Смирнов А.В. №ису:507243

ЛАБОРАТОРНАЯ РАБОТА №2

ВАРИАНТ №2

Для данной системы масса-пружина-демпфер кинетическая энергия $K = \frac{1}{2}mx'$, а потенциальная $P = \frac{1}{2}kx^2$. На рисунке 1 приведён вывод уравнения модели системы.

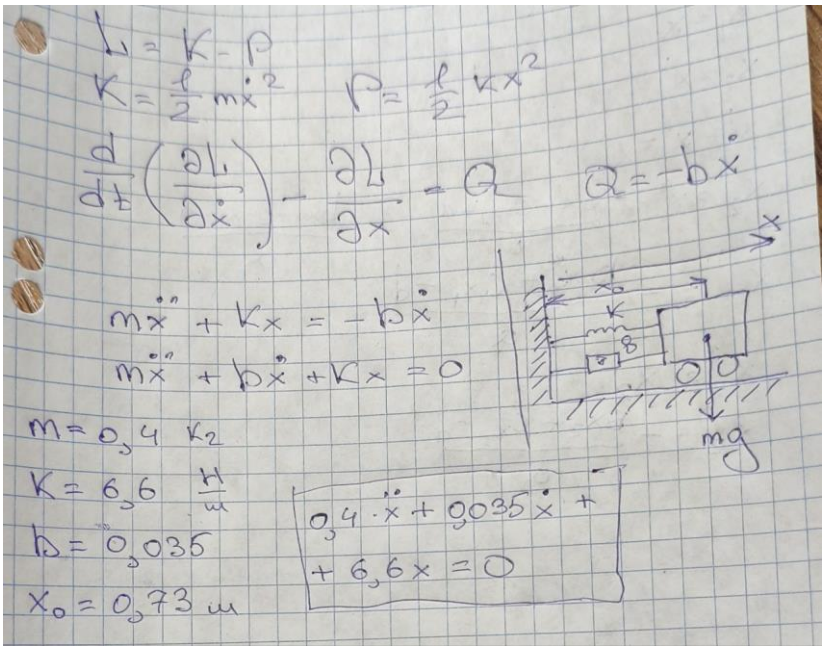

Рисунок 1 «Вывод уравнения модели системы масса-пружина-демпфер»

Для моделирования выразим $x'' = \frac{-1}{m} * (bx' + kx)$, где m=0,4 кг; b=0,035; k=6,6$\frac{\text{Н}}{\text{м}}$; а начальное положение системы x_0=0,73м.

Подставим данные значения в уравнение и получим:

$$x'' = \frac{-1}{0{,}4} * (0{,}035x' + 6{,}6x)$$

Построим модель в MatLab Simulink (рис. 2) и зададим начальное условие для положения модели (рис. 3).

Результаты моделирования в Simulink приведены на рисунке 4.

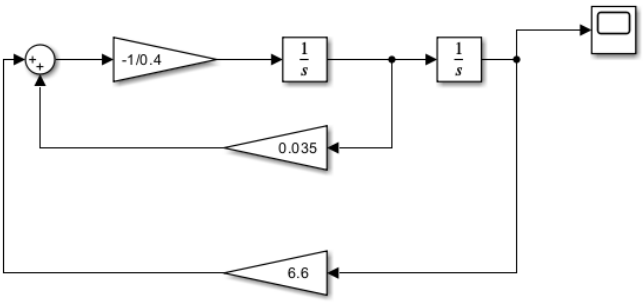

Рисунок 2 «Модель в Simulink»

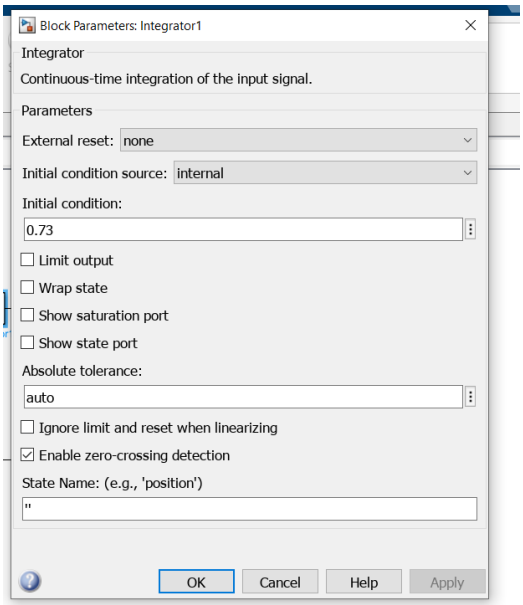

Рисунок 3 «Задача начальных условий (x_0)»

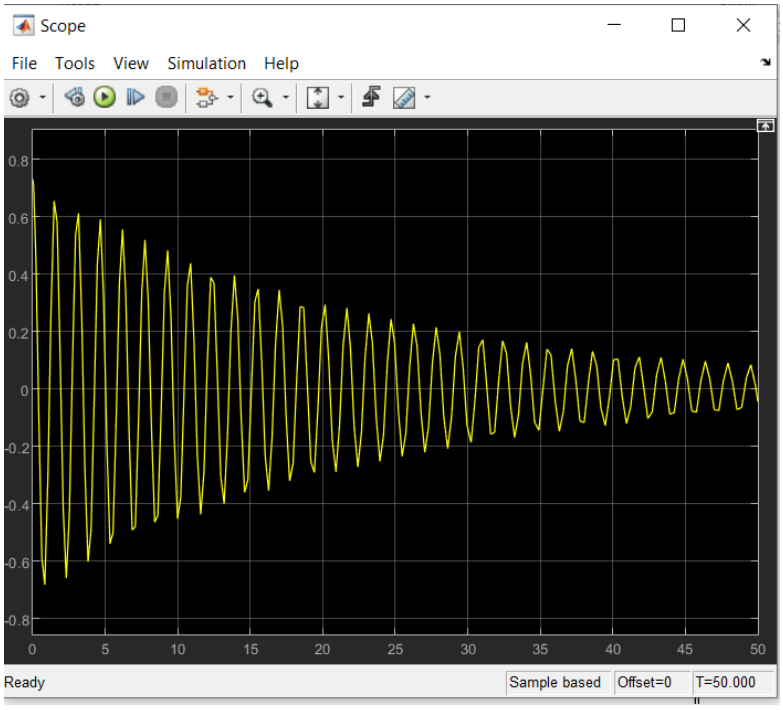

Рисунок 4 «Результат моделирования в Simulink»

Также с помощью библиотеки sympy в python было проведено моделирование данной системы (аналитическое решение) (рис. 5).

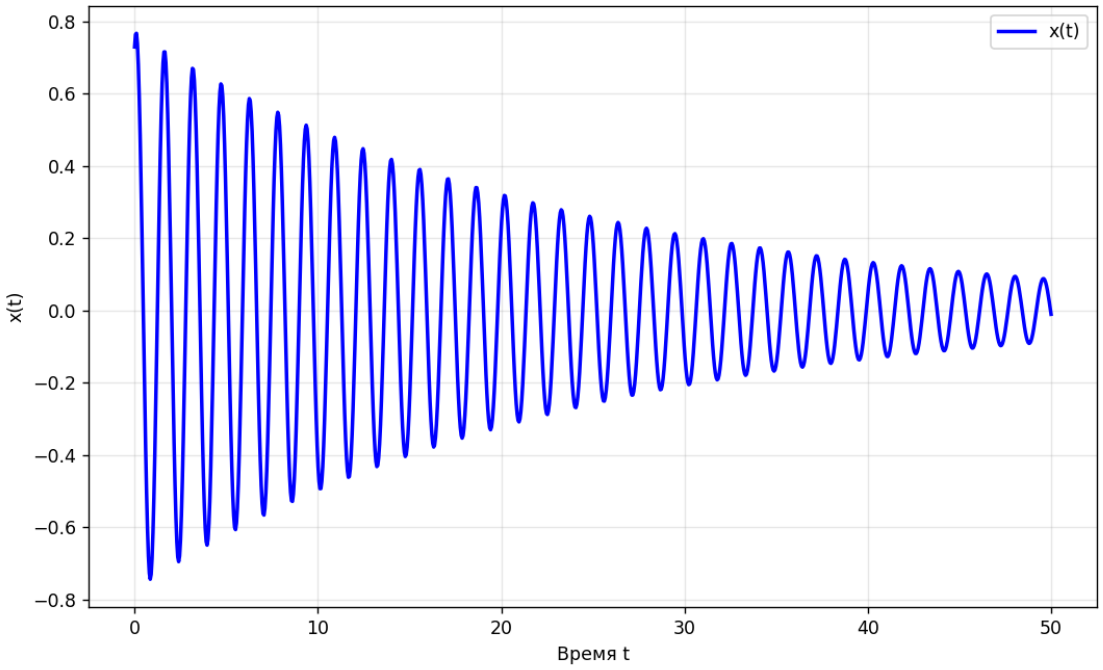

Рисунок 5 «Моделирование системы в Python»

Сравнивая рисунки 4 и 5, можно сделать вывод, что результат моделирования в Python и MatLab Simulink совпадают.

Найдём решение данной задачи тремя методами: явным и неявным Эйлером, а также методом Рунге-Кутта (рис.6).

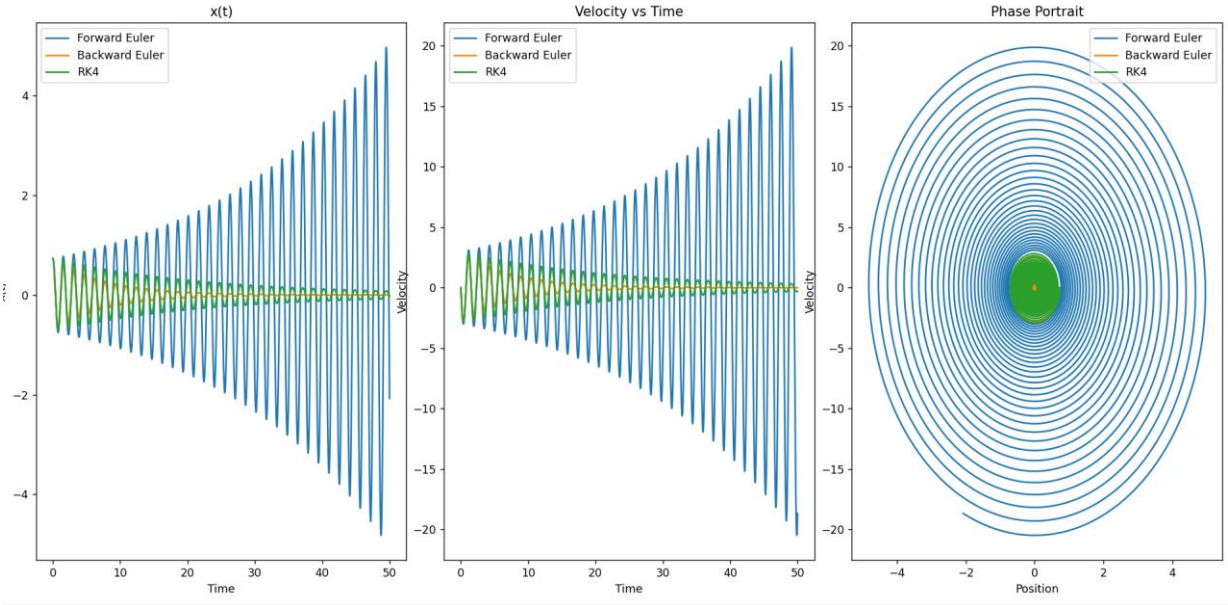

Рисунок 6 «Решение данной задачи явным и неявным Эйлером и методом Рунге-Кутта»

Выводы:

1. Метод явного Эйлера не справляется с решением данной задачи, так как для описания метода используется только текущее состояние системы.

2. Метод обратного Эйлера значительно лучше справляется с решением данной задачи, но из-за того, что ошибка уменьшается линейно, колебания примерно через 35с затухают.

3. Метод Рунге-Кутта совпадает с аналитическим решением (моделированием в Python и MatLab Simulink) и наиболее точно отражает процессы в масса-пружинно-демпферной системе.

Скетч кода для моделирования системы в Python (аналитическое решение):

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

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

a_val, b_val, c_val, d_val = 0.4, 0.035, 6.6, 0

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): 0.73, 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, 50, 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('Моделирование масса-пружина-демпферная система')
plt.grid(True, alpha=0.3)

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

Скетч кода для решения задачи 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/m) * (b*dx+k*x)
    State vector x = [x, dx]
    """
    m=0.4
    k=6.6
    b=0.035


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

    theta_ddot = (-1/m)*(b*theta_dot+k*theta)

    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([0.73, 0.0])  # Initial state: [dx, ddx]
Tf = 50.0
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()
```