```
import numpy as np
import matplotlib.pyplot as plt
def ode_dynamics(x):

    a, b, c, d = 9.85, -9.64, -8.73, 6.04

    pos = x[0]
    vel = x[1]

    accel = (d - b * vel - c * pos) / a

    return np.array([vel, accel])
def analytical_solution(t, x0, v0):

    a, b, c, d = 9.85, -9.64, -8.73, 6.04

    b_norm = b/a
    c_norm = c/a
    d_norm = d/a
    discriminant = b_norm**2 - 4*c_norm
    if discriminant >= 0:
        r1 = (-b_norm + np.sqrt(discriminant)) / 2
        r2 = (-b_norm - np.sqrt(discriminant)) / 2

        x_particular = d/c
        C2 = (v0 - r1*(x0 - x_particular)) / (r2 - r1)
        C1 = (x0 - x_particular) - C2

        return C1 * np.exp(r1 * t) + C2 * np.exp(r2 * t) + x_particular
    else:
        alpha = -b_norm/2
        beta = np.sqrt(-discriminant)/2

        x_particular = d/c

        A = x0 - x_particular
        B = (v0 - alpha * A) / beta

        return np.exp(alpha * t) * (A * np.cos(beta * t) + B * np.sin(beta *
t)) + x_particular
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
x0 = np.array([0.1, 0.0])  # Initial state: [position, velocity]
Tf = 10.0
h = 0.01
t_analytical = np.linspace(0, Tf, 1000)  # плотная сетка для гладкости
x_analytical = analytical_solution(t_analytical, x0[0], x0[1])
v_analytical = np.gradient(x_analytical, t_analytical)  # численная
производная
```

```
x_fe, t_fe = forward_euler(ode_dynamics, x0, Tf, h)
x_be, t_be = backward_euler(ode_dynamics, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(ode_dynamics, x0, Tf, h)
plt.figure(figsize=(24, 8))
plt.subplot(1, 3, 1)
plt.plot(t_analytical, x_analytical, 'k-', linewidth=2, label='Analytical')
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('Time')
plt.ylabel('Position')
plt.legend()
plt.title('Position vs Time')
plt.subplot(1, 3, 2)
plt.plot(t_analytical, v_analytical, 'k-', linewidth=2, label='Analytical')
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('Time')
plt.ylabel('Velocity')
plt.legend()
plt.title('Velocity vs Time')
plt.subplot(1, 3, 3)
plt.plot(x_analytical, v_analytical, 'k-', linewidth=2, label='Analytical')
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('Position')
plt.ylabel('Velocity')
plt.legend()
plt.title('Phase Portrait')
plt.tight_layout()
plt.show()
```

Выводы

Результаты при Tf = 10.0 и h = 0.01:

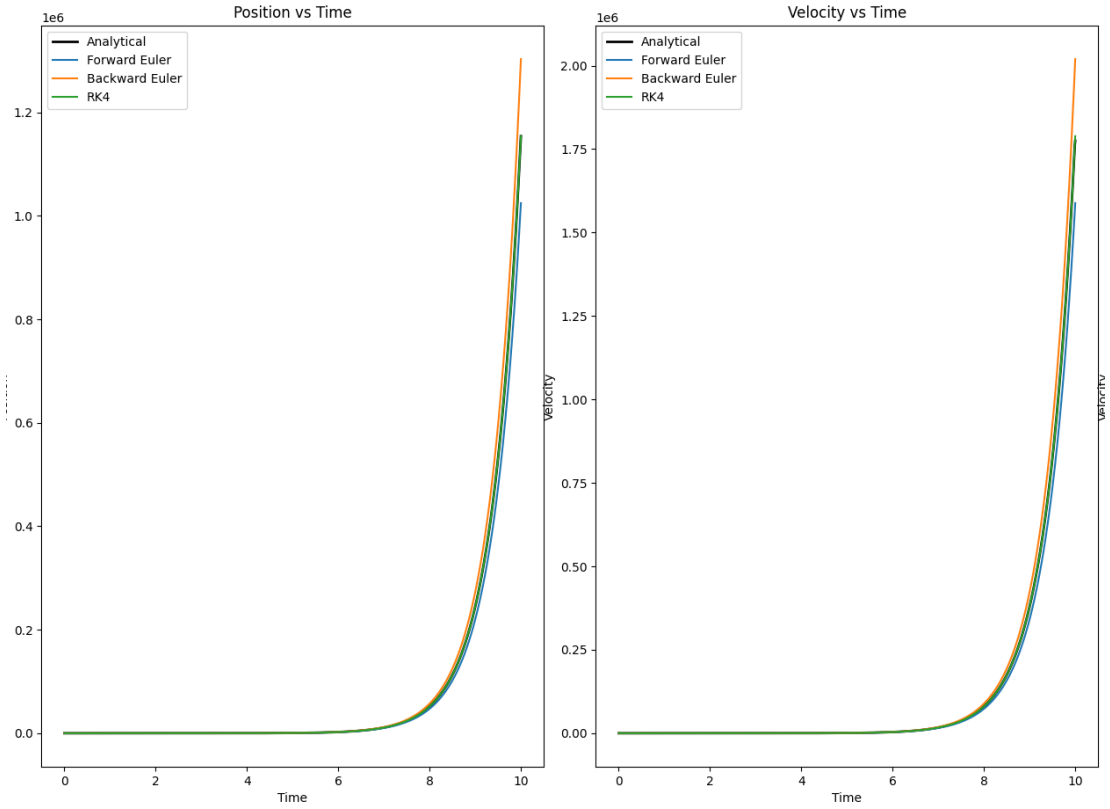

Рисунок 1 - Зависимость скорости и изменения положения от времени

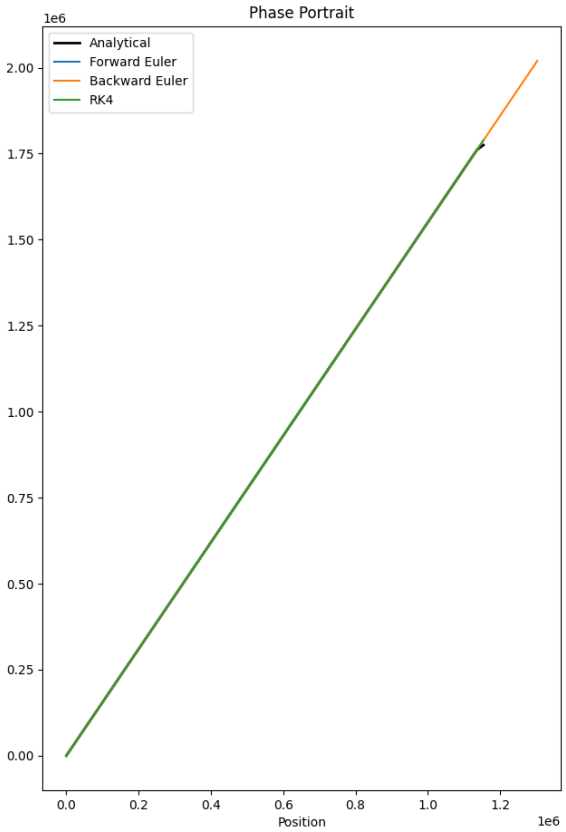

Рисунок 2 - Фазовый портрет

На основании результатов, полученных при выполнении кода, а также анализа графиков, представленных выше, можно сделать следующие выводы:

1. Для данной системы с параметрами и шагом 0.01 метод RK4 является предпочтительным, так как он обеспечивает точность и устойчивость и практически совпадает с аналитическим решением.

2. Методы Эйлера (явный и неявный) не подходят для точного моделирования этой системы при выбранном шаге. У явного Эйлера имеется значительное расхождение, а у неявного – сильное затухание, искажающее динамику.

3. Стоит отметить, что неявный Эйлер стабилен, но при этом вносит ложное демпфирование, что заставляет отдать предпочтение иному методу.