МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: «Прикладная математика и информатика»
Профиль подготовки: «Вычислительные методы и суперкомпьютерные технологии»

Отчет
по лабораторной работе №4
по курсу «Прикладная нелинейная динамика»

Выполнил:
студент группы 3823М1ПМвм
Бекетов Е.В.

Проверил:
д.ф.-м.н., доц., зав.каф. ПМ
Иванченко М.В.

Нижний Новгород
2024

1. Сравнение скорости сходимости методов Бернулли и Gillespie

Необходимо реализовать методы Бернулли и Gillespie с параметром $\gamma = 0.01$. Численно найти решение $\dot{x} = -\gamma x, \quad x(0) = x_0$, которое описывает поведение среднего количества молекул в процессе распада вещества, с помощью Рунге-Кутта 4 порядка и предложенных методов и провести анализ касаемо результатов.

Был написан скрипт и получено следующие оценки по времени при $x_0 = 100, \ \gamma = 0.01, \ t \in [0, 1000]$ с шагом РК4 $h = 0.0001$:

Методы	Время
Бернулли	0.088
Gillespie	0.004

Видно, что метод Gillespie гораздо быстрее, что согласуется с теорией. Посмотрим теперь на траектории на Рис. 1.

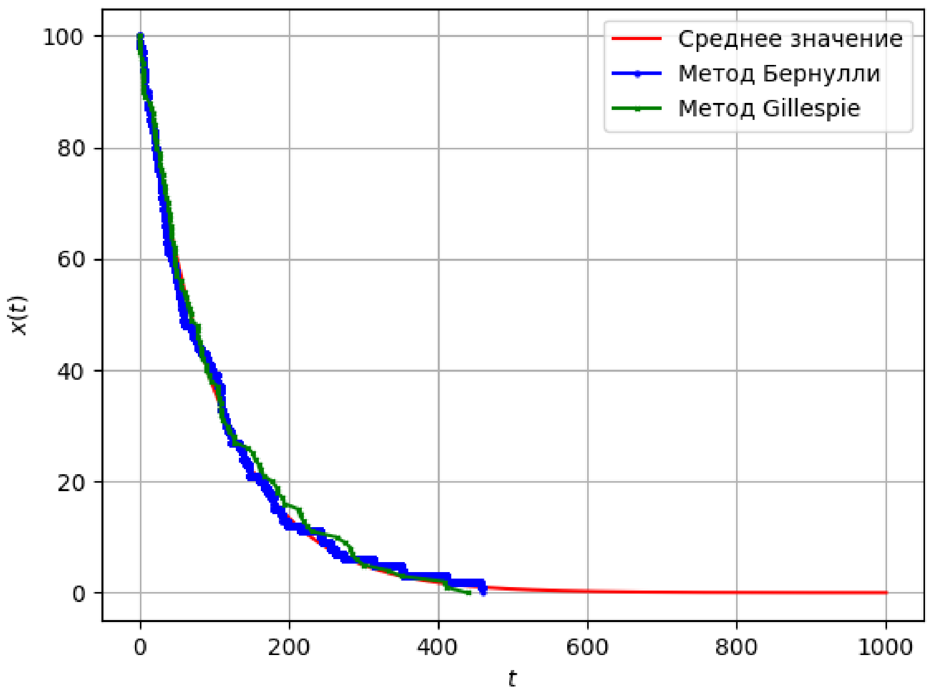

Рис. 1: Решения уравнения распада, полученные 3 методами.

Видно, что две случайные реализации процесса распада отличаются от численного решения и друг от друга. Случайные реализации заканчивают свои траектории раньше, потому как среднее число молекул становится равным нулю (дискретному числу), нежели решение РК4, которое описывает непрерывное решение.

В методе Бернулли условие $\Delta t = 0.01/a(0)$ обеспечивает для данного уравнения выполнение требования $\Delta t \ll 1/a(x)$ в течение всего процесса решения. График решения, полученный данным алгоритмом, имеет характерные ступени которые означают выполнение нескольких итераций подряд без уменьшения количества молекул. А метод Gillespie совершает итерации только в моменты распада, что как раз таки заставляет его работать на порядок быстрее.

2. Решение системы

Опробовав методы и убедившись в их корректности можно приступать к решению более сложной задачи. Во второй части работы следующая система была смоделирована с помощью метода РК4 ($h = 0.0001$) и Gillespie:

$$\begin{cases} \dot{m} = \frac{\alpha}{1+x^n} - m \\ \dot{x} = \beta m - \gamma x \end{cases}$$

При $\alpha = 20, \ \beta = \gamma = 1, \ n = 6, \ t \in [0, 50]$. С начальными условиями $m(0) = 30, \ x(0) = 15$. В результате было получено следующее решение Рис. 2 и Рис. 3.

Рис. 2: Решение первого уравнения системы методом РК4 и Gillespie.

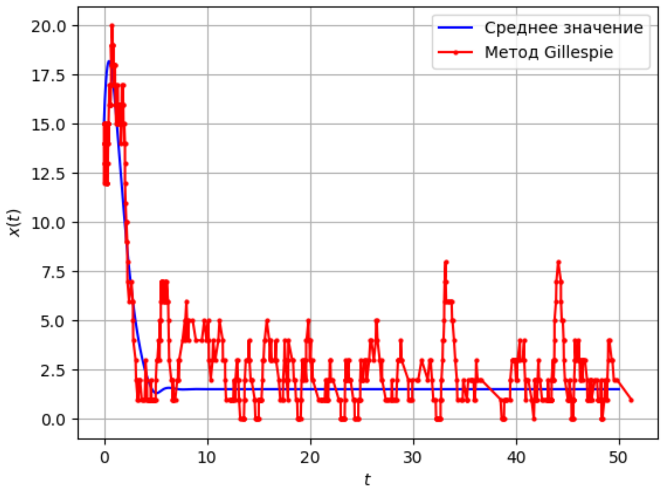

Рис. 3: Решение второго уравнения системы методом РК4 и Gillespie.

Видно что численное решение очень быстро приходит к состоянию равновесия $y = \frac{\alpha}{1+y^n}$, а реализация Gillespie показывает отчетливые колебания около с.р., что согласуется с теорией.

3. Приложение – Код программы

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

def RK4(func, start_t, end_t, init, h):
    num_steps = int((end_t - start_t) / h)
    x_path = [0.]*(num_steps + 1)
    t_path = [0.]*(num_steps + 1)
    x_path[0] = init
    t_path[0] = start_t

    for i in range(len(t_path) - 1):
        k1 = func(x_path[i], t_path[i])
        k2 = func(x_path[i] + 0.5*h*k1, t_path[i] + 0.5*h)
        k3 = func(x_path[i] + 0.5*h*k2, t_path[i] + 0.5*h)
        k4 = func(x_path[i] + h*k3, t_path[i] + h)
        x_path[i + 1] = x_path[i] + h*(k1 + 2*k2 + 2*k3 + k4) / 6
        t_path[i + 1] = t_path[i] + h

    return np.array(t_path), np.array(x_path)

def Bernoulli_solver(init, start_t, end_t, increment, decrement):
    time = start_t
    x = init.copy()
    x_s = [init]
    step = 1 / decrement[0](x) / 100
    times = [time]
    while time < end_t:
        a = decrement[0](x)
        r = np.random.random()
        if r < a*step:
            x[0] -= 1
        time += step
        x_s.append(x.copy())
        times.append(time)
        if x[0] <= 0:
            break

    return times, x_s

def Gillespie_solver(init, start_t, end_t, increment, decrement):
    time = start_t
    x = init.copy()
    x_s = [init]
    times = [time]
    while time < end_t:
        v_plus = []
        for rule in increment:
```

```
            v_plus.append(rule(x))
        v_minus = []
        for rule in decrement:
            v_minus.append(rule(x))
        a_0 = np.sum(np.array(v_plus + v_minus))
        if a_0 <= 0:
            break

        r1 = max(np.random.random(), 1e-12)
        tao = np.log(1 / r1) / a_0
        time += tao
        prob = np.array(v_plus + v_minus) / a_0
        idx = np.random.choice(2*len(init), p=prob)

        if idx >= len(increment):
            x[idx % len(decrement)] -= 1
        else:
            x[idx] += 1
        x_s.append(x.copy())
        times.append(time)

    return times, x_s

def main():
    np.random.seed(1)

    #! Первая часть
    init1 = 100
    gamma = 0.01
    time_st1 = 0
    time_fi1 = 1000
    t_rk4, x_rk4 = RK4(lambda x, t: -gamma*x, time_st1, time_fi1,
    init1, 1e-4)

    start = time.time()
    t_ber, x_ber= Bernoulli_solver([init1], time_st1, time_fi1,
    [lambda x: 0], [lambda x: gamma*x[0]])
    end = time.time()
    print('Время исполнения метода Бернулли ', end - start)

    start = time.time()
    t_gil, x_gil = Gillespie_solver([init1], time_st1, time_fi1,
    [lambda x: 0], [lambda x: gamma*x[0]])
    end = time.time()
    print('Время исполнения метода Gillespie ', end - start)

    plt.xlabel('$t$')
    plt.ylabel('$x(t)$')
    plt.plot(t_rk4, x_rk4, 'r-', label='Среднее значение')
    plt.plot(t_ber, np.array(x_ber).reshape(-1), 'b-o', markersize=2,
```

```
    label='Метод Бернулли')
    plt.plot(t_gil, np.array(x_gil).reshape(-1), 'g-x', markersize=2,
    label='Метод Gillespie')
    plt.grid()
    plt.legend()
    plt.savefig('Lab4_1.png')
    plt.clf()

    #! Вторая часть
    init2 = [30, 15]
    n = 6
    alpha = 20
    betta = 1
    time_st2 = 0
    time_fi2 = 50
    t_rk4, x_rk4 = RK4(lambda x, t: np.array([alpha/(1 + x[1]**n) - x[0],
    betta*(x[0] - x[1])]),
                       time_st2, time_fi2, np.array(init2), 1e-3)

    start = time.time()
    t_gil, x_gil = Gillespie_solver(init2, time_st2, time_fi2,
    [lambda x: alpha/(1 + x[1]**n), lambda x: betta*x[0]],
                            [lambda x: x[0], lambda x: betta*x[1]])
    end = time.time()
    print('Время исполнения метода Gillespie ', end - start)

    plt.xlabel('$t$')
    plt.ylabel('$m(t)$')

    plt.plot(t_rk4, np.array(x_rk4)[:, 0], 'b-',
    markersize=2, label='Среднее значение')
    plt.plot(t_gil, np.array(x_gil)[:, 0], 'r-o',
    markersize=2, label='Метод Gillespie')
    plt.grid()
    plt.legend()
    plt.savefig('Lab4_2.1.png')
    plt.clf()

    plt.xlabel('$t$')
    plt.ylabel('$x(t)$')
    plt.plot(t_rk4, np.array(x_rk4)[:, 1], 'b-',
    markersize=2, label='Среднее значение')
    plt.plot(t_gil, np.array(x_gil)[:, 1], 'r-o',
    markersize=2, label='Метод Gillespie')
    plt.grid()
    plt.legend()
    plt.savefig('Lab4_2.2.png')

if __name__ == '__main__':
    main()
```