Министерство образования и науки Российской Федерации ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Отчёт по практической работе №1

по дисциплине

«Имитационное моделирование робототехнических систем»

Отчёт выполнила: Румянцева А.В. (340890)

Преподаватель: Ракшин Е.А. (373529)

Дата выполнения: 06.11.2025

Санкт-Петербург, 2025

1. Постановка задачи и цель работы

В данной работе предложено проверить решение дифференциального уравнения (далее ODE), используя при этом три метода: явный Эйлер, неявный Эйлер и метод Рунге-Кутты. Также в работе предложено построить получившиеся графики и проверить их с аналитическим решением системы.

2. Решение ODE

a. Выполнение работы

В ходе работы было предложено исследовать следующее ODE, параметры которого зависят от варианта в предложенной таблице:

$$a\ddot{x} + b\dot{x} + cx = d, \qquad \begin{matrix} a = -9.28 \\ b = -9.91 \\ c = 7.54 \\ d = -0.18 \end{matrix},$$

однако перед работой над ODE, необходимо привести его к такому виду, которое даёт решение для $\ddot{x}$, то есть:

$$\ddot{x} = (d - b\dot{x} - cx)/a$$

В ходе выполнения работы был предоставлен исходный код для всех трёх методов, озвученных ранее, поэтому методы не будут детально описаны. Для аналитического решения была собрана схема в среде Simulink, предоставленная на рисунке 1.

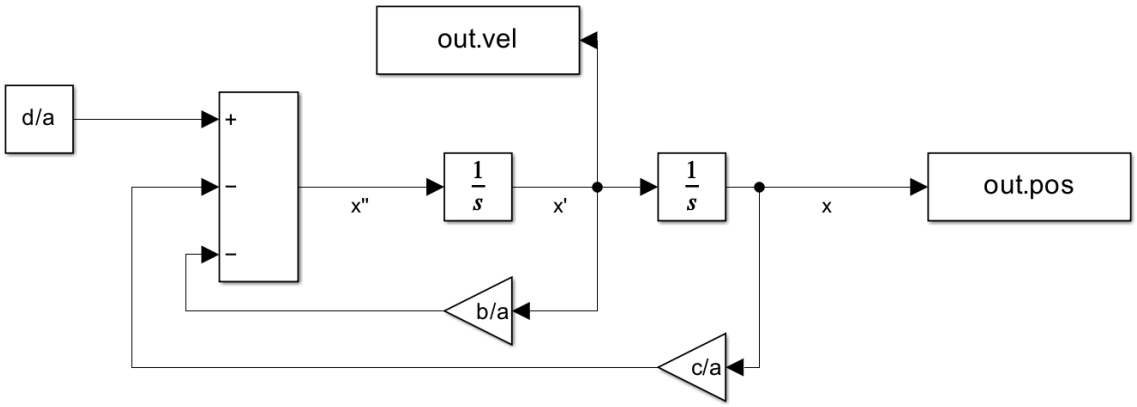

Рисунок 1. Схема решения ODE в среде Simulink

b. Результат

По завершению работы программы (приложение №1), получаются следующие графики, представленные на рисунке 2.

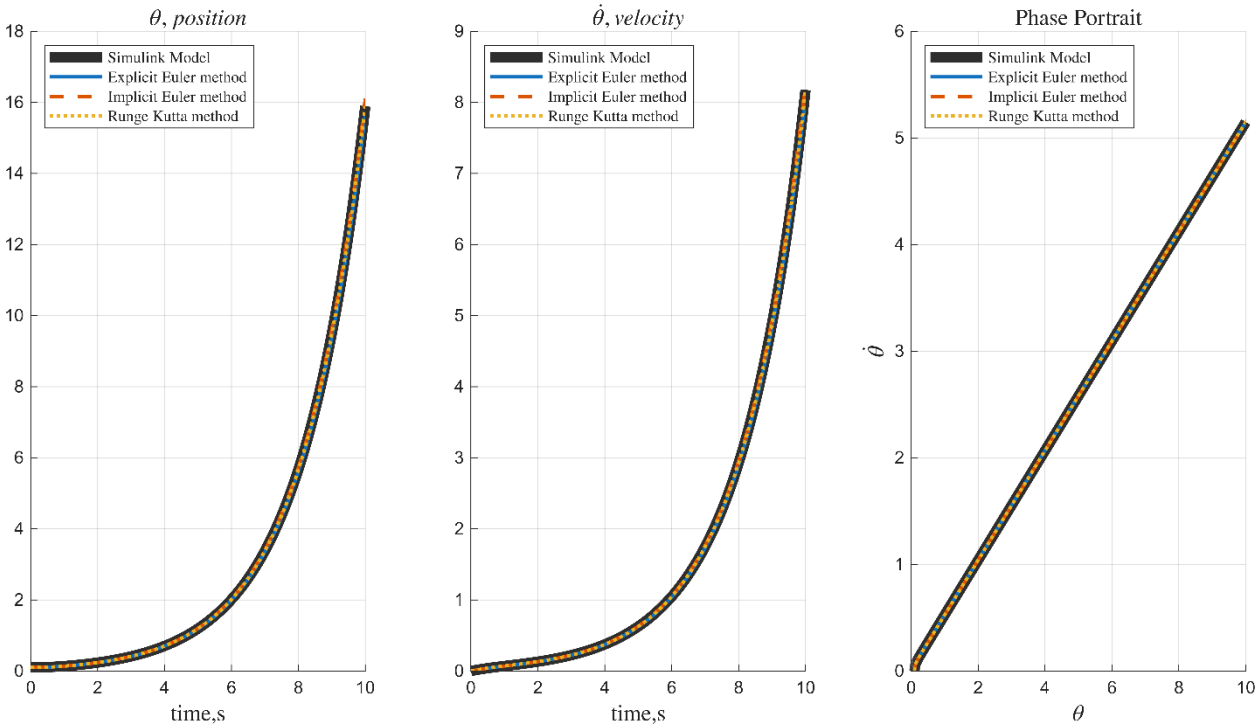

Рисунок 2. Результат аналитического решения и трёх методов

Как можно заметить из рисунка №2, общее поведение системы удалось передать с помощью всех трёх методов. Однако для дальнейшего сравнения необходимо рассмотреть их детально.

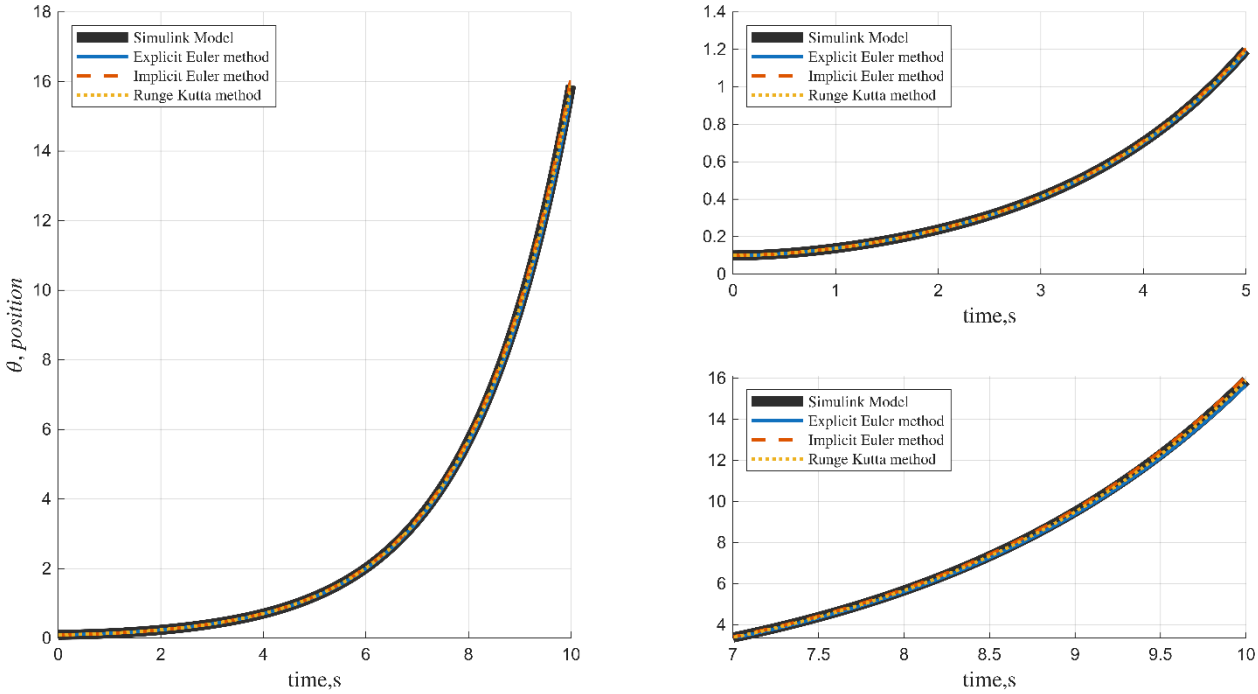

Рисунок 3. Результат аналитического решения для позиции и трёх методов в разных масштабах

При приближении временных отрезков видно, что с течением времени у явного и неявного времени «набегает» ошибка. При малых отрезках времени, решение будет достаточно точным, но с его течением точность будет теряться.

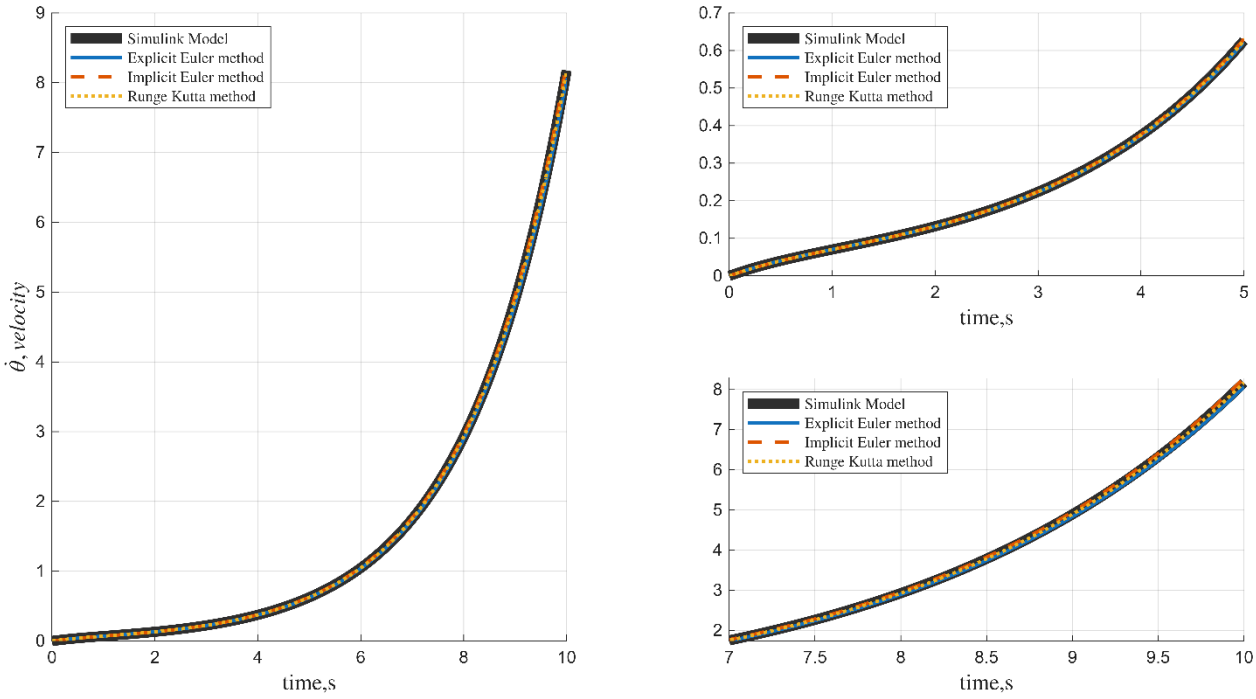

Рисунок 4. Результат аналитического решения для скорости и трёх методов в разных масштабах

На рисунке №4 наблюдается ровно та же ситуация, что и на рисунке №3. Эти же выводы подтверждаются при взгляде на рисунки №5 и №6.

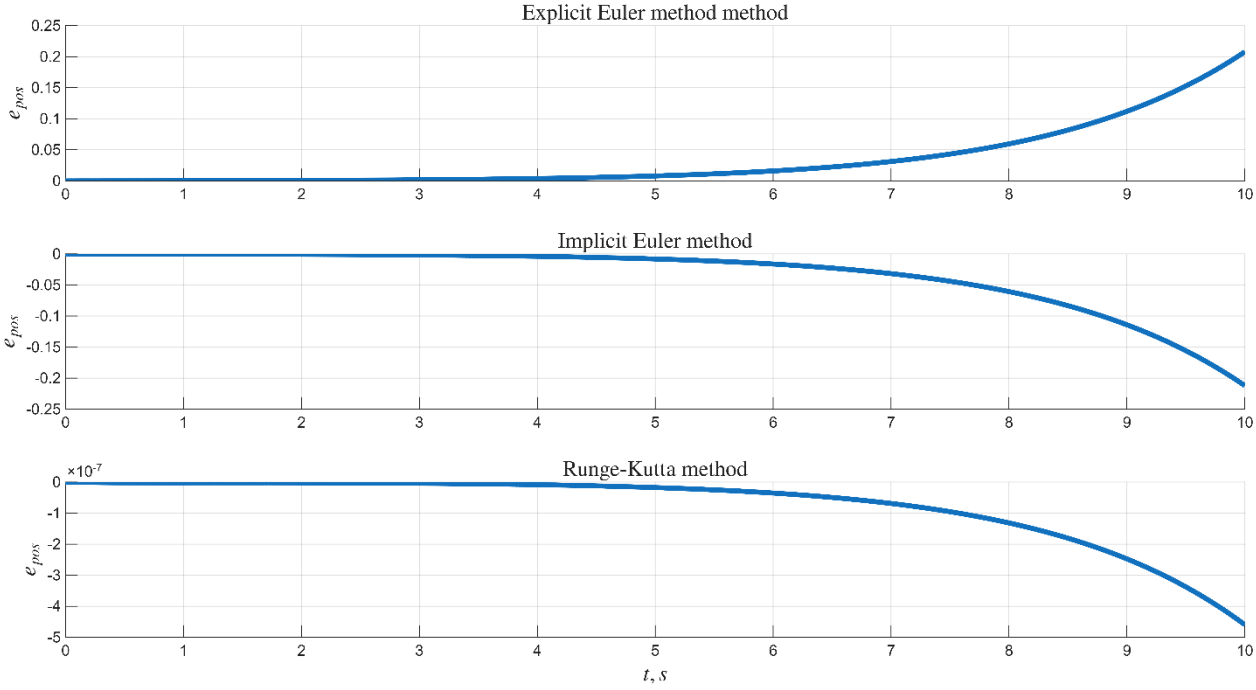

Рисунок 5. Графики ошибок аналитических решений по положению

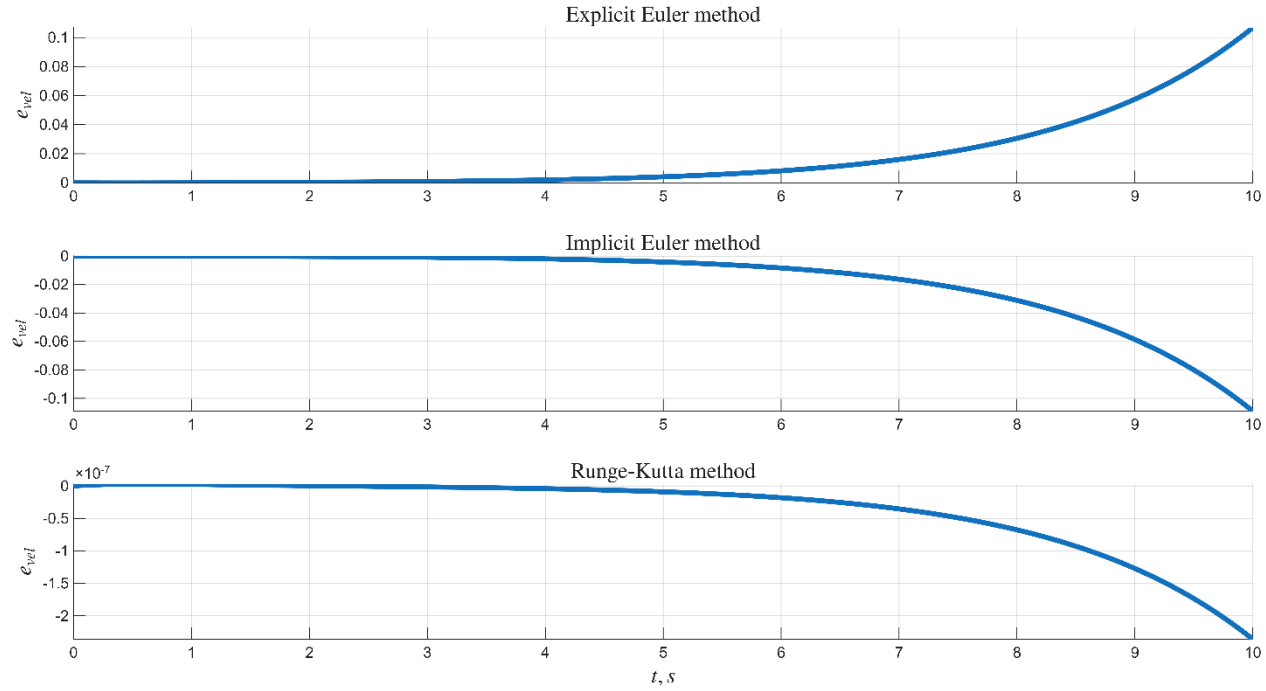

Рисунок 6. Графики ошибок аналитических решений по скорости

3. Выводы

В ходе лабораторной работы были рассмотрены три метода решения однородных дифференциальных уравнений. Уже на основании графического сравнения и его более приближенном варианте видны недостатки методов Эйлера. Проблема в виде нарастающего значения ошибки подтвердилось и при рассмотрении рисунок №5 и №6, полученных при вычитании решения каждого метода из значений, полученных симуляцией в среде Simulink. Благодаря этому сразу становится понятно, почему метод Рунге-Кутты является стандартом в отрасли робототехники, ведь ошибка этого метода является невероятно малой.

4. Приложения

Приложение №1. Решение с помощью трёх методов

Импорт необходимых библиотек

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

Параметры для варианта № 45

```
a = -9.28;
b = -9.91;
c = 7.54;
d = -0.18;
```

Функция для ODE

```
def ODE(x):

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

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

    return np.array([theta_dot, theta_ddot])
```

Необходимые три метода (явный Эйлер, неявный Эйлер и Рунги), взятые из готового файла

```
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
```

Запуск всех трёх методов

```
x0 = np.array([0.1, 0.0])  # Начальные условия: [положение, скорость]
Tf = 10.0
h = 0.01

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

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

# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(ODE, x0, Tf, h)
```

Экспорт значений в формат csv

```
df_fe = pd.DataFrame({
    'time': t_fe,
    'position': x_fe[0,:],
    'velocity': x_fe[1,:]
})

df_be = pd.DataFrame({
    'time': t_be,
    'position': x_be[0,:],
```

```
    'velocity': x_be[1,:]
})

df_rk4 = pd.DataFrame({
    'time': t_rk4,
    'position': x_rk4[0,:],
    'velocity': x_rk4[1,:]
})

df_fe.to_csv('forward_euler.csv', index=False)
df_be.to_csv('backward_euler.csv', index=False)
df_rk4.to_csv('runge_kutta.csv', index=False)
```