*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 №2
# по дисциплине
# «Имитационное моделирование робототехнических систем»

Отчёт выполнила: Румянцева А.В. (340890)

Преподаватель: Ракшин Е.А. (373529)

Дата выполнения: 09.11.2025

*Санкт-Петербург, 2025*

## 1. Постановка задачи и цель работы

В данной работе предложено составить ODE по нарисованной схеме и проверить решение энного, 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.

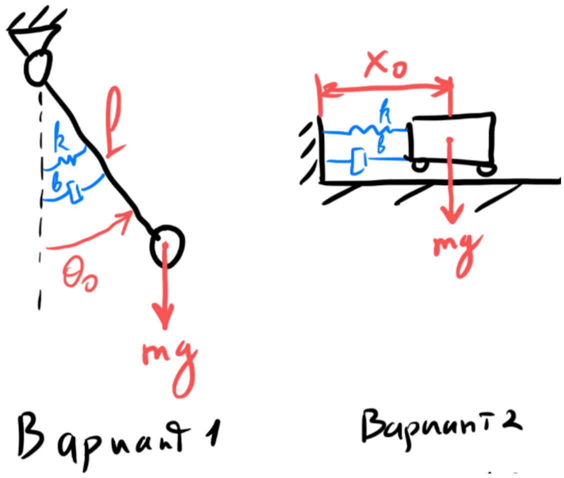


Рисунок 1. Два вида схем из технического задания

## 2. Решение ODE (вариант 1)

### a. Выполнение работы

Для того, чтобы составить ODE схемы, представленной на рисунке 1, необходимо записать его как Лагранжиан, а именно:

$$L = K - P, \qquad \begin{array}{c} K = \frac{1}{2}m\omega^2 \Rightarrow K = \frac{1}{2}ml^2\dot{\theta}^2 \\ P = -m * g * l * \cos\theta + \frac{1}{2}k\theta^2 \end{array},$$

однако только Лагранжиана недостаточно, ведь в схеме ещё присутствует демпфирование, которое не учитывается на данном этапе. Более того, чтобы получить ODE, необходимо применить метод Эйлера-Лагранжа:

$$\frac{d}{dx}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = Q, \qquad Q = -b\dot{\theta}.$$

Таким образом, получается следующее выражение:

$$ml^2\ddot{\theta} + m * g * l * \sin\theta + k\theta = -b\dot{\theta},$$

которое необходимо привести к общему виду. Делается это путём переноса за знак равно и деление на коэффициенты при $\ddot{x}$. Таким образом:

$$\ddot{x} = -\frac{b\dot{\theta}}{ml^2} - \frac{g * \sin\theta}{l} - \frac{k\theta}{ml^2}.$$

Для решения задания будет использован тот же код, что и в прошлом задании. Единственное изменение – само уравнение ODE. Сравнение с аналитическим решением будет таким же, в среде Simulink.

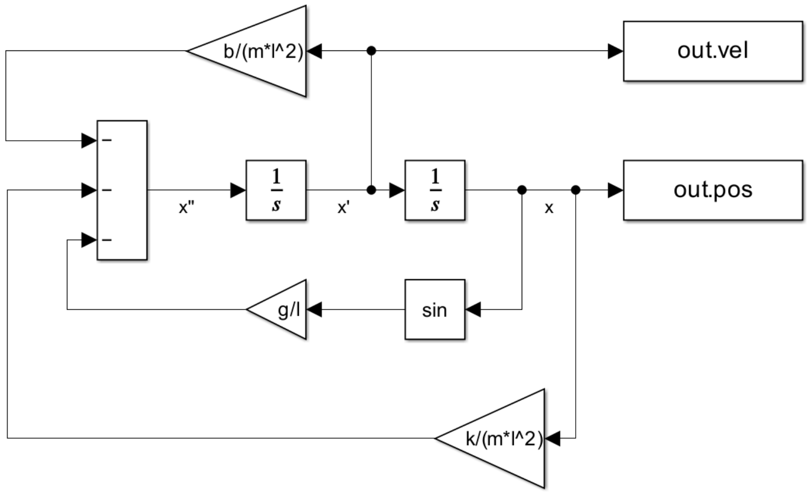


Рисунок 2. Схема моделирования ODE в среде Simulink

## b. Результат

При выполнении кода (приложение №1), получились следующие результаты.

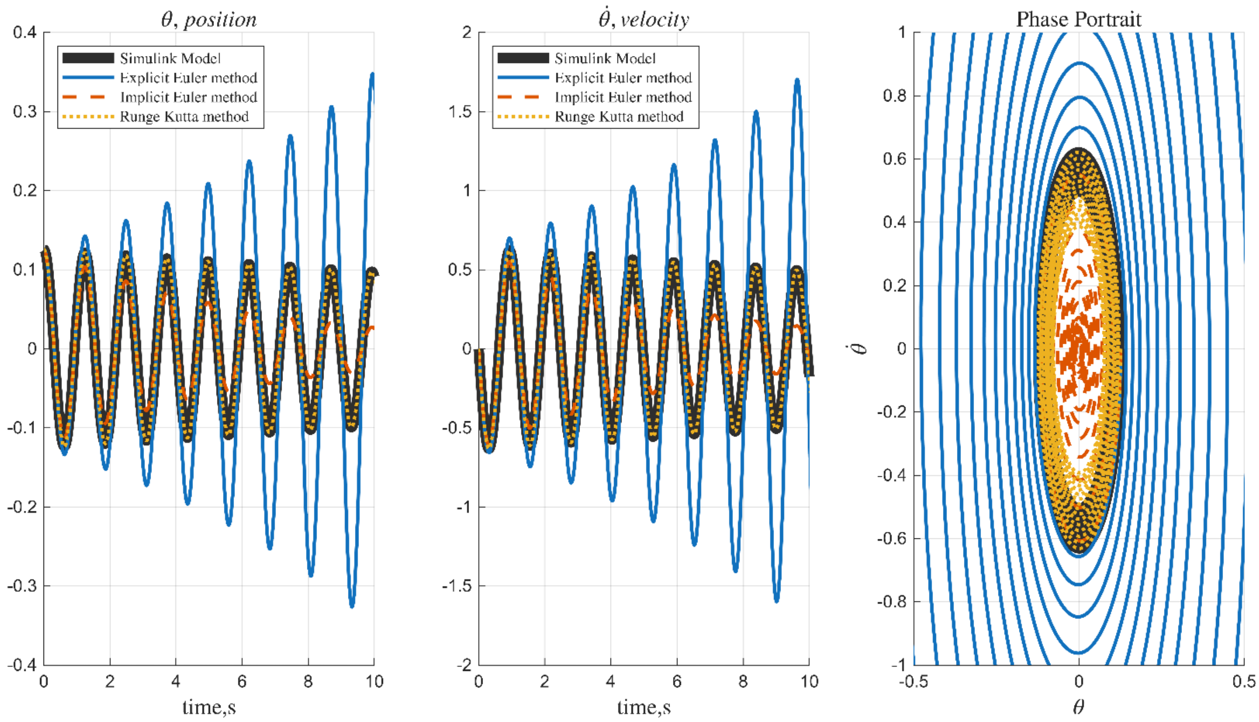


Рисунок 3. Результат аналитического решения и трёх методов

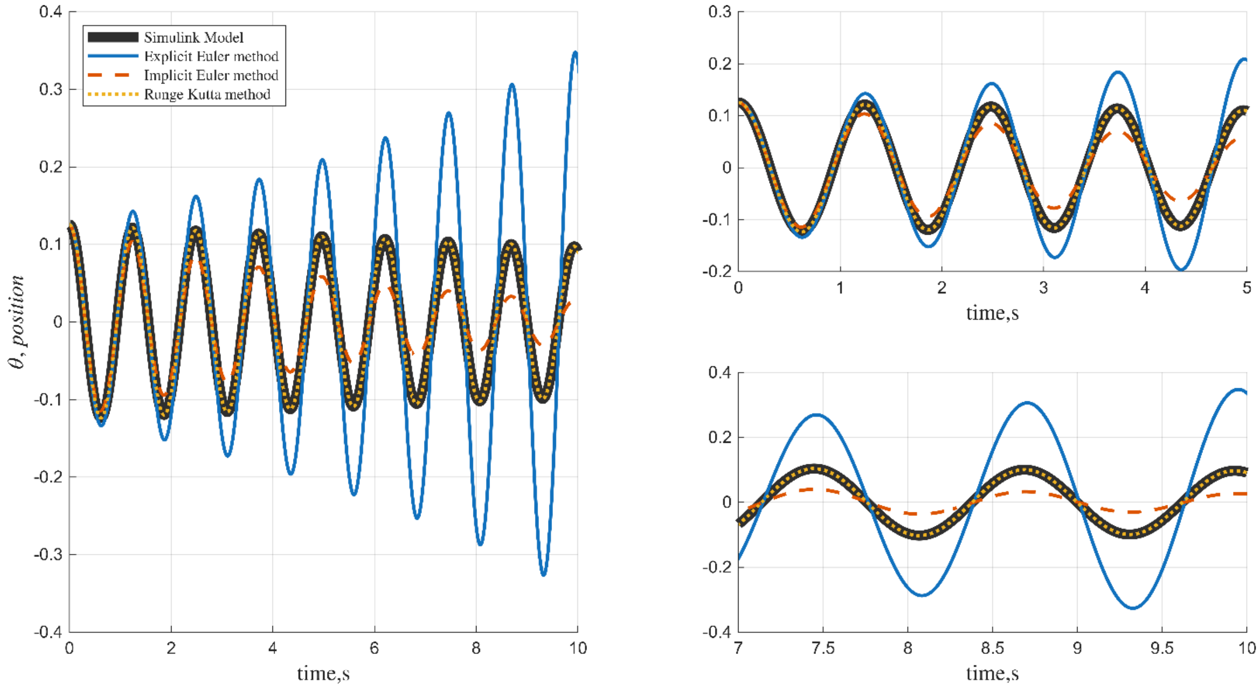


Рисунок 4. Результат аналитического решения для позиции и трёх методов в разных масштабах

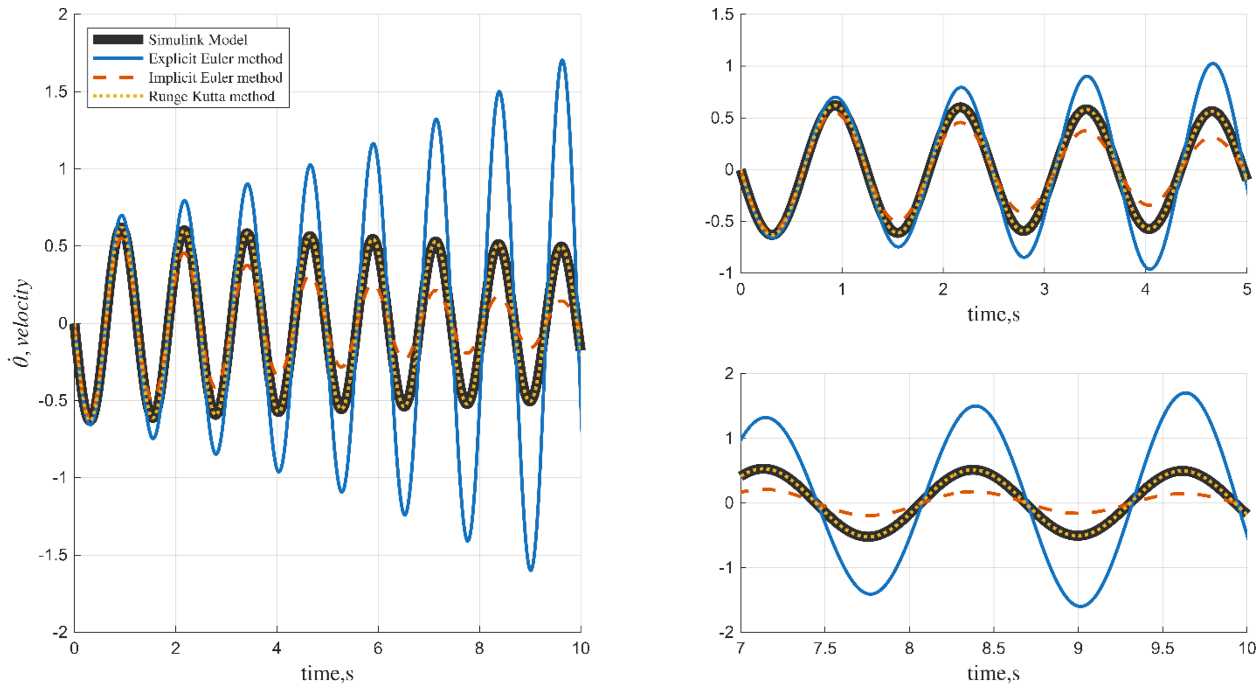


Рисунок 5. Результат аналитического решения для скорости и трёх методов в разных масштабах

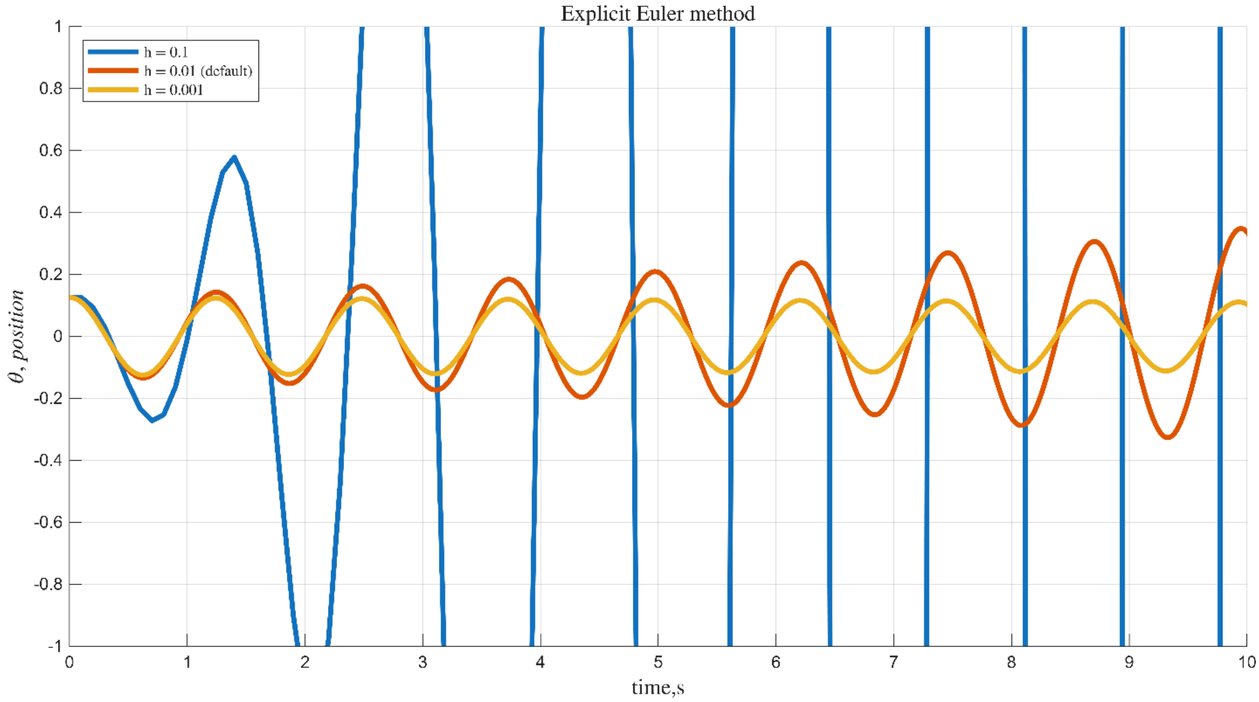


Рисунок 6. Сравнение зависимости явного Эйлера от шага моделирования при анализе позиции

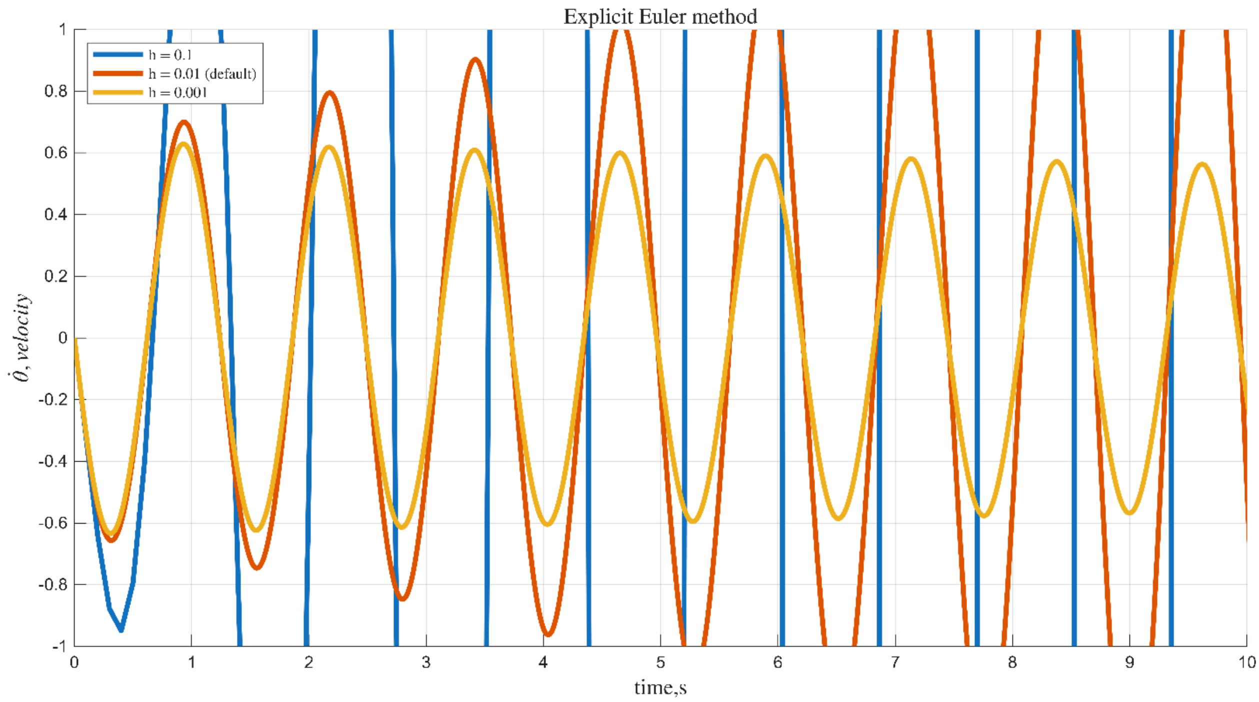


Рисунок 7. Сравнение зависимости явного Эйлера от шага моделирования при анализе скорости

## 3. Выводы

В ходе лабораторной работы подтвердились и теоретические знания, полученные в ходе лекций, и выводы из прошлой лабораторной работы. Сам по себе маятник - стабильная система, которая должна затухать, однако из-за нестабильности и зависимости от шага моделирования, явный метод Эйлера показывает возрастающий график. Неявный метод уже показывает общую динамику системы и её стремление к нулю. Метод Рунге-Кутты, как и в прошлой лабораторной работе, совпадает идеально и имеет минимальную ошибку, ещё раз подтверждая звание производственного стандарта.

## 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

```
m = 0.7;
k = 10.8;
b = 0.035;
l = 0.99;
theta_0 = 0.1256676092;
g = 9.81;
```

Функция для ODE

```
def ODE(x):

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

    theta_ddot = -(b*theta_dot)/(m*l*l) - g/l * np.sin(theta) -
k/(m*l*l)*theta

    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([theta_0, 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)
```