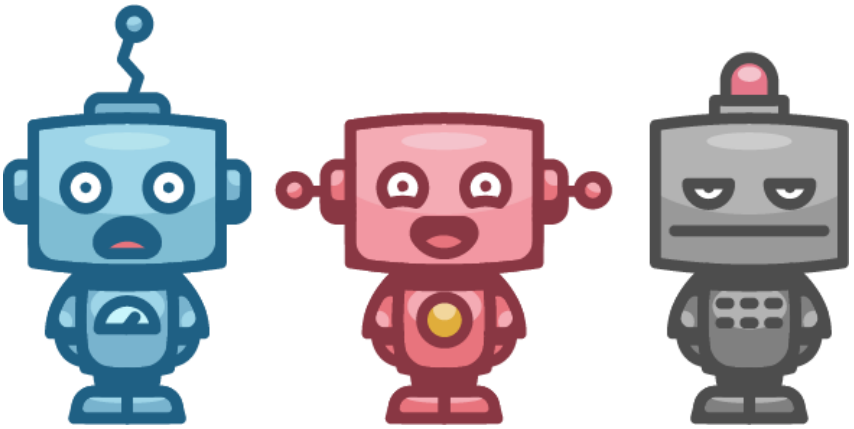
Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ–ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙУНИВЕРСИТЕТ ИТМО

ФСУиР

ЛАОРАТОРНАЯ РАБОТА №2:

Вариант №2



Выполнил: Гусаров С.А.
Группа: R4133c

г. Санкт–Петербург, 2025

ОСНОВНАЯ ЧАСТЬ

1 Аналитическое решение

Рассмотрим систему «масса-пружина-демпфер» (рисунок 1)

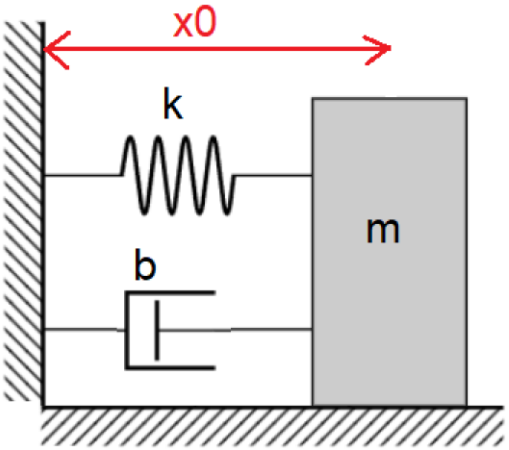


Рисунок 1 – Исследуемая система

Определим Лагранжиан системы:

$$\mathcal{L} = K - P, \tag{1.1}$$

где $K = 0.5m\dot{x}^2 -$ кинетическая энергия системы, $P = mgx + 0.5kx^2 -$ потенциальная энергия системы.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) - \frac{\partial \mathcal{L}}{\partial x} = Q, \qquad Q = -b\dot{x} \tag{1.2}$$

$$\ddot{x} + \frac{b}{m}\dot{x} + \frac{k}{m}x = -g \tag{1.3}$$

Частное решение неоднородного уравнения:

$$x_p(t) = \frac{d}{c} = -\frac{mg}{k} = -1.308$$

Общее решение:

$$\begin{cases} \lambda_1 = -0.03 + 2.75i \\ \lambda_2 = -0.03 - 2.75i \end{cases}$$

Тогда:

$$x(t) = c_1 e^{-0.03t}\sin(2.75t) + c_2 e^{-0.03t}\cos(2.75t) - 1.308 \tag{1.4}$$

Так как $x(0) = 0.96,\ \dot{x}(0) = 0$, тогда:

$$\begin{cases} c_2 - 1.308 = 0.96 \\ 2.74c_1 - 0.03c_2 = 0 \end{cases} \Leftrightarrow \begin{cases} c_1 = 0.025 \\ c_2 = 2.268 \end{cases} \quad (1.5)$$

Итого:

$$x(t) = 0.025e^{-0.03t}\sin(2.75t) + 2.268e^{-0.03t}\cos(2.75t) - 1.308 \quad (1.6)$$

2 Интеграторы

Сравним полученное аналитическое решение с решениями с помощью метода Рунге-Кутта. Данные полученные в ходе сравнения приведем на рисунке 2.

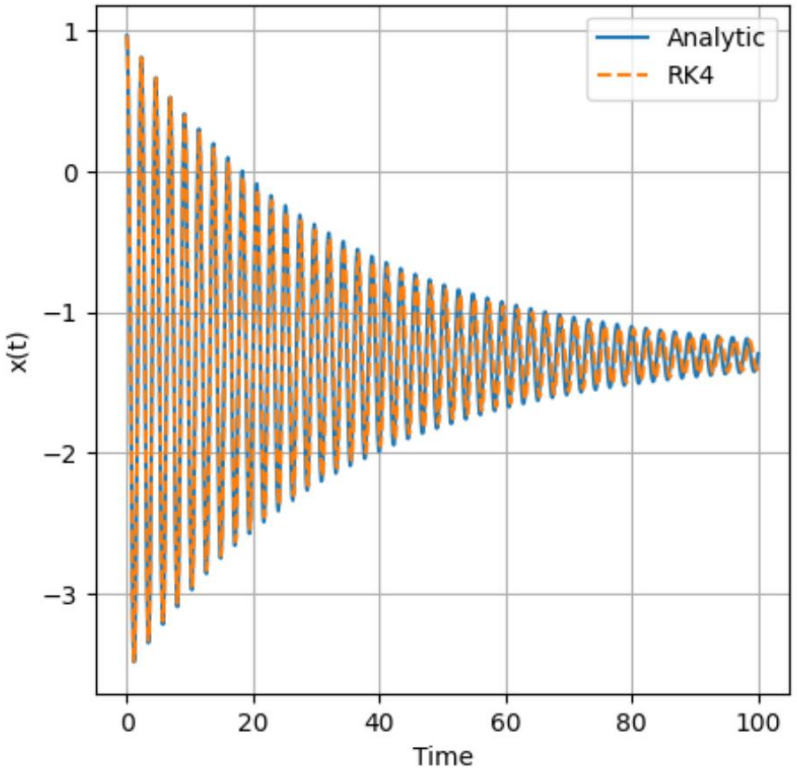


Рисунок 2 – Сравнительное моделирование при $h = 0.1$

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы мы составили дифференциальное уравнение для системы 2-ого порядка «масса-демпфер-пружина» и рассмотрели его аналитическое решение. Также сравнили его с решением, полученными с помощью метода Рунге-Кутта.

Стоит отметить, полученные результаты совпали, а также с «физической» точки зрения логика не нарушена, так как система получилось с затухающими колебаниями из-за наличия вязкого трения (демпфера).

Лабораторную работу считаю выполненной полностью.

Библиотеки

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

Исходные данные

```
m = 0.4
k = 3
bb = 0.025
g = 9.81
x0 = 0.96
```

Коэффициенты дифференциального уравнения

```
a = 1
b = bb/m
c = k/m
d = -g
xp = d/c
```

Решение однородного уравнения:

```
D = b**2 - 4*a*c
sqrt_D = 5.5j
x1 = (-b + sqrt_D)/(2*a)
x2 = (-b - sqrt_D)/(2*a)
print(x1, x2)
```

```
(-0.03125+2.75j) (-0.03125-2.75j)
```

Определение c1 и c2:

```
c2 = x0 - xp
c1 = 0.03*c2/2.75
print(c1,c2)
```

```
0.024741818181818177 2.268
```

Аналитическое решение

```
def dynamics(x):
    """
    Pendulum dynamics:
    x' = x'
    x'' = -(b/m)x' -(k/m)x - g
    State vector x = [x, x']
    """
    x1 = x[0]
    x2 = x[1]

    dx1 = x2
    dx2 = -(bb/m)*x2 - (k/m)*x1 - g

    return np.array([dx1, dx2])

def analytic(t):
  return c1*np.exp(-0.03*t)*np.sin(2.75*t) + c2*np.exp(-0.03*t)*np.cos(2.75*t) + xp

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_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):
    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
```

Визуализация

```
# Test all integrators
x0 = np.array([0.96, 0.0])
Tf = 100.0
h = 0.1

# Forward Euler
x_fe, t_fe = forward_euler(dynamics, x0, Tf, h)
# Backward Euler
x_be, t_be = backward_euler(dynamics, x0, Tf, h)
# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(dynamics, x0, Tf, h)

# Plot results
plt.figure(figsize=(5, 5))
plt.plot(t_fe, analytic(t_fe), label='Analytic')
#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.grid(True)
plt.show()
```