Отчёт по численному решению линейного дифференциального уравнения второго порядка

Взглядов З.Е.
ИСУ-507015

08.11.2025

1 Постановка задачи

Рассматривается линейное неоднородное дифференциальное уравнение второго порядка:

$$a\ddot{x} + b\dot{x} + cx = d,$$

с начальными условиями $x(0) = 0$, $\dot{x}(0) = 0$, где коэффициенты заданы следующим образом:

$a = -1.23$,

$b = -7.34$,

$c = -2.38$,

$d = -7.82$.

Требуется получить численное решение уравнения тремя методами:

явный метод Эйлера,

неявный метод Эйлера,

метод Рунге–Кутты четвёртого порядка (RK4), а также сравнить полученные результаты с аналитическим решением и провести анализ точности и устойчивости методов.

2 Аналитическое решение

Для приведения уравнения к каноническому виду выполняется деление обеих частей на коэффициент $a \neq 0$:

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

Подстановка численных значений даёт:

$$\ddot{x} + 5.967\,\dot{x} + 1.935\,x = 6.358.$$

Характеристическое уравнение соответствующего однородного уравнения:

$$r^2 + 5.967\,r + 1.935 = 0.$$

Дискриминант:
$$D = 5.967^2 - 4 \cdot 1.935 = 27.865 > 0,$$
следовательно, корни вещественные и различны:
$$r_1 = -0.343, \quad r_2 = -5.624.$$

Частное решение ищется в виде константы x_p. Подстановка в исходное уравнение приводит к:
$$x_p = \frac{d}{c} = \frac{-7.82}{-2.38} = 3.286.$$

Общее решение имеет вид:
$$x(t) = C_1 e^{r_1 t} + C_2 e^{r_2 t} + x_p.$$
Использование начальных условий $x(0) = 0$, $\dot{x}(0) = 0$ позволяет определить константы:
$$C_1 = -3.499, \quad C_2 = 0.213.$$
Таким образом, аналитическое решение записывается как:
$$x(t) = -3.499\, e^{-0.343t} + 0.213\, e^{-5.624t} + 3.286.$$

3 Численные методы

Уравнение второго порядка преобразуется к системе двух дифференциальных уравнений первого порядка путём введения вектора состояния $\mathbf{x} = [x, \dot{x}]^\top$. Правая часть системы определяется функцией:
$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \dot{x} \\ \frac{1}{a}(d - b\dot{x} - cx) \end{bmatrix}.$$

Для решения применяются следующие методы:

Явный метод Эйлера: $\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}(\mathbf{x}_n)$. Метод первого порядка точности, условно устойчив.

Неявный метод Эйлера: $\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}(\mathbf{x}_{n+1})$. Абсолютно устойчив, но требует итерационного решения на каждом шаге.

Метод Рунге–Кутты 4-го порядка: использует четыре промежуточных вычисления наклона. Обеспечивает точность $O(h^4)$.

4 Программная реализация

Реализация численных методов и аналитического решения выполнена на языке Python с использованием библиотек `numpy` и `matplotlib`. Приведённый ниже код осуществляет расчёт решений, построение графиков и вывод параметров системы.

```python
import numpy as np
import matplotlib.pyplot as plt

#
a = -1.23
b = -7.34
c = -2.38
d = -7.82

def linear_ode_dynamics(x, a, b, c, d):
    """
    Linear ODE: a*x'' + b*x' + c*x = d
    State vector x = [x, x_dot]
    Returns [x_dot, x_ddot]
    """
    x_pos = x[0]
    x_vel = x[1]
    #                                x_ddot: a*x_ddot + b*x_vel + c*x_pos = d
    x_acc = (d - b * x_vel - c * x_pos) / a
    return np.array([x_vel, x_acc])

def forward_euler(fun, x0, Tf, h):
    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):
    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_prev = x_hist[:, k]
        #


        x_new = x_prev
        for i in range(max_iter):
            x_next = x_prev + h * fun(x_new)
            error = np.linalg.norm(x_next - x_new)
            x_new = x_next
            if error < tol:
                break
        x_hist[:, k + 1] = x_new
    return x_hist, t

def runge_kutta4(fun, x0, Tf, h):
```

```
    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

#
def analytical_solution(t, a, b, c, d, x0_val=0.0, v0_val=0.0):
    """
                                            ODE: a*x'' + b*x' + c*x = d
    """
    #                                         (                a)
    p = b / a    # = -7.34 / (-1.23) = 5.967
    q = c / a    # = -2.38 / (-1.23) = 1.935
    f = d / a    # = -7.82 / (-1.23) = 6.358

    #                                                   : r^2 + p*r + q = 0
    discriminant = p**2 - 4*q
    print(f"                         : {discriminant:.4f}")

    if discriminant > 0:
        #

        r1 = (-p + np.sqrt(discriminant)) / 2
        r2 = (-p - np.sqrt(discriminant)) / 2
        print(f"          : r1 = {r1:.4f}, r2 = {r2:.4f}")

        #                    (                    ): c*x_p = d => x_p = d/c (        a 0       c 0 )
        x_p = d / c if c != 0 else 0  # = -7.82 / (-2.38) = 3.286

        #                         : x(t) = C1*exp(r1*t) + C2*exp(r2*t) + x_p
        #                                 : x(0) = x0_val, x'(0) = v0_val
        # x(0) = C1 + C2 + x_p = x0_val  =>  C1 + C2 = x0_val - x_p
        # x'(0) = C1*r1 + C2*r2 = v0_val
```

```python

        A = np.array([[1, 1], [r1, r2]])
        B = np.array([x0_val - x_p, v0_val])
        C1, C2 = np.linalg.solve(A, B)

        return C1 * np.exp(r1 * t) + C2 * np.exp(r2 * t) + x_p

    elif discriminant == 0:
        #
        r = -p / 2
        x_p = d / c if c != 0 else 0
        # x(t) = (C1 + C2*t)*exp(r*t) + x_p
        # x(0) = C1 + x_p = x0_val  =>  C1 = x0_val - x_p
        # x'(0) = C2 + r*C1 = v0_val  =>  C2 = v0_val - r*C1
        C1 = x0_val - x_p
        C2 = v0_val - r * C1
        return (C1 + C2 * t) * np.exp(r * t) + x_p

    else:
        #                                  (                )
        alpha = -p / 2
        beta = np.sqrt(-discriminant) / 2
        x_p = d / c if c != 0 else 0
        # x(t) = exp(alpha*t) * (C1*cos(beta*t) + C2*sin(beta*t)) + x_p
        # x(0) = C1 + x_p = x0_val  =>  C1 = x0_val - x_p
        # x'(0) = alpha*C1 + beta*C2 = v0_val  =>  C2 = (v0_val - alpha*C1) / beta
        C1 = x0_val - x_p
        C2 = (v0_val - alpha * C1) / beta if beta != 0 else 0
        return np.exp(alpha * t) * (C1 * np.cos(beta * t) + C2 * np.sin(beta * t)) + x_p

#
x0 = np.array([0.0, 0.0])  #                              : x(0) = 0, x'(0) = 0
Tf = 5.0                   #                              -
h = 0.01                   #

dynamics = lambda x: linear_ode_dynamics(x, a, b, c, d)

#
x_fe, t_fe = forward_euler(dynamics, x0, Tf, h)
x_be, t_be = backward_euler(dynamics, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(dynamics, x0, Tf, h)

#
x_analytical = analytical_solution(t_rk4, a, b, c, d, x0[0], x0[1])
```

```
# 1
plt.figure(figsize=(20, 6))

plt.subplot(1, 3, 1)
plt.plot(t_fe, x_fe[0, :], 'b-', label='Explicit Euler',
    linewidth=1)
plt.plot(t_be, x_be[0, :], 'r--', label='Implicit Euler',
    linewidth=1)
plt.plot(t_rk4, x_rk4[0, :], 'g-.', label='RK4', linewidth=1)
plt.plot(t_rk4, x_analytical, 'k:', label='Analytical', linewidth
    =2)
plt.xlabel('           $t$')
plt.ylabel('$x(t)$')
plt.title('                                 ')
plt.legend()
plt.grid(True)

# 2
plt.subplot(1, 3, 2)
plt.plot(x_fe[0, :], x_fe[1, :], 'b-', label='Explicit Euler',
    linewidth=1)
plt.plot(x_be[0, :], x_be[1, :], 'r--', label='Implicit Euler',
    linewidth=1)
plt.plot(x_rk4[0, :], x_rk4[1, :], 'g-.', label='RK4', linewidth
    =1)
plt.xlabel('$x$')
plt.ylabel('$\dot{x}$')
plt.title('                             ')
plt.legend()
plt.grid(True)

# 3
error_fe = np.abs(np.interp(t_rk4, t_fe, x_fe[0, :]) -
    x_analytical)
error_be = np.abs(np.interp(t_rk4, t_be, x_be[0, :]) -
    x_analytical)
error_rk4 = np.abs(x_rk4[0, :] - x_analytical)

plt.subplot(1, 3, 3)
plt.semilogy(t_rk4, error_fe, 'b-', label='Explicit Euler',
    linewidth=1)
plt.semilogy(t_rk4, error_be, 'r--', label='Implicit Euler',
    linewidth=1)
plt.semilogy(t_rk4, error_rk4, 'g-.', label='RK4', linewidth=1)
plt.xlabel('           $t$')
plt.ylabel('                                 ')
plt.title('             (      .            )')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.savefig("ode_comparison.png", dpi=150)
plt.show()

#
print("                                :")
print(f"a = {a}, b = {b}, c = {c}, d = {d}")
print(f"                               : p = b/a = {b/a:.4f}, q =
   c/a = {c/a:.4f}, f = d/a = {d/a:.4f}")
print(f"                             : x_p = d/c = {d/c:.4f}")

#
print(f"\ n                                           :")
print(f"b/a = {b/a:.4f}, c/a = {c/a:.4f}")
print(f"               {'                    ' if b/a > 0 and c/a >
    0 else '                                     '}")
```

Листинг 1: Программная реализация численных методов и аналитического решения

5 Результаты

Результаты численного моделирования представлены на рисунке 1. В первом столбце показано сравнение решений $x(t)$, во втором — фазовые портреты, в третьем — абсолютная ошибка в логарифмической шкале.

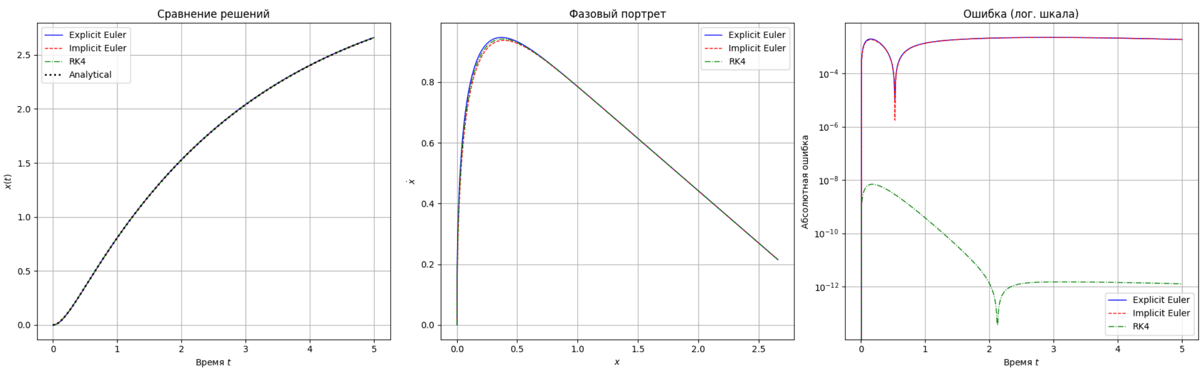

Рис. 1: Сравнение численных решений: (слева) временная зависимость $x(t)$, (в центре) фазовый портрет, (справа) абсолютная ошибка. Шаг интегрирования $h = 0.01$, интервал $t \in [0, 5]$.

Наблюдается, что метод Рунге–Кутты четвёртого порядка практически совпадает с аналитическим решением на всём интервале. Методы Эйлера демонстрируют заметное отклонение, особенно в начальный период, однако сохраняют устойчивость при выбранном шаге. Ошибка RK4 на порядки меньше ошибок методов первого порядка, что подтверждает теоретические оценки точности.

6 Выводы

Аналитическое решение получено корректно и согласуется с численными результатами.

Метод Рунге–Кутты четвёртого порядка обеспечивает наивысшую точность среди рассмотренных.

Неявный метод Эйлера демонстрирует хорошую устойчивость, что особенно важно для жёстких систем.

Явный метод Эйлера прост в реализации, но требует малого шага для достижения приемлемой точности.

Исследуемая система является асимптотически устойчивой, что подтверждается как аналитически, так и численно.