4')
   plt.xlabel('Time')
   plt.ylabel('Position x')
   plt.legend()
   plt.title('Position vs Time')

   plt.subplot(1, 3, 2)
   plt.plot(t_fe, x_fe[1, :], label='Forward Euler')
   plt.plot(t_be, x_be[1, :], label='Backward Euler')
   plt.plot(t_rk4, x_rk4[1, :], label='RK4')
   plt.xlabel('Time')
   plt.ylabel('Velocity dx/dt')
   plt.legend()
   plt.title('Velocity vs Time')

   plt.subplot(1, 3, 3)
   plt.plot(x_fe[0, :], x_fe[1, :], label='Forward Euler')
   plt.plot(x_be[0, :], x_be[1, :], label='Backward Euler')
   plt.plot(x_rk4[0, :], x_rk4[1, :], label='RK4')
   plt.xlabel('Position x')
   plt.ylabel('Velocity dx/dt')
   plt.legend()
   plt.title('Phase Portrait (x vs dx/dt)')

   plt.tight_layout()
   plt.show()
```

6. out put graph:



7. Conclusion

Comparison of analytical solutions and numerical results,

Analytical solusion: $x(t)=C_1e^{0.99t}+C_2e^{-5.166t}-0.6005$

Numerical results:

(1) Position vs. velocity curves

The exponential growth term '$e^{0.95t}$' dominates long-term behavior, exhibiting an exponential growth trend, with its position and speed increasing exponentially over time.

The constant term -0.6005 has only a slight effect in the initial stage; the effect is initially mild but then diverges rapidly.

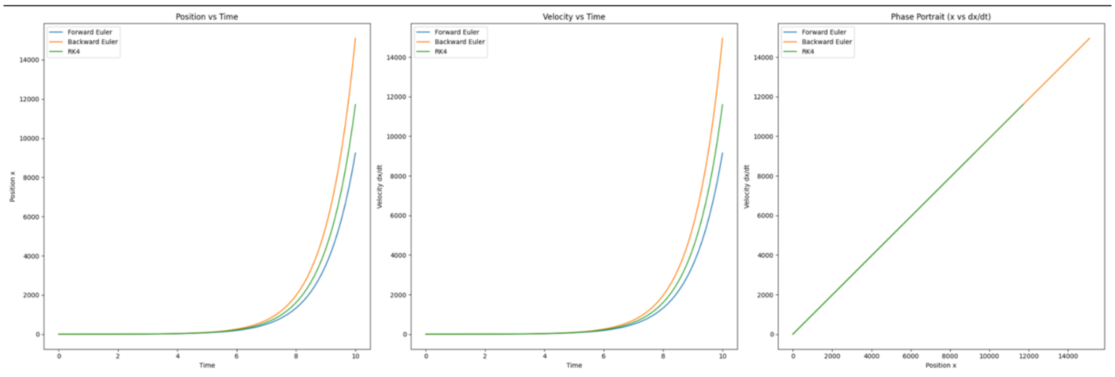
It increases exponentially over time ; There are no periodic or stable values; the three integration methods almost overlap indicating consistent integrators and high numerical accuracy.

(2) Phase diagram

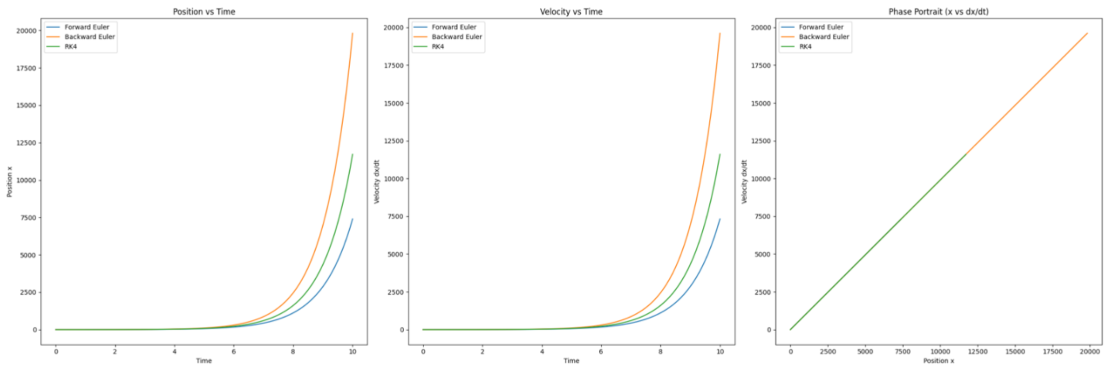
The trajectory is a nearly straight diagonal line; this indicates an almost linear positive correlation between velocity and position; the absence of a closed trajectory, suggests that the system is not in periodic motion, but rather undergoes continuous divergent growth.

8. Parameter Adjustment and Method Comparison

The three integration methods exhibit differences in numerical performance for divergent systems at the same time length h = 0.05.



When h=0.1:



By comparing the numerical integration results of the explicit Euler, implicit Euler, and fourth-order Runge-Kutta methods for divergent systems at different step sizes, it can be found that there are significant differences in stability and accuracy among the methods.

When the step size h = 0.01, the results of the three methods almost completely overlap, accurately reproducing the exponential divergence trend of the system.

When the step size increases to h = 0.05, slight differences begin to appear among the three methods. Although all maintain the correct divergence trend, the explicit Euler method deviates slightly in the later stages of the calculation, while the results of RK4 and the implicit Euler method are closer to the true solution.

When the step size is further increased to h = 0.1, the differences become significant. The explicit Euler method exhibits obvious numerical divergence and rapid error accumulation; the implicit Euler method remains stable but slightly underestimates the growth rate; RK4 falls between the two, balancing stability and accuracy.

In summary, the step size “h” has a crucial impact on the stability and accuracy of numerical integration: at smaller step sizes, the methods behave similarly; as the step size increases, explicit methods gradually become unstable, while implicit methods still maintain numerical stability. RK4 exhibits balanced performance at different step sizes and is suitable for situations requiring high accuracy but where the system is not rigid. For divergent or rigid systems, the implicit Euler method is more reliable; while in general dynamic problems, RK4 provides a good trade-off between accuracy and computational efficiency.