МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет систем управления и робототехники

Практическая работа №2
по дисциплине
«Имитационное моделирование робототехнических систем»
ВАРИАНТ №1
ID 506527

Студен:
Группа R4136c *Воронцов К.В.*

Преподаватель:
Ассистент *Е.А. Ракшин*

Санкт-Петербург 2025 г.

ЦЕЛЬ РАБОТЫ.

- Составить дифференциальное уравнение;
- Решить в аналитическом виде ОДУ
- Аналитически решить ОДУ. Если нет возможности, описать причину и возможно решение?
- Сравнить результаты этих методов с аналитическим решением (если оно существует), сформулировать мысли в отчёте.

#	Имя/Name	Вариант /Variant	ИСУ/ISU	m, kg	k, N/m, Nm/rad	b, N*s/m, Nm*s/rad	l, m	theta_0, rad	x_0, m
16	13 Воронцов Константин Владимирович	1	506527	0.4	15.4	0.005	0.41	-1.131008483	0.76

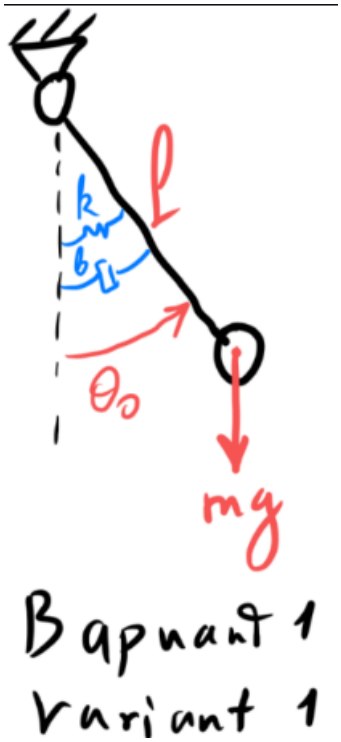

Рисунок 1 – Схема модели

АНАЛИТИЧЕСКОЕ РЕШЕНИЕ ОДУ ВТОРОГО ПОРЯДКА.

Требуется аналитическим методом решить ОДУ вида:

$$K = \frac{m \cdot l^2 \cdot \dot{\theta}^2}{2} \tag{1}$$

$$P = mgl + \frac{k \cdot x^2}{2} = mgl(1 - cos\theta) + \frac{k \cdot \theta^2}{2}$$

Запишем Лагранжиан

$$L = K - P$$

Подставим потенциальную и кинетическую энергию в Лагранжиан

$$L = \frac{m \cdot l^2 \cdot \dot{\theta}^2}{2} - gl(1 - cos\theta) + \frac{k \cdot \theta^2}{2}$$

Запишем уравнение Эйлера-Лагранжа

$$\frac{d}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\theta}}\right) - \frac{\mathrm{dL}}{\mathrm{d\theta}} = \mathrm{Q}$$

$$\mathrm{Q} = -\mathrm{b}\dot{x} = -b\dot{\theta}$$

$$\frac{\partial \mathrm{L}}{\partial \dot{\theta}} = ml^2\dot{\theta}$$

$$\frac{d}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\theta}}\right) = ml^2\ddot{\theta}$$

$$\frac{\mathrm{dL}}{\mathrm{d\theta}} = -mglsin\theta - k\theta$$

Собираем все вместе

$$ml^2\ddot{\theta} + mglsin\theta + k\theta = -b\dot{\theta}$$

Приведем к стандартному виду ОДУ

$$\ddot{\theta} = \frac{-mglsin\theta - k\theta - b\dot{\theta}}{ml^2}$$

Сделаем замену

$$R = \dot{\theta}$$

$$\frac{dR}{\mathrm{dt}} = \frac{-mglsin\theta - k\theta - bR}{ml^2}$$

Это нелинейное уравнение из-за члена $\sin\theta$.
Аналитического решения в элементарных функциях НЕ СУЩЕСТВУЕТ.

Если предположить малые углы, то $\sin\theta \approx \theta$, и уравнение становится линейным:

$$R = \dot{\theta}$$

$$\frac{dR}{\mathrm{dt}} = \frac{-mgl \cdot \theta - k \cdot \theta - bR}{ml^2}$$

Это линейное ОДУ с постоянными коэффициентами → аналитическое решение существует

Приведем код аналитического решения для данного уравнения.

```python

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp


m = 0.4      # масса
l = 0.41      # длина маятника
g = 9.81     # ускорение свободного падения
k = 15.4      # жёсткость "углового" элемента (в Н·м/рад)
b = 0.005      # коэффициент демпфирования (в Н·м·с)


theta0 = -1.131008483      # начальный угол (малый!)
dtheta0 = 0.0     # начальная угловая скорость

p = b / (m * l**2)
q = g / l + k / (m * l**2)


D = p**2 - 4 * q

```

```
22
23  if D < 0:
24      # Комплексные корни → затухающие колебания
25      alpha = -p / 2
26      beta = np.sqrt(-D) / 2
27
28      C1 = theta0
29      C2 = (dtheta0 - alpha * theta0) / beta
30
31      def theta_analytical(t):
32          return np.exp(alpha * t) * (C1 * np.cos(beta * t) + C2 * np.sir
33
34  elif D == 0:
35      # Критическое демпфирование
36      r = -p / 2
37      C1 = theta0
38      C2 = dtheta0 - r * theta0
39
40      def theta_analytical(t):
41          return (C1 + C2 * t) * np.exp(r * t)
42
43  else:
44      # Два действительных корня
45      r1 = (-p + np.sqrt(D)) / 2
46      r2 = (-p - np.sqrt(D)) / 2
47
48      A = np.array([[1, 1], [r1, r2]])
49      B = np.array([theta0, dtheta0])
```

```
        return np.exp(alpha * t) * (C1 * np.cos(beta * t) + C2 * np.sir

elif D == 0:
    # Критическое демпфирование
    r = -p / 2
    C1 = theta0
    C2 = dtheta0 - r * theta0

    def theta_analytical(t):
        return (C1 + C2 * t) * np.exp(r * t)

else:
    # Два действительных корня
    r1 = (-p + np.sqrt(D)) / 2
    r2 = (-p - np.sqrt(D)) / 2

    A = np.array([[1, 1], [r1, r2]])
    B = np.array([theta0, dtheta0])
    C1, C2 = np.linalg.solve(A, B)

    def theta_analytical(t):
        return C1 * np.exp(r1 * t) + C2 * np.exp(r2 * t)

def pendulum_exact(t, y):
    theta, dtheta = y
    d2theta = (-b * dtheta - m * g * l * np.sin(theta) - k * theta) / (
    return [dtheta, d2theta]

# Время
```

```
t_num = sol_num.t
theta_num = sol_num.y[0]

theta_an = theta_analytical(t_num)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(t_num, theta_num, 'b', label='Численное (с sinθ)')
plt.plot(t_num, theta_an, '--r', label='Аналитическое (линеаризованное)
plt.xlabel('Время (с)')
plt.ylabel('θ (рад)')
plt.title('Сравнение решений')
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(t_num, np.abs(theta_num - theta_an), 'k')
plt.xlabel('Время (с)')
plt.ylabel('Ошибка |θ_num - θ_an|')
plt.title('Ошибка линеаризации')
plt.yscale('log')
plt.grid()

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

Рисунок 2 –Сравнение численного и аналитического решения

Также можно собрать модель в Matlab Simulink и сравнить полученные решения

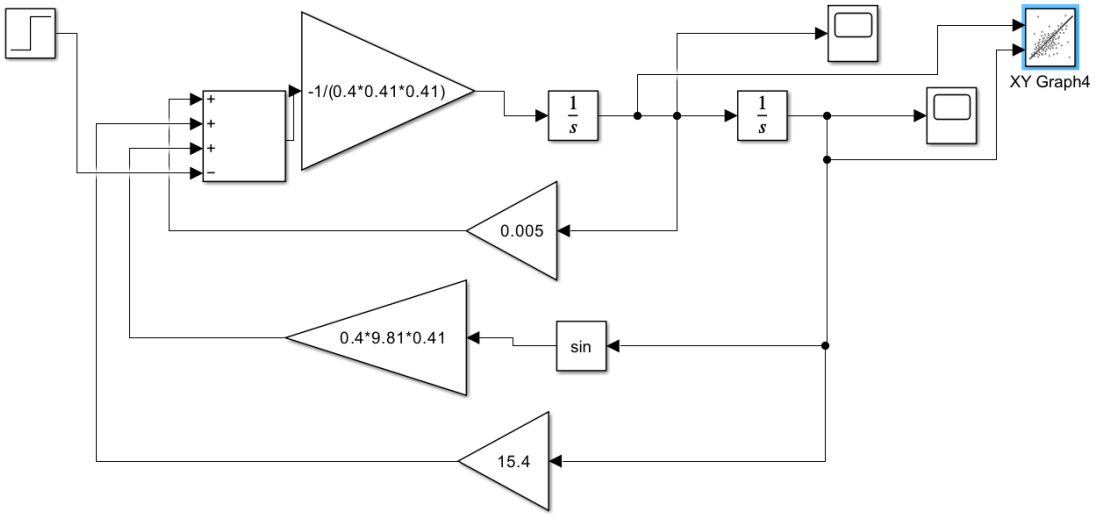

Рисунок 3 – Схема модели

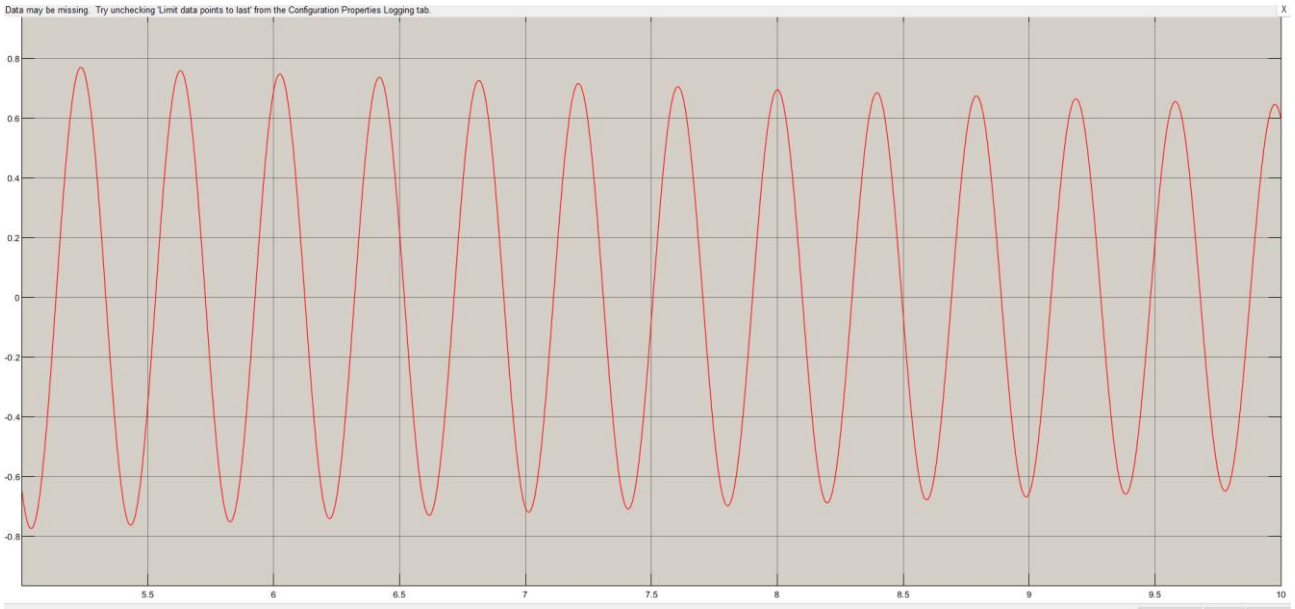

Рисунок 4 – Переходный процесс

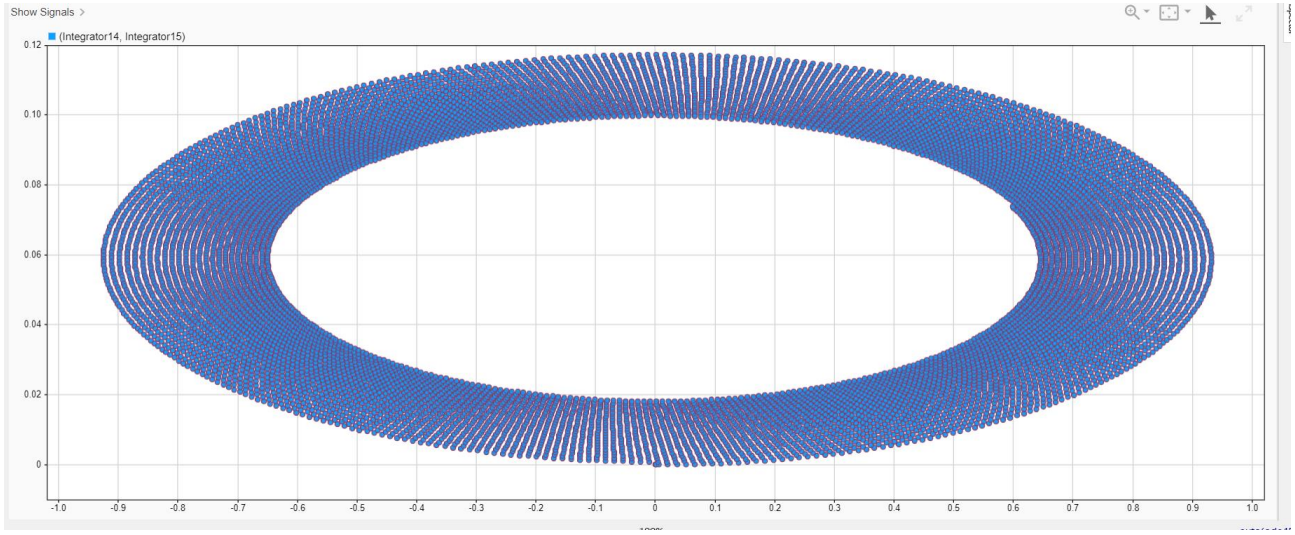

Рисунок 5 – Фазовый портрет (устойчивый фокус)

РЕШЕНИЕ ОДУ ВТОРОГО ПОРЯДКА ЧИСЛЕННЫМИ МЕТОДАМИ.

Если решать через код, который приведен для различных численных решателей, то получим следующее.

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

# Параметры системы
m = 0.4      # масса
l = 0.41      # длина маятника
g = 9.81     # ускорение свободного падения
k = 15.4      # жёсткость пружины
b = 0.005      # коэффициент демпфирования
    # недеформированная длина пружины (можно изменить)

def pendulum_dynamics(X):

    theta = X[0]
    dtheta = X[1]

    d2theta = ((-b*dtheta-m*g*l*np.sin(theta)-k*theta)/(m*l**2))


    return np.array([dtheta, d2theta])

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])
```

```
30
31      return x_hist, t
32
33  def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
34      """Implicit Euler integration method using fixed-point iteration"""
35      t = np.arange(0, Tf + h, h)
36      x_hist = np.zeros((len(x0), len(t)))
37      x_hist[:, 0] = x0
38
39      for k in range(len(t) - 1):
40          x_hist[:, k + 1] = x_hist[:, k]  # Initial guess
41
42          for i in range(max_iter):
43              x_next = x_hist[:, k] + h * fun(x_hist[:, k + 1])
44              error = np.linalg.norm(x_next - x_hist[:, k + 1])
45              x_hist[:, k + 1] = x_next
46
47              if error < tol:
48                  break
49
50      return x_hist, t
51
52  def runge_kutta4(fun, x0, Tf, h):
53      """4th order Runge-Kutta integration method"""
54      t = np.arange(0, Tf + h, h)
55      x_hist = np.zeros((len(x0), len(t)))
56      x_hist[:, 0] = x0
57
58      for k in range(len(t) - 1):
```

```
    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

    return x_hist, t


x0_state = np.array([-1.131008483, 0.0])  # [theta(0), dtheta(0)]
Tf = 10.0
h = 0.0001

# Forward Euler
x_fe, t_fe = forward_euler(pendulum_dynamics, x0_state, Tf, h)

# Backward Euler
x_be, t_be = backward_euler(pendulum_dynamics, x0_state, Tf, h)

# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(pendulum_dynamics, x0_state, Tf, h)


plt.figure(figsize=(24, 8))

# Угол от времени
plt.subplot(1, 3, 1)
plt.plot(t_fe, x_fe[0, :], label='Forward Euler')
```

```
# Угол от 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 θ (рад)')
plt.legend()
plt.title('Угол от времени')
plt.grid()

# Угловая скорость от времени
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('Угловая скорость от времени')
plt.grid()

# Фазовый портрет
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 θ (рад)')
```

```
plt.ylabel('Угловая скорость (рад/с)')
plt.legend()
plt.title('Угловая скорость от времени')
plt.grid()

# Фазовый портрет
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 θ (рад)')
plt.ylabel('Угловая скорость (рад/с)')
plt.legend()
plt.title('Фазовый портрет')
plt.grid()

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

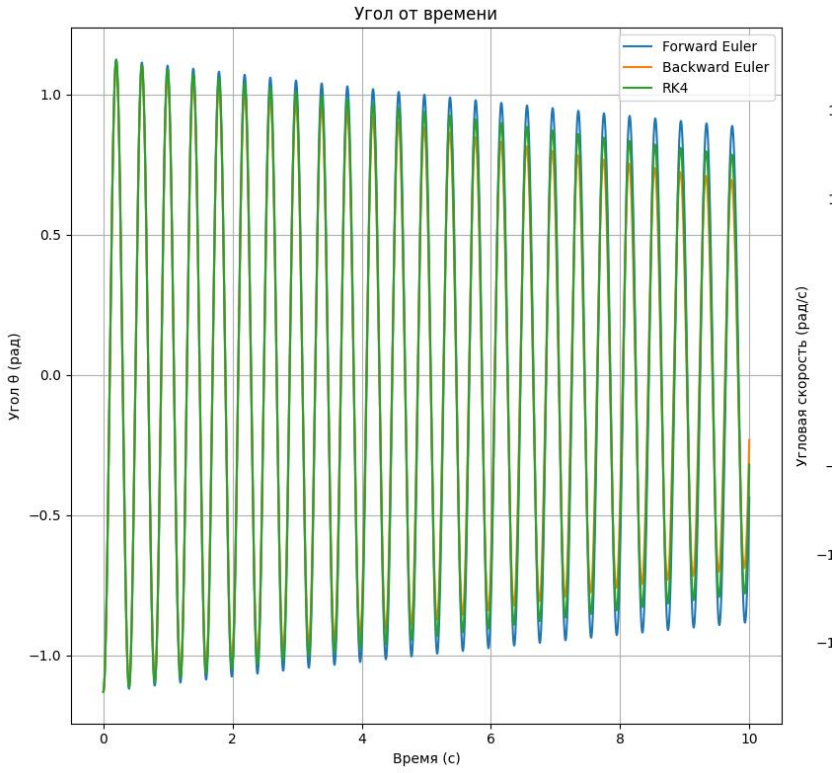

Рисунок 6 – Переходный процесс

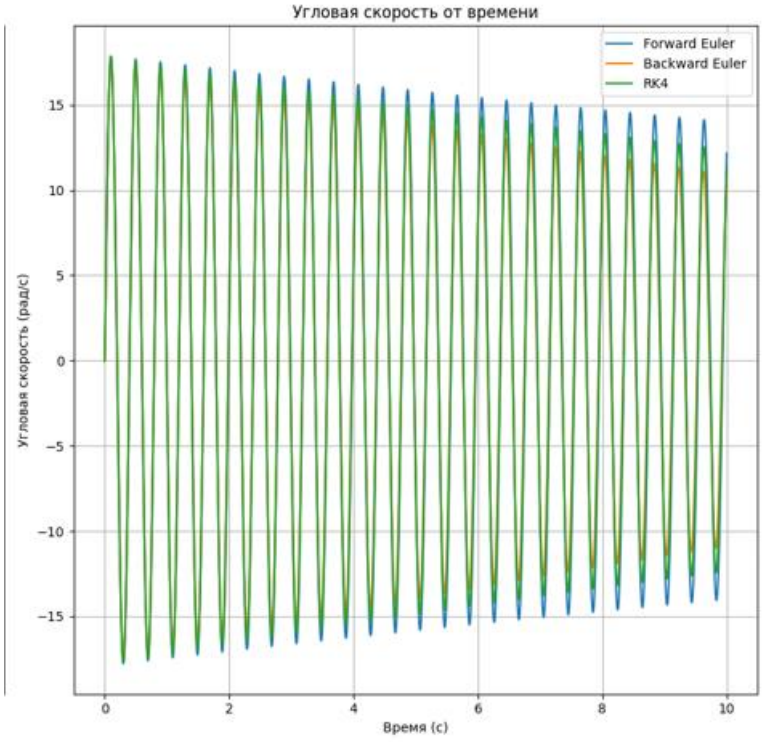

Рисунок 7 – Переходный процесс по скорости

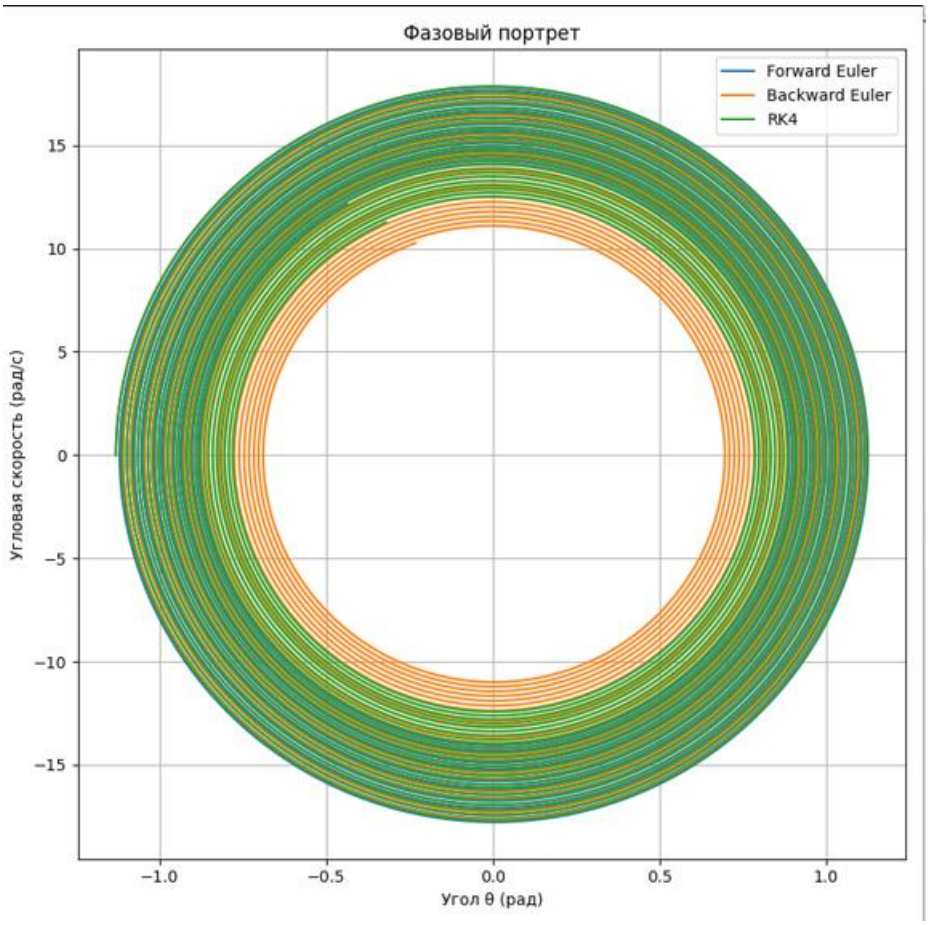

Рисунок 8 – Фазовый портрет

Как можно видеть фазовые портреты совпадают для аналитического решения через Matlab Simulink и через численное решение.

Аналитическое решение как и было сказано выше возможно только в случае рассмотрения малых углов

ВЫВОД.

Согласно полученным данным и визуальной оценкой, можно сделать вывод о том, что результаты различных способов расчета совпадают и некоторые приближения, принятые в ходе не сильно сказались на результатах.

По переходным процессам в численном расчете можно заметить, что Неявный Эйлер быстрее сходиться после него идет метод Рунги-Кутыы.