Смирнов А.В. №ису:507243

ЛАБОРАТОРНАЯ РАБОТА №1

$$ax'' + bx' + cx = d\ ,$$

где a=6,44; b=3,28; c=8,64; d=5,04.

С помощью библиотеки sympy в python найдем аналитическое решение данного ДУ (рис. 1).

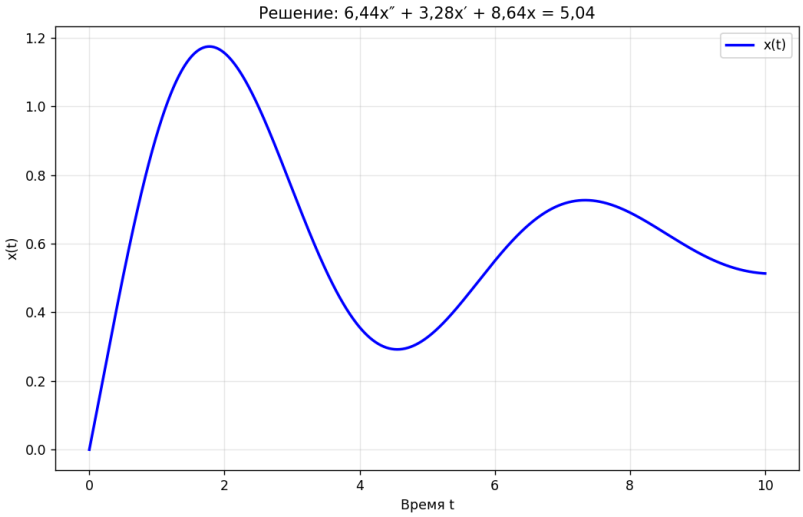


Рисунок 1 «Аналитическое решение ДУ»

Решим данное ДУ тремя методами явным и неявным Эйлером и методом Рунге-Кутта (рис. 2).

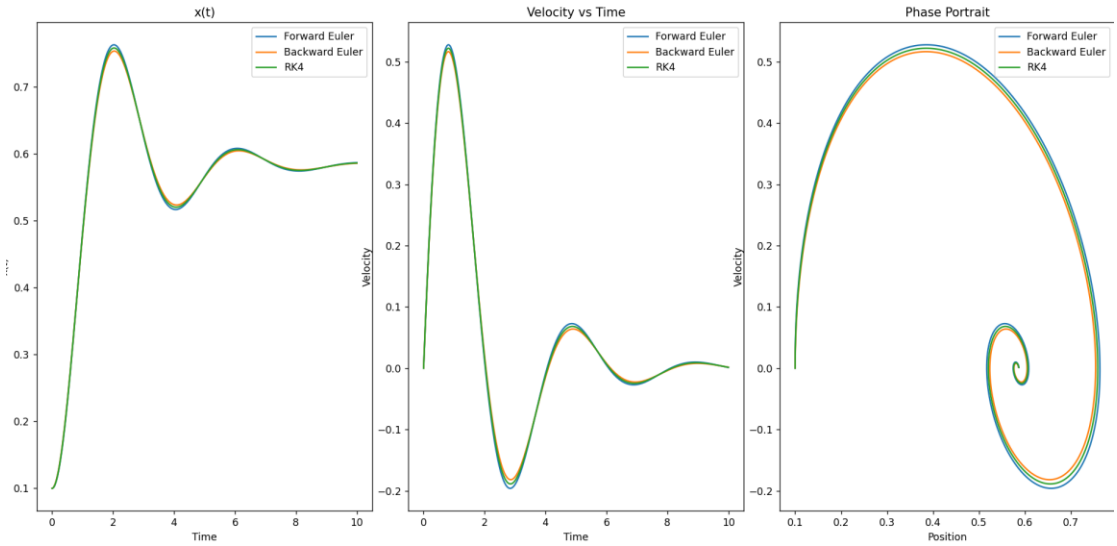


Рисунок 2 «Решение ДУ тремя методами: явный и не явный Эйлер и метод Рунге-Кутта»

Выводы:

1. Аналитическое решение ДУ совпадает с решением методами Эйлера и Рунге-Кутта (графики на рисунке 1 и график x(t) на рисунке 2). Установившееся значение составляет около 0,6 (если увеличить время симулирования).

2. Система с данными коэффициентами является устойчивой.

3. Метод Рунге-Кутта является наиболее точным на начальном этапе симуляции (переходном процессе), так как в методе Эйлера ошибка зависит от шага и уменьшается линейно. В установившемся режиме данные методы решения показывают практически одинаковый результат.

Скетч кода для аналитического решения ДУ:

```
import sympy as sp
from sympy import symbols, Function, dsolve, Eq

t = symbols('t')
x = Function('x')(t)


#Коэфициенты уравнения
a_val, b_val, c_val, d_val = 6.44, 3.28, 8.64, 5.04

# Подставляем численные значения
equation_num = Eq(a_val * x.diff(t, 2) + b_val * x.diff(t) + c_val * x, d_val)
solution_num = dsolve(equation_num, ics={x.subs(t, 0): 0, x.diff(t).subs(t, 0): 1})


# Визуализация решения
import numpy as np
import matplotlib.pyplot as plt


x_func = sp.lambdify(t, solution_num.rhs, 'numpy')

t_vals = np.linspace(0, 10, 1000)
x_vals = x_func(t_vals)

plt.figure(figsize=(10, 6))
plt.plot(t_vals, x_vals, 'b-', linewidth=2, label='x(t)')
plt.xlabel('Время t')
plt.ylabel('x(t)')
plt.title('Решение: 6,44x″ + 3,28x′ + 8,64x = 5,04')
plt.grid(True, alpha=0.3)

plt.legend()
plt.show()
```

Скетч кода для решения ДУ методами Эйлера (явный и неявный) и методом Рунге-Кутта:

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

def pendulum_dynamics(x):
    """
    Pendulum dynamics: ddx = -(1/a) * (b*dx+c*x-D)
    State vector x = [x, dx]
    """
    a=6.44
    b=3.28
    c=8.64
    d=5.04

    theta = x[0]
    theta_dot = x[1]

    theta_ddot = (-1/b)*(b*theta_dot+c*theta-d)

    return np.array([theta_dot, theta_ddot])

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

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

# Test all integrators
x0 = np.array([0.1, 0.0])  # Initial state: [dx, ddx]
Tf = 10.0
h = 0.01

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

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

# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(pendulum_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('x(t)')
plt.legend()
plt.title('x(t)')

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')
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')
plt.ylabel('Velocity')
plt.legend()
plt.title('Phase Portrait')

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