ИТМО
МФКТУ
ФСУИР

Практическая работа №2

Дисциплина: Имитационное моделирование робототехнических систем

Студент: Воробьёв Роман, R4133c

Преподаватель: Ракшин Егор Александрович

Санкт-Петербург
2025

Параметры системы: $m = 1$ кг, $g = 9{,}8\frac{\text{м}}{\text{с}^2}$, $k = 13{,}6\ \frac{\text{Н·м}}{\text{рад}}$, $b = 0{,}005\ \frac{\text{Н·м·с}}{\text{рад}}$, $l = 0{,}91$ м, $\theta_0 = 0{,}0314169022934339$ рад.

Кинетическую и потенциальную энергии качающегося математического маятника можно выразить через угол отклонения от вертикали следующим образом:

$$K = \frac{1}{2}ml^2\dot{\theta}^2, P = mgl(1 - \cos\theta) + k\theta^2.$$

Тогда Лагранжиан системы записывается как

$$\mathcal{L} = K - P = \frac{1}{2}ml^2\dot{\theta}^2 - (mgl(1 - \cos\theta) + \frac{1}{2}k\theta^2).$$

Уравнение движение $\frac{d}{dt}\left(\frac{\delta\mathcal{L}}{\delta\dot{\theta}}\right) - \frac{\delta\mathcal{L}}{\delta\theta} = Q$ будет иметь вид

$$ml^2\ddot{\theta} + mgl\sin\theta\,\dot{\theta} + k\theta = -b\dot{\theta}.$$

Получилось нелинейное ДУ, которое сложно решать, поэтому линеаризуем его, с учетом того, что начальное отклонение (а значит и все последующие значения θ) мало – оно составляет $0{,}01\pi$.

Получаем линейное ДУ

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

1. Аналитическое решение

$$ml^2\ddot{\theta} + b\dot{\theta} + k\theta = 0$$

Для решения неоднородного ДУ сначала найдем общее решение – решение соответствующего ОДУ.

Решим характеристическое уравнение:

$$ml^2\lambda^2 + b\lambda + k = 0$$

$$D = b^2 - 4kml^2 = -45{,}049 < 0$$

Дискриминант отрицателен, значит корни – комплексные числа, а решение ОДУ имеет вид $\theta(t) = e^{\alpha t}(C_1 \cos(\beta t) + C_2 \sin(\beta t))$,

$$\text{где} \begin{cases} \alpha = \frac{-b}{2a} = -0{,}0025 \\ \beta = \frac{\sqrt{D}}{2ai} = 3{,}356 \end{cases}$$

Полное решение будет иметь вид:

$$\theta(t) = e^{-0{,}0025t}(C_1 \cos(3{,}356t) + C_2 \sin(3{,}356t)).$$

C_1 и C_2 определяются начальными условиями.

Пусть $\theta(0) = \theta_0, \dot{\theta}(0) = 0, \ddot{\theta}(0) = 0$ тогда

$$\begin{cases} C_1 = \theta_0, \\ -0{,}0025 \cdot 3{,}356 \cdot C_2 = 0 \end{cases} \Rightarrow \begin{cases} C_1 = \theta_0 \\ C_2 = 0. \end{cases}$$

2. Численное решение

Сравним получившееся аналитическое решение с численными решениями уравнения разными методами. При этом численно решать будем изначальное нелинейное уравнение, чтобы убедиться в правомерности сделанной линеаризации.

Для начала убедимся, что аналитическое и численные решения близки для линеаризованного уравнения. На рисунке 1 показаны все решения.

Ошибка методов считалась как среднеквадратическое отклонение (СКО). Видно, что с течением времени явный метод Эйлера все больше расходится со всеми остальными решениями, поэтому далее использовать его не будем.

Решения методами Рунге-Кутта и неявным методом Эйлера очень близки к полученному аналитическому решению, а значит можно понять допустимость линеаризации сравнив аналитическое решение с численными решениями нелинейного уравнения этими методами.

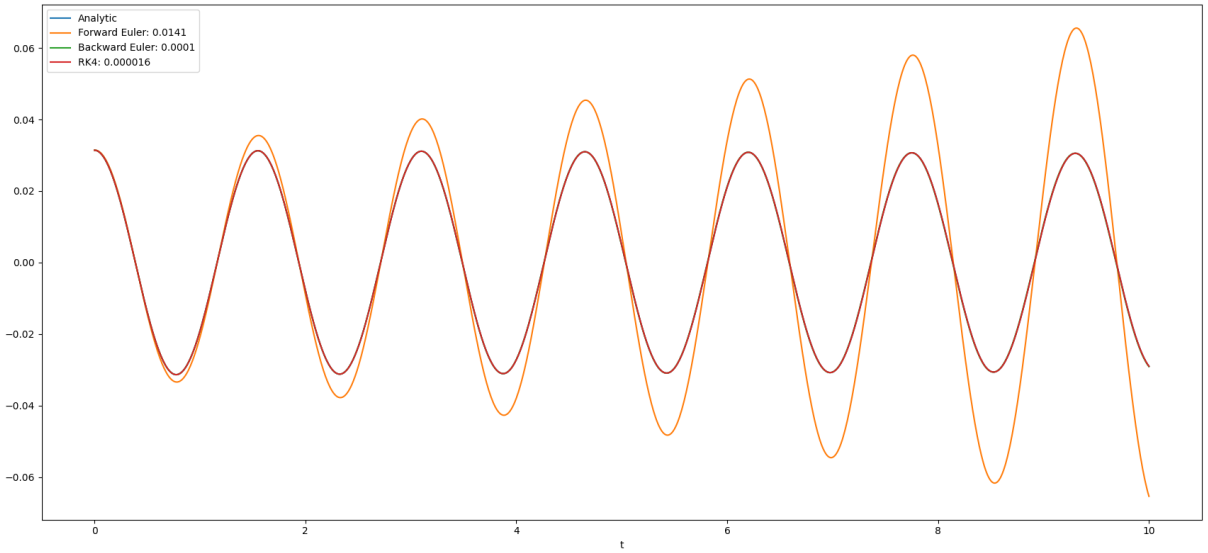

Рисунок 1 – Аналитическое и численные решения линейного дифференциального уравнения колебания маятника

Изменим функцию f(), возвращающую производные переменных уравнения, добавив нелинейность:

```
def f(x, y_vec):
    y, z = y_vec
    return np.array([z, (-m*g*l*z*np.sin(y) - b*z - k*y)/a])
```

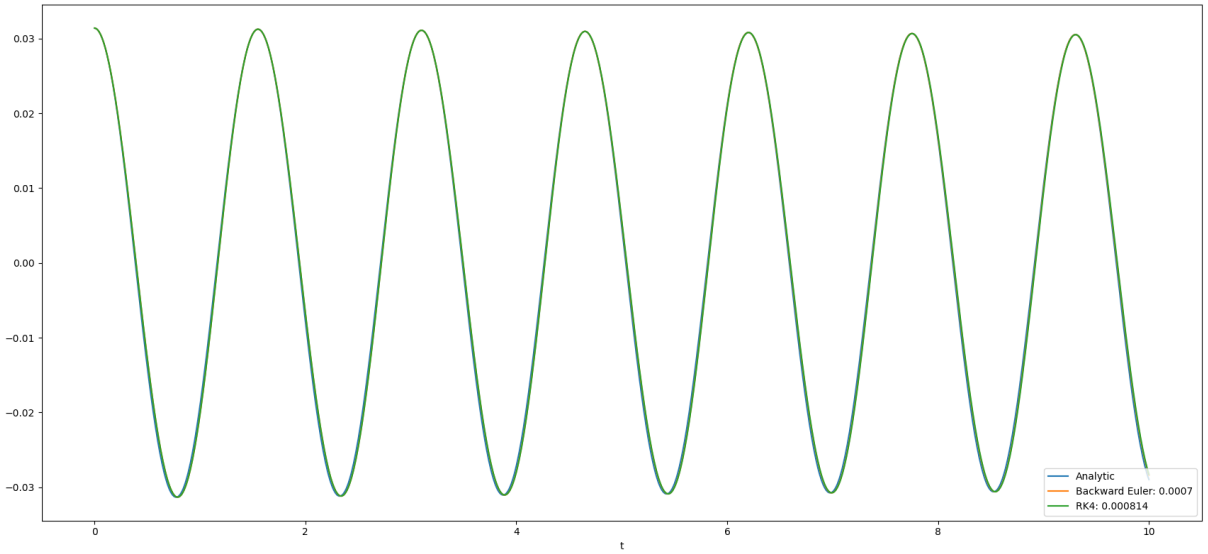

Рисунок 2 – Аналитическое решение линеаризованного уравнения и численные решения изначального нелинейного дифференциального уравнения колебания маятника

Отклонение численных решений нелинейного уравнения от аналитического решения линеаризованного уравнения больше раннее полученных значений, но все еще не так велико.

Увеличим время моделирования до 100 секунд и посмотрим, увеличится ли расхождение.

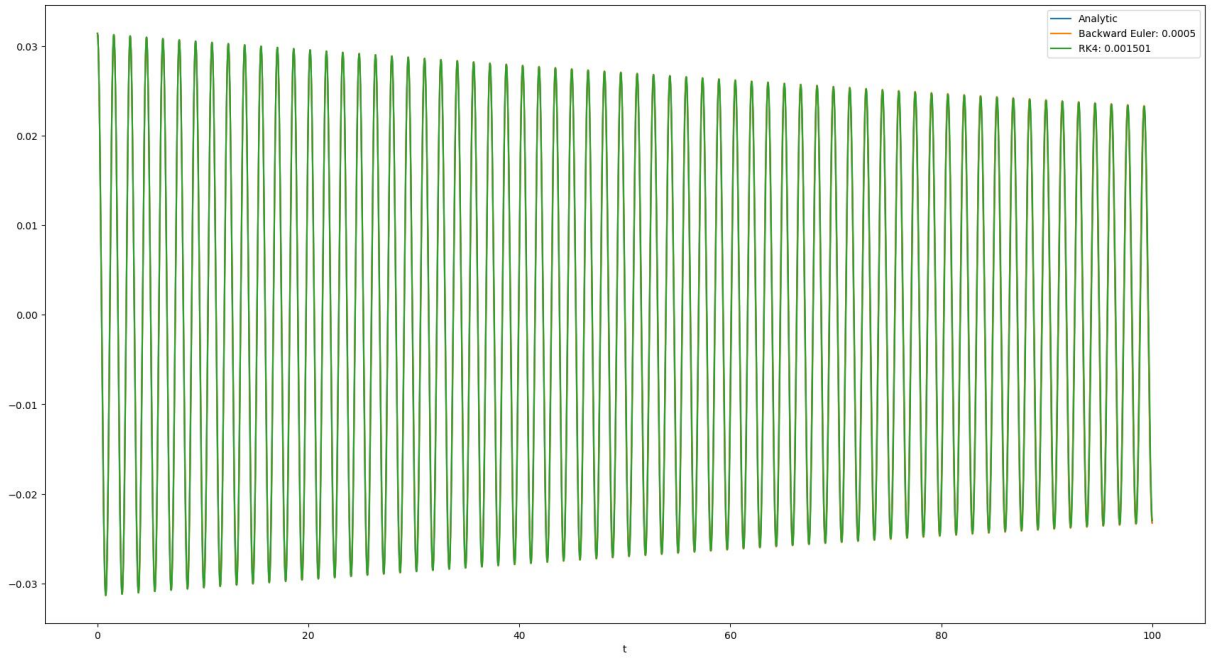

Рисунок 3 – Аналитическое решение линеаризованного уравнения и численные решения изначального нелинейного дифференциального уравнения колебания маятника на 100с

Отклонение решения методом Рунге-Кутта от аналитического решения линеаризованного уравнения выросло почти в 2 раза, а отклонение решения методом неявного Эйлера даже уменьшилось, что объясняется накоплением ошибки вычислений этого метода. Считая метод Рунге-Кутта достаточно точным, можно сказать, что линеаризация уравнения в данном случае допустима.

3. Код

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

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(t[k], x_hist[:, k])

    return x_hist, t

def backward_euler(fun, x0, Tf, h):
    """
    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] + h*fun(t[k] + 0.5*h, x_hist[:, k] +
0.5*h*fun(t[k], x_hist[:, k]))  # Initial guess

    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(t[k], x_hist[:, k])
        k2 = fun(t[k] + 0.5*h, x_hist[:, k] + 0.5 * h * k1)
        k3 = fun(t[k] + 0.5*h, x_hist[:, k] + 0.5 * h * k2)
        k4 = fun(t[k] + h, 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

#Analytic solution
def analitic_solution(Tf, h):
    t = np.arange(0, Tf + h, h)
    D = b**2 - 4*a*k
    alpha = 0.5*(-b)/a
    beta = 0.5*np.sqrt(-D)/a
    d = m*g*l/k
    return np.exp(alpha*t)*y0*np.cos(beta*t), t
```

```
m = 1
b = 0.005
k = 13.6
l = 0.91
g = 9.8

a = m*l*l

#Solution parameters
Tf = 10.0
h = 0.01
y0 = 0.031416
z0 = 0.0

#Function for system of DEs

def f(x, y_vec):
    y, z = y_vec
    return np.array([z, (-m*g*l*z*np.sin(y) - b*z - k*y)/a])


x0 = np.array([y0, z0])

x_an, t_an = analitic_solution(Tf, h)

# Forward Eiler
x_fe, t_fe = forward_euler(f, x0, Tf, h)

# Backward Eiler
x_be, t_be = backward_euler(f, x0, Tf, h)

# Runge-Kutta
x_rk4, t_rk4 = runge_kutta4(f, x0, Tf, h)

fe_err = np.sqrt(sum((x_fe[0, :] - x_an)**2)/len(t_an))
be_err = np.sqrt(sum((x_be[0, :] - x_an)**2)/len(t_an))
rk4_err = np.sqrt(sum((x_rk4[0, :] - x_an)**2)/len(t_an))

print("Результаты в конечной точке:")
print(f"Аналитическое решение: x({10}) = {x_an[-1]:.6f}")
print(f"Неявный Эйлер:   x({10}) = {x_be[0, -1]:.6f}, погрешность:
{be_err:.4f}")
print(f"Рунге-Кутта 4:   x({10}) = {x_rk4[0, -1]:.6f}, погрешность:
{rk4_err:.6f}")

# Plot results
plt.figure(figsize=(24, 8))

plt.plot(t_an, x_an, label=f'Analytic')
plt.plot(t_be, x_be[0, :], label=f'Backward Euler: {be_err:.4f}')
plt.plot(t_rk4, x_rk4[0, :], label=f'RK4: {rk4_err:.6f}')
plt.xlabel('t')
plt.ylabel('x')
plt.legend()

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