Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Отчет по практической работе № 1
по дисциплине
“Имитационное моделирование робототехнических систем”

Студент: Хомяков Д.А. (506873)
Группа: R4133c
Преподаватель: Ракшин Е.А (373529)

Санкт-Петербург
2025

1. Цель работы

В данной работе предлагается провести проверку решения дифференциального уравнения (ODE) с использованием трёх численных методов: метода явного Эйлера, метода неявного Эйлера и метода Рунге–Кутты. Кроме того, необходимо построить графики полученных результатов и сравнить их с аналитическим решением уравнения.

2. Решение ОДУ

В рамках данной работы предлагалось рассмотреть определённое дифференциальное уравнение (ОДУ), параметры которого задаются в соответствии с вариантом, указанным в предоставленной таблице.

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

$$a = 9.87\ b = 4.87\ c = 5.51\ d = -2.41$$

Приведём ОДУ к стандартному виду относительно $\ddot{x}$:

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

В процессе выполнения задания был получен исходный код, реализующий все три упомянутых ранее метода, поэтому их подробное описание не приводится. Для получения аналитического решения была создана схема в среде Simulink, представленная на рисунке 1.

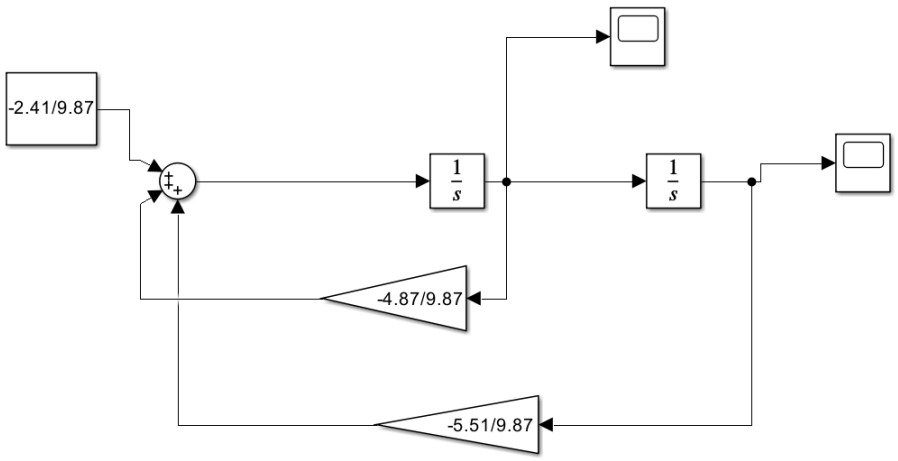

Рисунок 1. Схема ОДУ в Simulink

3. Полученные результаты

Результаты по завершению работы программы (приложение № 1) представлены на рисунках 2-4.

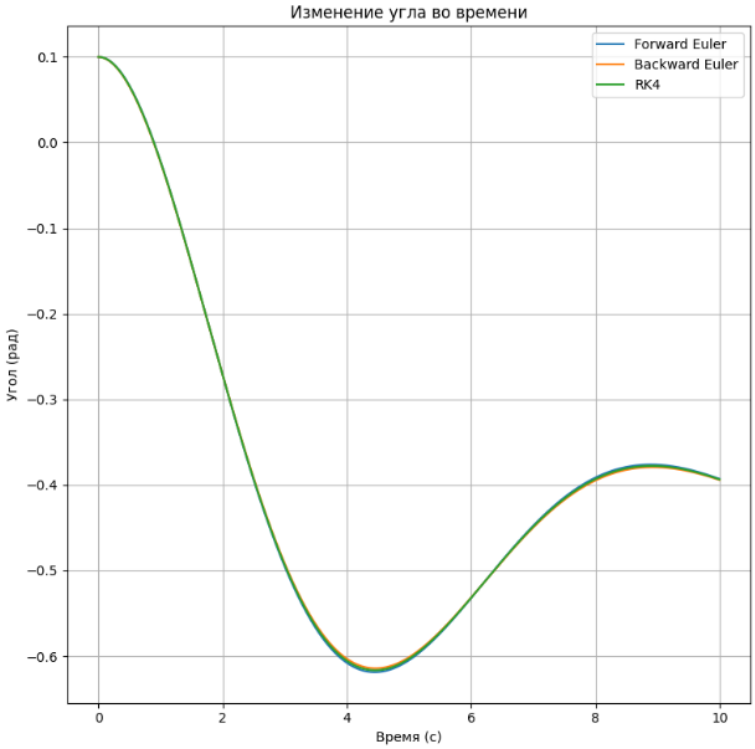

Рисунок 2. График положения после выполнения кода

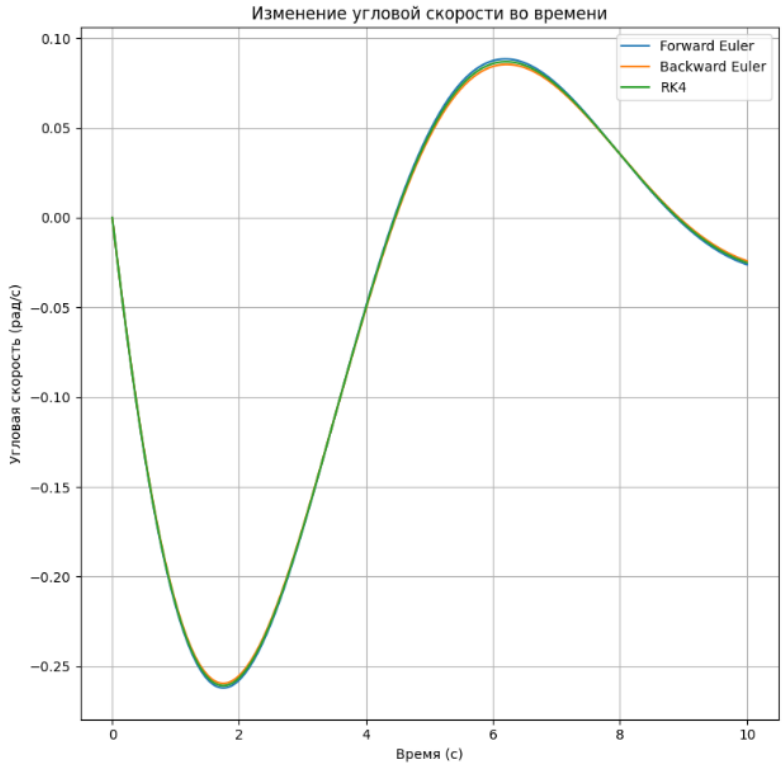

Рисунок 3. График скорости после выполнения кода

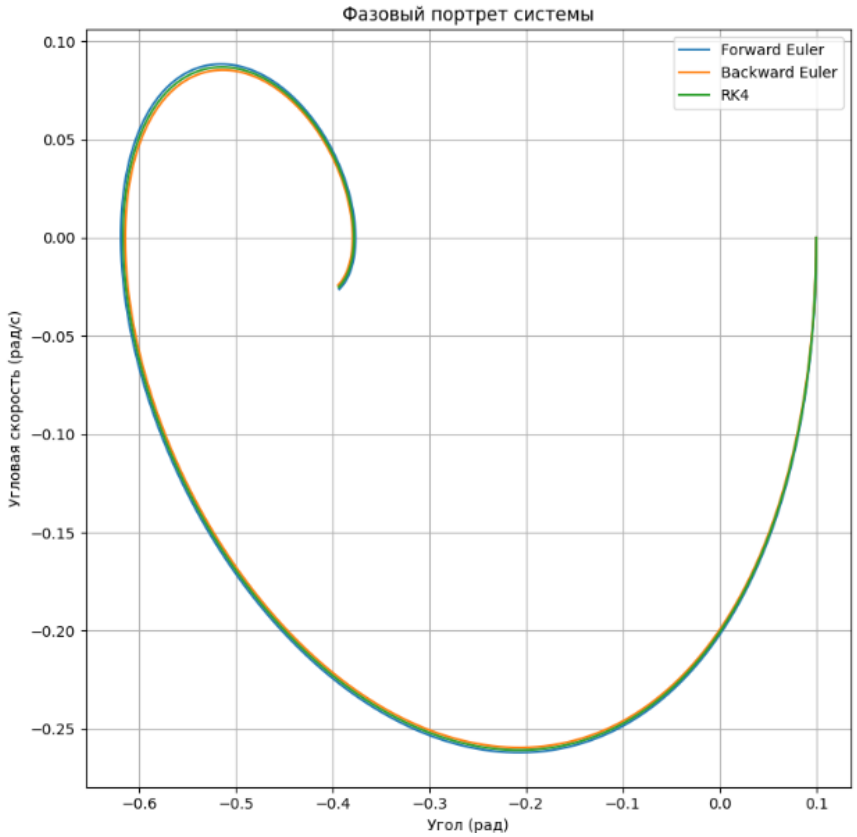

Рисунок 4. График фазы после выполнения кода

После постройки графиков делаем проверку в Simulink. Построим схему ОДУ (рисунок 5).

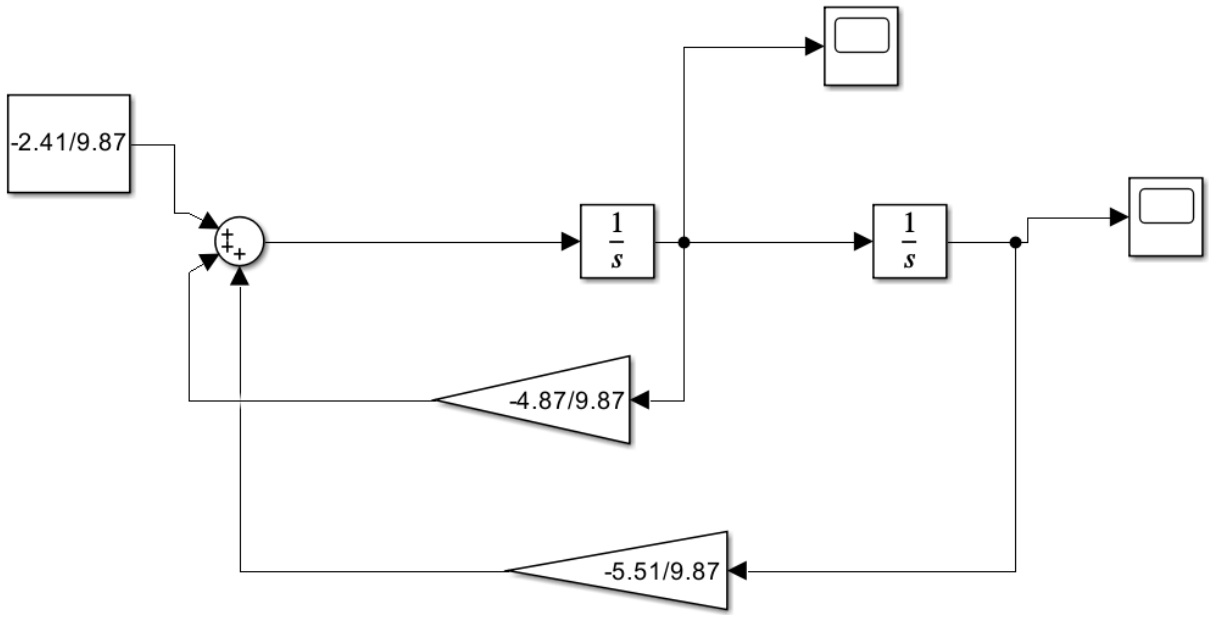

Рисунок 5. Схема ОДУ в Simulink

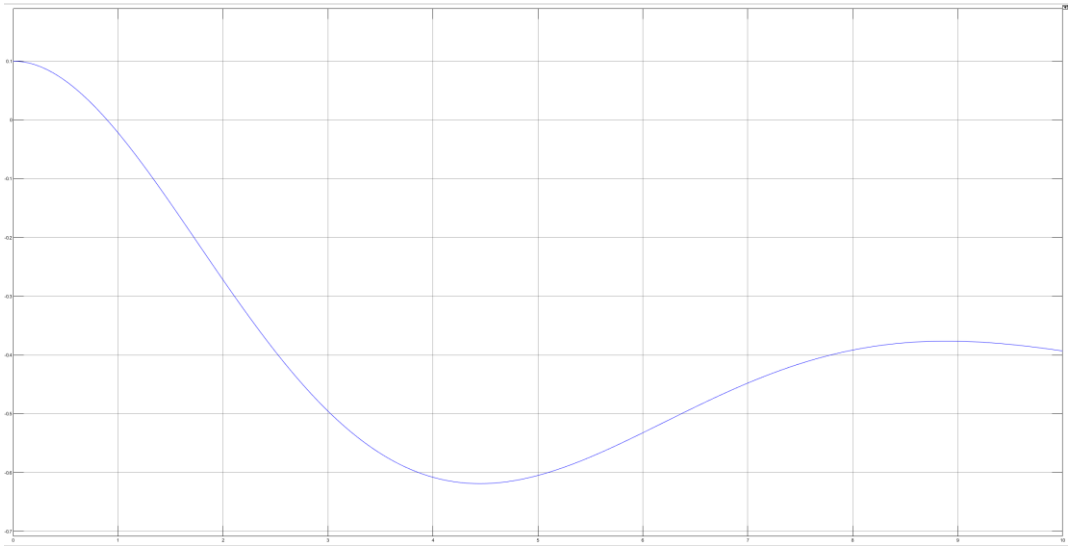

Рисунок 6. График положения методом явного Эйлера

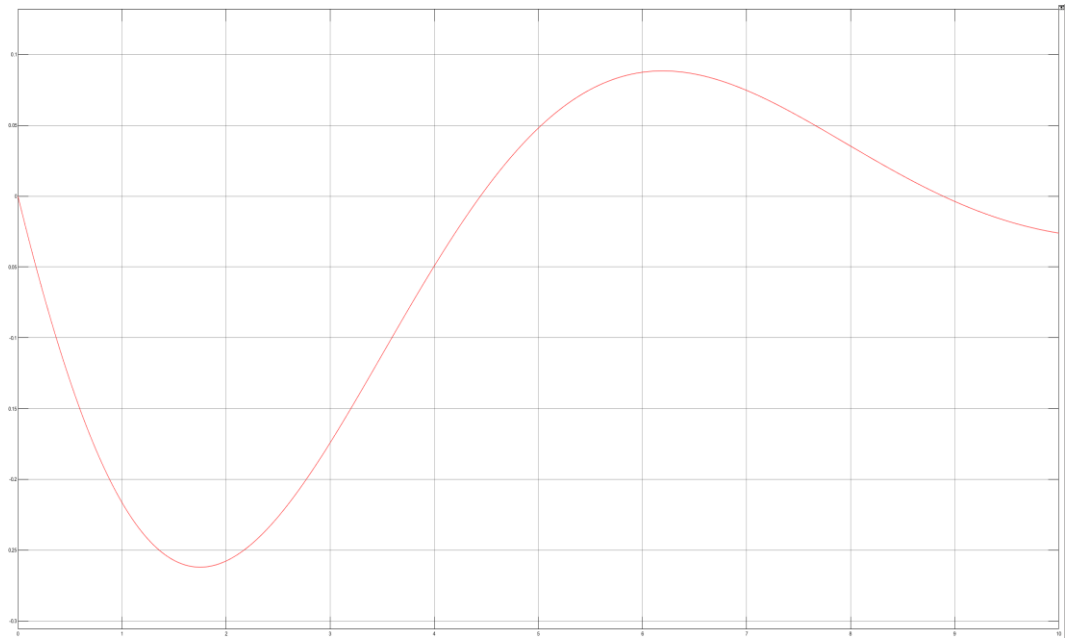

Рисунок 7. График скорости методом явного Эйлера

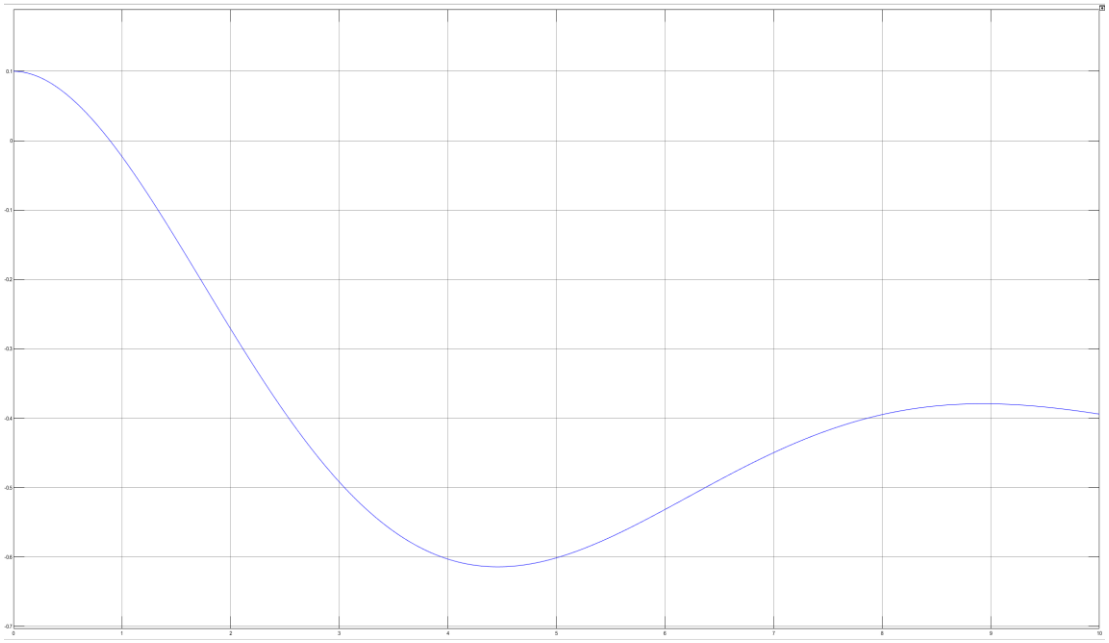

Рисунок 8. График положения методом неявного Эйлера

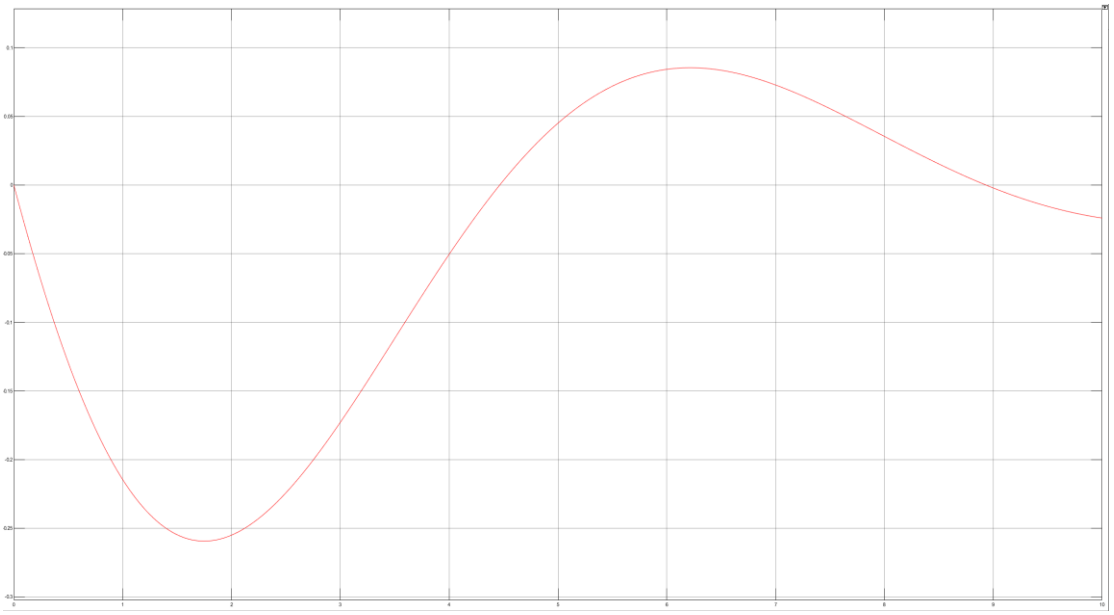

Рисунок 9. График скорости методом неявного Эйлера

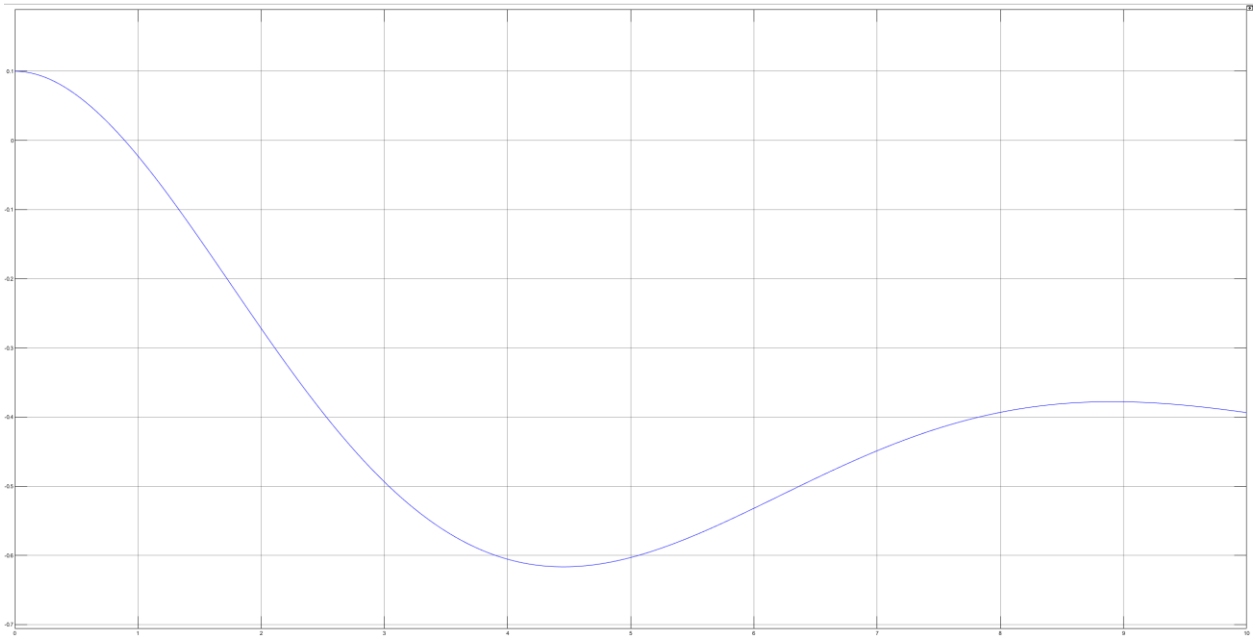

Рисунок 10. График положения методом Рунге-Кутта

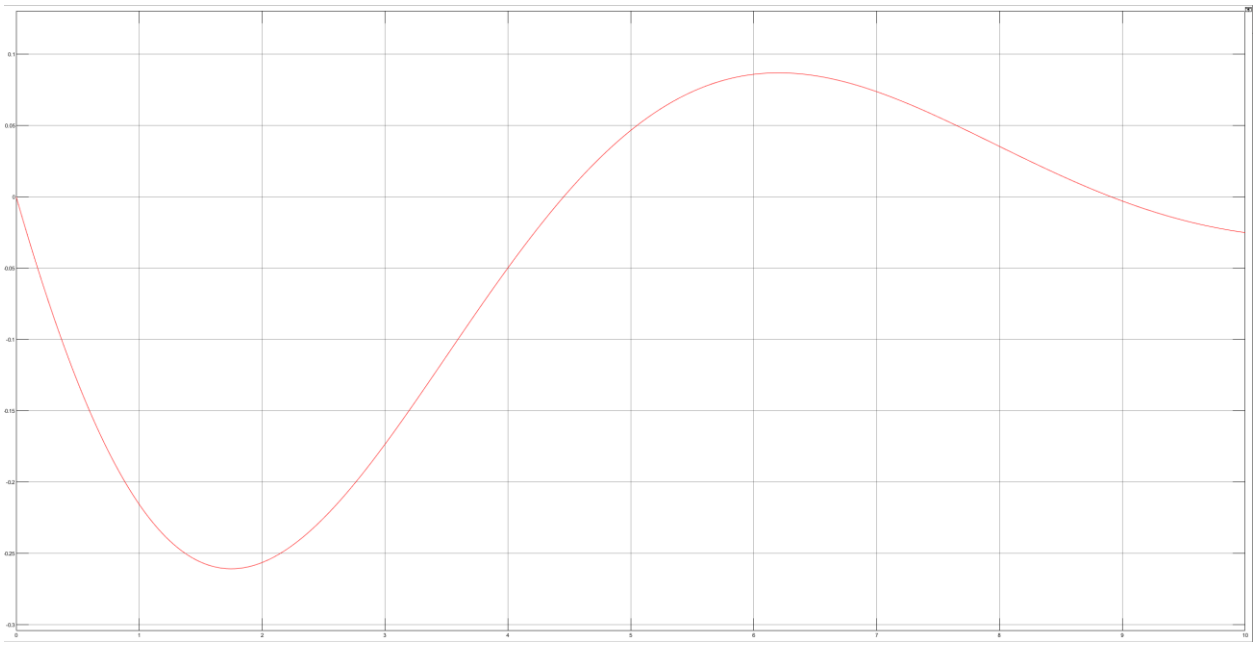

Рисунок 11. График скорости методом Рунге-Кутта

Выводы по графикам:

Все три метода успешно аппроксимируют аналитическое решение, но метод Рунге–Кутты — наиболее точный.

Моделирование в Simulink показало идентичные тенденции, что доказывает правильность реализации ОДУ и корректность выбранных параметров.

4. Вывод

В ходе выполнения практической работы была проведена проверка решения заданного дифференциального уравнения с использованием трёх численных методов — явного Эйлера, неявного Эйлера и метода Рунге–Кутты. Для верификации результатов была создана схема в среде Simulink, что позволило визуально сравнить полученные решения с аналитическим.

Анализ графиков положения, скорости и фазовой траектории показал, что все реализованные методы корректно описывают динамику исследуемой системы. Наиболее точные результаты демонстрирует метод Рунге–Кутты, поскольку он обеспечивает меньшие погрешности и лучшую устойчивость при том же шаге интегрирования.

Таким образом, поставленная цель работы достигнута: проверка численных методов решения ОДУ успешно проведена, их результаты совпадают с аналитическим решением, а моделирование в Simulink подтвердило корректность построенных моделей и выбранных параметров.

5. Приложение

Приложение №1:

Код Python для ОДУ и построения графиков

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


def pendulum_dynamics(x):
    """
    Уравнения движения системы (модифицированного маятника).
    x = [θ, dθ/dt] — вектор состояния.
    Формула: d²θ/dt² = (-b*dθ/dt - c*θ + d) / a
    """
    a = 9.87
    b = 4.87
    c = 5.51
    d = -2.41
    theta = x[0]
    theta_dot = x[1]

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

    return np.array([theta_dot, theta_ddot])


def forward_euler(fun, x0, Tf, h):
    """
    Метод прямого (явного) Эйлера для численного интегрирования.
    """
    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):
        # Следующее состояние вычисляется из текущего с помощью шага h
        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):
    """
    Метод обратного (неявного) Эйлера с итерацией до сходимости.
    """
    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):
        # Начальное приближение для итерации — текущее состояние
        x_hist[:, k + 1] = x_hist[:, k]

        for i in range(max_iter):
            # Итерационно уточняем следующее состояние
            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

            # Проверяем условие сходимости
            if error < tol:
```

```
                break

    return x_hist, t


def runge_kutta4(fun, x0, Tf, h):
    """
    Метод Рунге-Кутты 4-го порядка для численного интегрирования.
    """
    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):
        # Четыре промежуточных шага для повышения точности
        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


# Начальные условия и параметры моделирования
x0 = np.array([0.1, 0.0])  # [начальный угол, начальная угловая скорость]
Tf = 10.0  # общее время моделирования
h = 0.01   # шаг интегрирования

# Расчёт решений тремя методами
x_fe, t_fe = forward_euler(pendulum_dynamics, x0, Tf, h)
x_be, t_be = backward_euler(pendulum_dynamics, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(pendulum_dynamics, x0, Tf, h)

# Визуализация результатов
plt.figure(figsize=(24, 8))

# График изменения угла во времени
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('Время (с)')
plt.ylabel('Угол (рад)')
plt.legend()
plt.title('Изменение угла во времени')
plt.grid(True)

# График изменения угловой скорости
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('Время (с)')
plt.ylabel('Угловая скорость (рад/с)')
plt.legend()
plt.title('Изменение угловой скорости во времени')
plt.grid(True)

# Фазовый портрет (угол vs угловая скорость)
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('Угол (рад)')
plt.ylabel('Угловая скорость (рад/с)')
plt.legend()
plt.title('Фазовый портрет системы')
plt.grid(True)

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