Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ №4**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

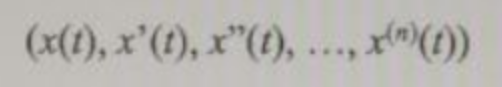
**Цель работы**

Составить программу, которая будет моделировать поведение среды, на которую влияет агент. Требуется определить предельные значения некоторых параметров, от которых зависит устойчивость системы при воздействии агента.

**Описание задачи**

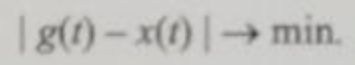
Устойчивость динамической среды с агентом

Имеется динамическая среда (объект управления), описываемая обыкновенным дифференциальным уравнением n-го порядка с постоянными

неотрицательными коэффициентами. Её состояние в каждый момент времени t задаётся вектором .

Агент (регулятор) управляет динамической средой, руководствуясь заданной извне целью g(t). Цель означает желаемое состояние

динамической среды, т.е. в идеале должно быть x(t) = g(t), но этого невозможно достичь ввиду инерционности среды

Тогда мы хотели бы, чтобы состояние x(t) как можно быстрее "догоняло" цель: .

Простейшее поведение агента было рассмотрено в лекции: агент производит действия action - вырабатывает управление средой в виде

, т.е. пропорциональное разности цели и состояния, но с некоторым запаздыванием tau >= 0

Задача

Провести моделирование функционирования динамической среды с агентом во времени, исследовать систему на устойчивость

Динамическая среда (объект управления) описывается обыкновенным дифференциальным уравнением 2-го порядка с постоянными положительными

коэффициентами. Датчики (устройства сбора информации в среде) имеют запаздывание величины tau, в результате чего агент

(регулятор) действует на основе устаревшей информации

При tau = 0 система устойчива. При небольшом увеличении tau устойчивость сохраняется, но при дальнейшем увеличении tau устойчивость

может потеряться. Определить зависимости величины граничного значения запаздывания tau, при которых система ещё устойчива, от других

числовых параметров  среды и агента (k). Функция цели g(t) = H(t) - функция Хевисайда

Построить графики зависимостей

Рекомендации

Время в задаче непрерывно, но для расчётов на компьютере нужно провести дискретизацию задачи и решения. Обозначим

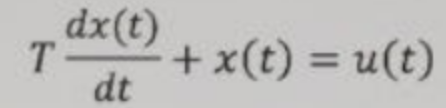
delta\_t - шаг по времени, например delta\_t = 0.01. Тогда состояния динамической среды изменяются в дискретные моменты

времени t\_i = 0, . Состояния среды в эти моменты времени

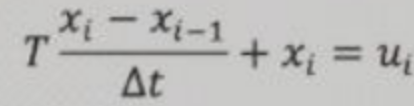
обозначим x\_i = x(t\_i), значения уравнения u\_i = u(t\_i)

Начальное состояние среды (0, 0, 0, ..., 0)

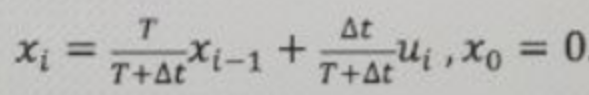
Дифференциальное уравнение заменяется разностным (как в дисциплине "Методы вычислений"). Например, для уравнения 1-го порядка (T > 0):



получаем



Отсюда выводим рекуррентное соотношение (при нулевых начальных условиях) для выполнения итераций:



Моделирование начинаем в момент времени t = 0 и заканчиваем в момент времени t = t\_stop, когда станет очевидно, что процесс сходится

(устойчивость) или что процесс расходится (неустойчивость)

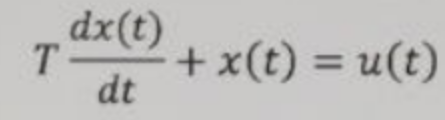
**Описание решения**

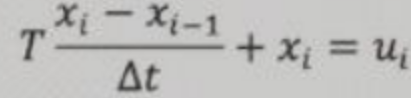
Для написания программы был выбран язык Python ввиду его удобства использования для математических вычислений.

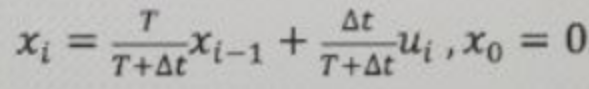
Был выбран функциональный подход, так как данную задачу весьма трудно разбить на какие-то классы с сохранением простоты и читаемости кода.

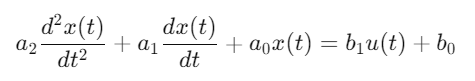
Было создано 3 файла: task.txt (текстовое описание задачи), main\_v2.py (первый вариант решения задачи, для дифференциального уравнения 1-го порядка), main\_v5.py (окончательный вариант решения задачи, для дифференциального уравнения 2-го порядка).

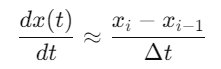
Теоретическая часть решения

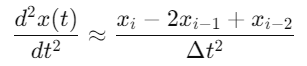
В рекомендациях к решению задачи был приведён пример дифференциального уравнения первого порядка, а также соответствующие ему разностное уравнение и рекуррентная формула:  




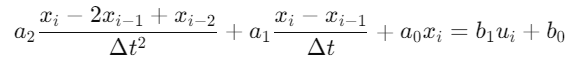


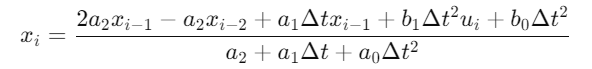
В соответствии с этими равенствами было получено дифференциальное уравнение второго порядка:  


Далее, методом аппроксимации были получены примерные замены производных:  




В соответствии с полученными заменами было составлено следующее уравнение:



И группировка членов для определения x\_i:  


Было решено пренебречь параметрами a\_0 (этот коэффициент в примере равняется 1) и b\_0 (пренебрегаем, потому что отсутствует соответствующий ему a\_0), так как они оказывают не такое сильное влияние на систему, как прочие параметры.

Практическая часть решения

Далее будет описан код, относящийся к конечной версии реализации программы

Используемые функции:

calculate\_g\_t(): данная функция определяет значения целевой функции (функции Хэвисайда) в зависимости от времени симуляции (здесь сделано допущение: классическая функция Хэвисайда принимает значения = 0 при отрицательных значениях параметра, а = 1 при значениях > 0, однако в условии задачи было сказано, что симуляция начинается с момента времени = 0, так что наша функция g\_t возвращает 0 при времени < половине общего времени симуляции, а в остальных случаях 1);

calculate\_u\_t(): данная функция вычисляет воздействие агента на среду;

calculate\_interpolated\_x\_i(): данная функция реализует интерполяцию по отношению к функции x(t) (в задаче используется определение x(t - tau), и в ситуации, когда у нас значение t – tau не равняется ни одному вычисленному на предыдущих шагах значению t + delta\_t, для корректного получения значения используется именно интерполяция, а не обращение к ближайшему целому значению t + delta\_t);

calculate\_x\_i(): данная функция реализует вычисление нового состояния системы x\_i;

check\_stability(): данная функция вычисляет разницу между текущим целевым состоянием среды и полученным после воздействия агента. В коде эта функция реализуется в связке с параметром max\_error\_threshold, который определяет допустимое значение ошибки, при котором система считается устойчивой;

simulation(): данная функция выполняет одну симуляцию для комбинации параметров системы (tau, k, T\_1 (a\_2), T\_2 (a\_1), D (b\_1)) и возвращает максимальную величину ошибки;

refine\_simulate(): данная функция выполняет сужение диапазонов параметров системы (tau, k, T\_1 (a\_2), T\_2 (a\_1), D (b\_1)) до тех пор, пока ни разу не будет выполнена симуляция без превышения ошибки устойчивости, или пока не закончатся шаги приближения.

Основная логика программы заключается в вызове последней описанной функции, которая возвращает 6 списков (5 диапазонов параметров и список максимальных ошибок для комбинаций параметров).

Далее, для этих данных выполняется построение графика поведения системы, однако здесь возникли некоторые трудности:

1) Построение графика от 6 параметров не представляется возможным в привычном понимании, так что был выбран 3-мерный график в комбинации с цветовой картой значений точек результата;

2) На вышеописанном графике не получилось отобразить все 5 параметров вместе с величиной ошибки по следующей причине: в библиотеке matplotlib был найден баг, из-за которого при построении графика с цветовыми точками цвета этих самых точек изменяются при изменении угла обзора графика (Рисунок 1, Рисунок 2):

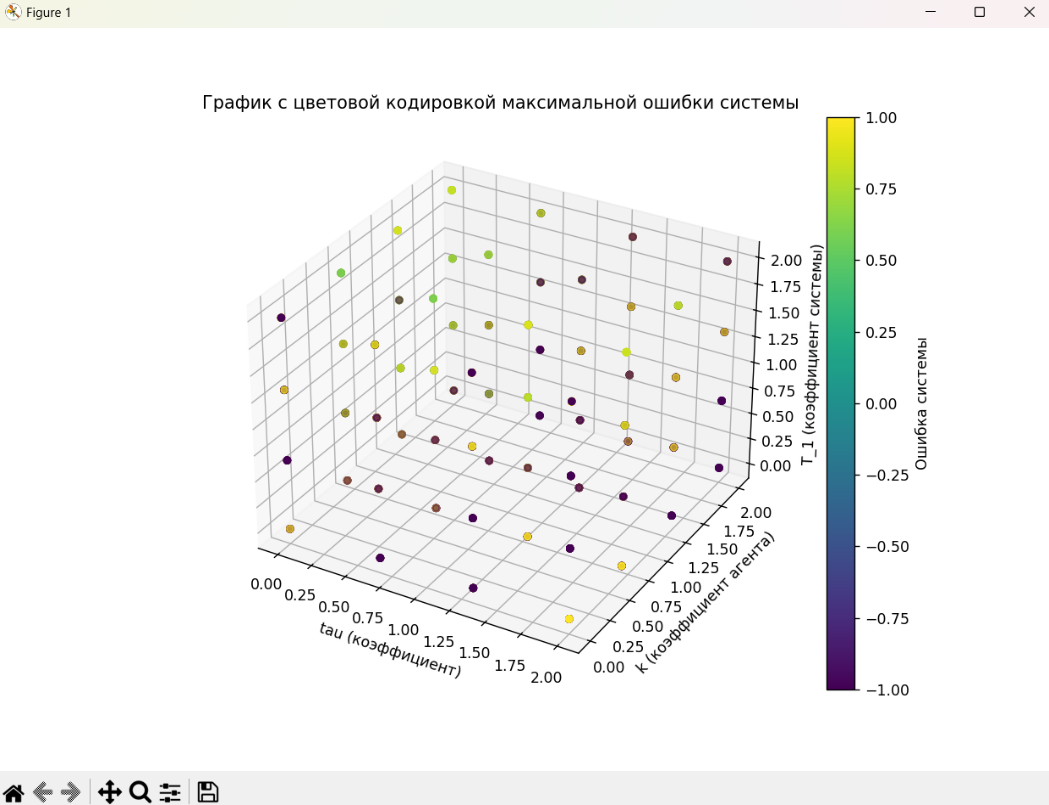


Рисунок 1 – график, полученный после прогонки всех комбинаций параметров

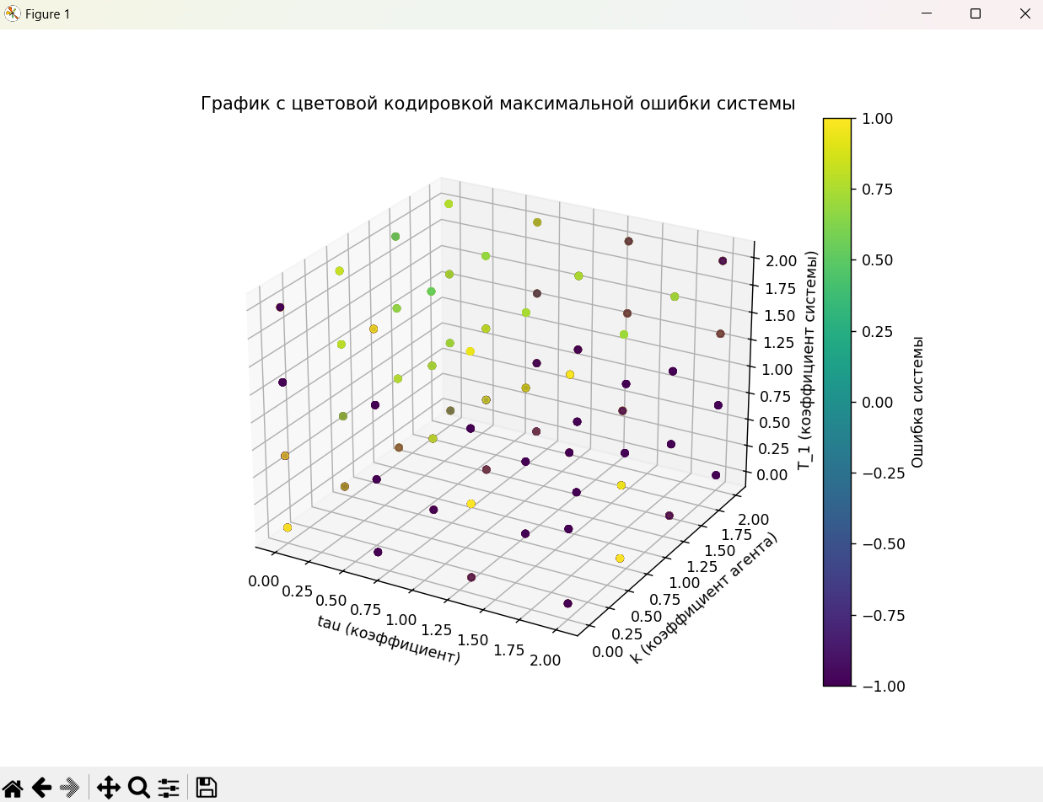


Рисунок 2 – тот же самый график, но с изменёнными точками из-за изменённого угла обзора

На приведённых выше рисунках видно изменение цветов, например, по точке с координатами (tau=2.00, k=0.00, T\_1=0.00): была зелёной, а стала синей.

Было проведено расследование, в результате которого выяснилось, что подобная ошибка была найдена ещё 11 лет назад (1 ноября 2013 года):

[Axes3D scatter changes the color in version 1.2.1 during rotation · Issue #2564 · matplotlib/matplotlib · GitHub](https://github.com/matplotlib/matplotlib/issues/2564/)

Был составлен актуальный отчёт об ошибке и отправлен разработчикам (9 октября 2024 года):  
[[Bug]: Changing the color of points in a 3-dimensional graph when changing the viewing angle · Issue #28958 · matplotlib/matplotlib · GitHub](https://github.com/matplotlib/matplotlib/issues/28958)

Выяснилось, что скорее всего, ошибка связана с некорректным поведением маппинга цветов точек цветовой картой. При составлении графика такие значения должны вычисляться 1 раз и вызываться в дальнейшем, при изменении графика (например, при повороте), а реальным поведением является вычисление таких значений при каждом изменении графика.

Из-за вышеописанной ошибки в качестве итогового представлен график, на котором изображено поведение системы в зависимости от 3 переменных (tau, k, T) и ошибки системы (Рисунок 3):

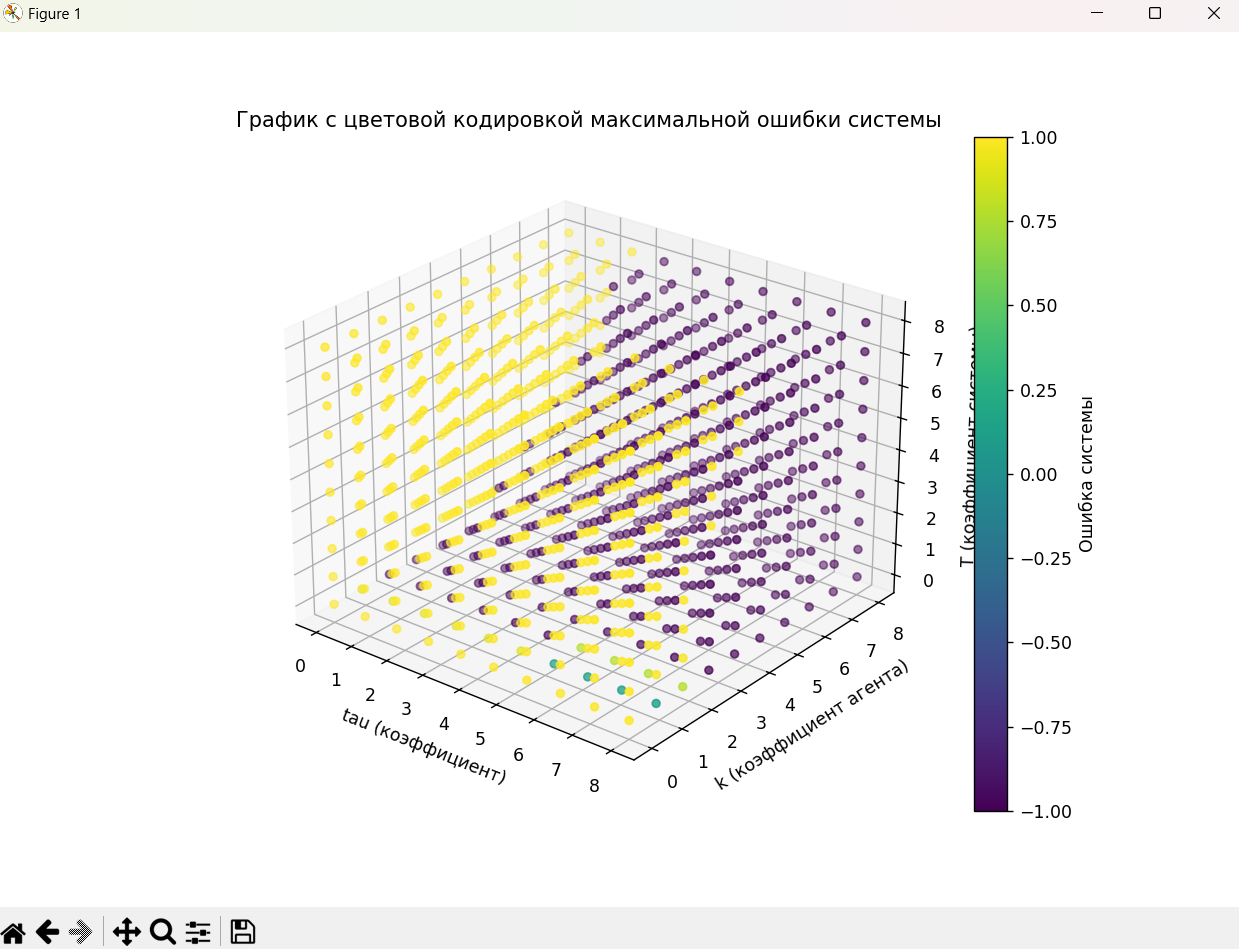


Рисунок 3 – поведение системы относительно параметров tau, k, T

На рисунке 3 изображён график, корректный для системы от 3 параметров (дифференциальное уравнение первого порядка, т.е. от 3 параметров (tau, k, T)), хотя код этого графика идентичен коду графика, который отображает тоже 3 параметра (tau, k, T\_1) для системы второго порядка (неясно, почему вышепоказанный график строится корректно).

А на рисунке (Рисунок 4) показан двумерный график для системы с 5-ю параметрами:

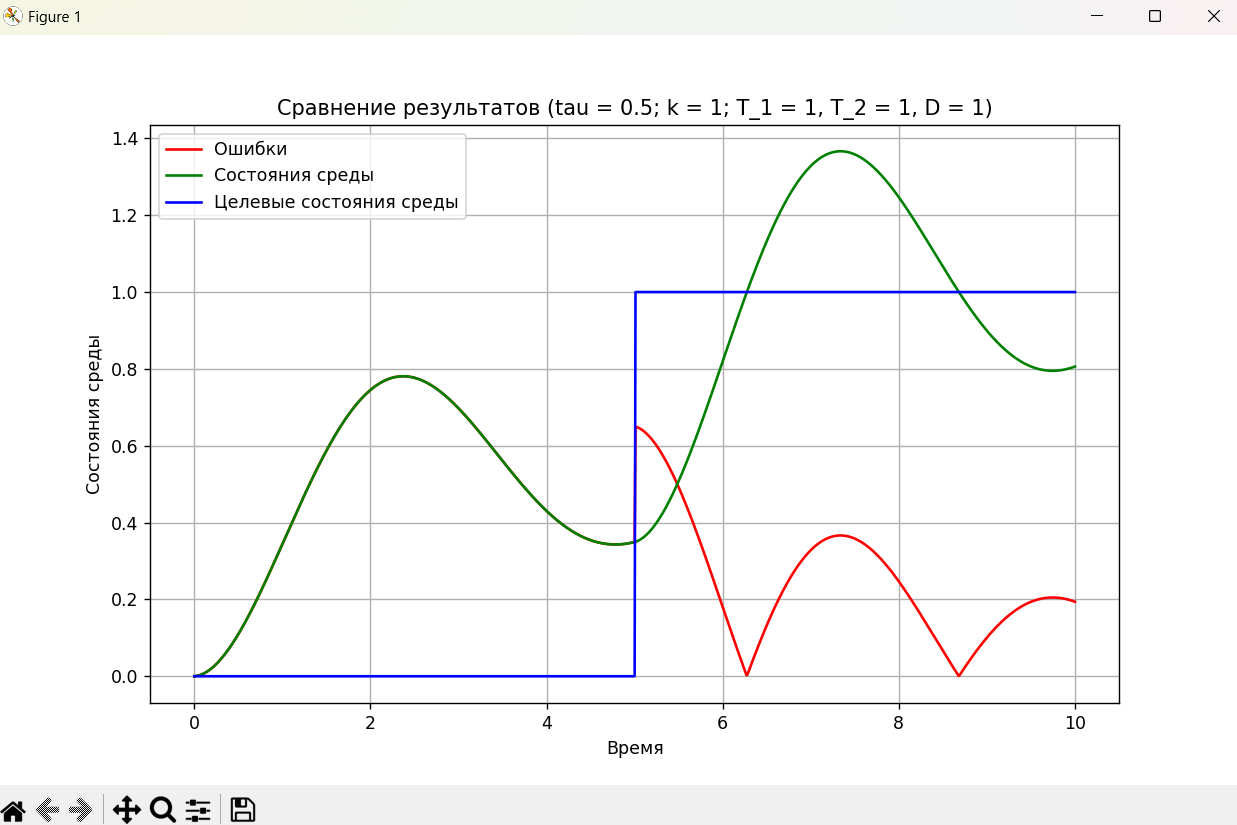


Рисунок 4 – устойчивое поведение системы для tau = 0.5

**Код программы**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

def calculate\_g\_t(*t*: float, *t\_stop*: int) -> int:

    """

    Функция, которая вычисляет целевое состояние среды в зависимости от времени

    Args:

        t (float): Текущее время

        t\_stop (int): Лимит симуляции

    Returns:

        result (int): Текущее целевое состояние среды

    """

    mid = *t\_stop* / 2

    return 0 if 0 <= *t* <= mid else 1

def calculate\_u\_t(*k*: float, *g\_t*: int, *x\_t\_tau*: float) -> float:

    """

    Функция, которая вычисляет воздействие агента на среду

    Args:

        k (float): Коэффициент агента

        g\_t (int): Целевое состояние среды

        x\_t\_tau (float): Предыдущее состояние среды

    Returns:

        result (float): Воздействие агента на среду

    """

    return *k* \* (*g\_t* - *x\_t\_tau*)

def calculate\_interpolated\_x\_i(*all\_time*: list, *all\_x\_i*: list, *t\_minus\_tau*: float) -> float:

    """

    Функция, которая реализует линейную интерполяцию для вычисления значения x(t-tau) вместо подбора ближайшего значения

    Args:

        all\_time (list): Список всех моментов времени симуляции

        all\_x\_i (list): Список всех значений функции x(t)

        t\_minus\_tau (float): Текущее значение t-tau, для которого ищем значение функции

    Returns:

        result (float): Значение функции x(t) для заданного t-tau

    """

    if *t\_minus\_tau* <= *all\_time*[0]:

        return *all\_x\_i*[0]

    return np.interp(*t\_minus\_tau*, *all\_time*, *all\_x\_i*)

def calculate\_x\_i(*big\_T\_1*: float, *big\_T\_2*: float, *big\_D*: float, *x\_i\_prev*: float, *x\_i\_prev\_2*: float, *delta\_t*: float, *u\_i*: float) -> float:

    """

    Функция, которая рассчитывает новое состояние среды

    Args:

        big\_T\_1 (float): Коэффициент среды

        big\_T\_2 (float): Коэффициент среды

        big\_D (float): Коэффициент воздействия

        x\_i\_prev (float): Предыдущее состояние среды

        x\_i\_prev\_2 (float): Предыдущее перед предыдущим состояние среды

        delta\_t (float): Временной шаг

        u\_i (float): Воздействие агента на среду

    Returns:

        result (float): Новое состояние среды

    """

    return ((2 \* *big\_T\_2* \* *x\_i\_prev*) - (*big\_T\_2* \* *x\_i\_prev\_2*) + (*big\_T\_1* \* *delta\_t* \* *x\_i\_prev*) + (*big\_D* \* (*delta\_t*\*\*2) \* *u\_i*) + (*delta\_t*\*\*2)) / (*big\_T\_2* + (*big\_T\_1* \* *delta\_t*) + *delta\_t*\*\*2)

def check\_stability(*g\_t*: int, *x\_t*: float) -> float:

    """

    Функция, вычисляющая разность между целевым и текущим состоянием среды

    Args:

        g\_t (int): Целевое состояние

        x\_t (float): Текущее состояние

    Returns:

        result (float): Разность целевого и текущего состояния

    """

    return abs(*g\_t* - *x\_t*)

def simulation(*t*: float, *delta\_t*: float, *t\_stop*: int, *tau*: float, *k*: float, *big\_T\_1*: float, *big\_T\_2*: float, *big\_D*: float, *max\_error\_threshold*: float) -> float:

    """

    Функция, которая проводит симуляцию системы до времени t\_stop

    Args:

        t (float): Начальный момент времени

        delta\_t (float): Временной шаг

        t\_stop (int): Конец симуляции

        tau (float): Величина tau

        k (float): Параметр агента

        big\_T\_1 (float): Параметр среды

        big\_T\_2 (float): Параметр среды

        big\_D (float): Параметр воздействия

        max\_error\_threshold (float): Порог ошибки устойчивости системы

    Returns:

        result (float): Максимальная ошибка системы

    """

*#заполняем массивы начальными данными для того, чтобы они все были одной длины*

    all\_time = [0]

    all\_g\_t = [0]

    all\_x\_i = [0, 0]

    all\_stability = [0]

    while *t* < *t\_stop*:

        all\_time.append(*t*)

        cur\_g\_t = calculate\_g\_t(*t*, *t\_stop*)

        all\_g\_t.append(cur\_g\_t)

        t\_minus\_tau = *t* - *tau*

        cur\_x\_t\_minus\_tau = calculate\_interpolated\_x\_i(all\_time, all\_x\_i, t\_minus\_tau)

        cur\_u\_t = calculate\_u\_t(*k*, cur\_g\_t, cur\_x\_t\_minus\_tau)

        cur\_x\_i = calculate\_x\_i(*big\_T\_1*, *big\_T\_2*, *big\_D*, all\_x\_i[-1], all\_x\_i[-2], *delta\_t*, cur\_u\_t)

        all\_x\_i.append(cur\_x\_i)

        cur\_stability = check\_stability(cur\_g\_t, cur\_x\_i)

        all\_stability.append(cur\_stability)

        if cur\_stability > *max\_error\_threshold*:

            return -1

*t* += *delta\_t*

    return max(all\_stability)

def refine\_simulate(*t*: float, *delta\_t*: float, *t\_stop*: int, *prev\_tau\_list*: list, *prev\_k\_list*: list, *prev\_big\_T\_1\_list*: list, *prev\_big\_T\_2\_list*: list, *prev\_big\_D\_list*: list, *max\_error\_threshold*: float, *step*: float) -> list:

    """

    Функция, реализующая сужение диапазона значений tau, k, big\_T\_1, big\_T\_2, big\_D для поиска оптимальных значений для устойчивости системы

    Args:

        t (float): Начальный момент времени

        delta\_t (float): Временной шаг

        t\_stop (int): Конец симуляции

        prev\_tau\_list (list): Список предыдущих значений tau

        prev\_k\_list (list): Список предыдущих значений k

        prev\_big\_T\_list (list): Список предыдущих значений big\_T

        max\_error\_threshold (float): Порог ошибки устойчивости системы

        step (float): Шаг уменьшения пределов параметров

    Returns:

        list, list, list, list, list, list (list): 6 списков, описывающие 5 параметров (tau, k, big\_T\_1, big\_T\_2, big\_D) и максимальные ошибки для каждой комбинации параметров

    """

    tau\_list = *prev\_tau\_list*

    k\_list = *prev\_k\_list*

    big\_T\_1\_list = *prev\_big\_T\_1\_list*

    big\_T\_2\_list = *prev\_big\_T\_2\_list*

    big\_D\_list = *prev\_big\_D\_list*

    for i in range(10):

        print(f"{i + 1} попытка сужения диапазонов значений")

        tau\_critical = []

        tau\_stable = []

        x1, x2, y1, y2, z, colors = [], [], [], [], [], []

        error\_occurred = False

        print("Начало симуляции:")

        for tau in tau\_list:

            for k in k\_list:

                for big\_T\_1 in big\_T\_1\_list:

                    for big\_T\_2 in big\_T\_2\_list:

                        for big\_D in big\_D\_list:

                            max\_stability = simulation(*t*, *delta\_t*, *t\_stop*, tau, k, big\_T\_1, big\_T\_2, big\_D, *max\_error\_threshold*)

                            if max\_stability == -1:

                                tau\_critical.append((tau, k, big\_T\_1, big\_T\_2, big\_D))

                                error\_occurred = True

                            else:

                                tau\_stable.append((tau, k, big\_T\_1, big\_T\_2, big\_D))

                            x1.append(tau)

                            x2.append(k)

                            y1.append(big\_T\_1)

                            y2.append(big\_T\_2)

                            z.append(big\_D)

                            colors.append(max\_stability)

            print(f"Симуляция для tau = {tau} закончена")

        if error\_occurred and tau\_stable:

            tau\_min = min([t[0] for t in tau\_stable])

            tau\_max = max([t[0] for t in tau\_stable]) - *step*

            k\_min = min([t[1] for t in tau\_stable])

            k\_max = max([t[1] for t in tau\_stable]) - *step*

            big\_T\_1\_min = min([t[2] for t in tau\_stable])

            big\_T\_1\_max = max([t[2] for t in tau\_stable]) - *step*

            big\_T\_2\_min = min([t[3] for t in tau\_stable])

            big\_T\_2\_max = max([t[3] for t in tau\_stable]) - *step*

            big\_D\_min = min([t[4] for t in tau\_stable])

            big\_D\_max = max([t[4] for t in tau\_stable]) - *step*

            tau\_list = np.linspace(tau\_min, tau\_max, 4)

            k\_list = np.linspace(k\_min, k\_max, 4)

            big\_T\_1\_list = np.linspace(big\_T\_1\_min, big\_T\_1\_max, 4)

            big\_T\_2\_list = np.linspace(big\_T\_2\_min, big\_T\_2\_max, 4)

            big\_D\_list = np.linspace(big\_D\_min, big\_D\_max, 4)

            print(f"Новые диапазоны: tau [{tau\_min}, {tau\_max}], k [{k\_min}, {k\_max}], T\_1 [{big\_T\_1\_min}, {big\_T\_1\_max}], T\_2 [{big\_T\_2\_min}, {big\_T\_2\_max}], D [{big\_D\_min}, {big\_D\_max}]")

        elif not tau\_stable:

            print("Все комбинации параметров приводят к ошибке")

            break

        else:

            print("Система устойчива при данных параметрах")

            break

    return x1, x2, y1, y2, z, colors

t = 0.00

delta\_t = 0.01

t\_stop = 10.00

tau\_list = np.linspace(0.00, 10.0, 4)

k\_list = np.linspace(0.00, 10.0, 4)

big\_T\_1\_list = np.linspace(0.00, 10.0, 4)

big\_T\_2\_list = np.linspace(0.00, 10.0, 4)

big\_D\_list = np.linspace(0.00, 10.0, 4)

max\_error\_threshold = 1

step = 1

x1, x2, y1, y2, z, colors = refine\_simulate(t, delta\_t, t\_stop, tau\_list, k\_list, big\_T\_1\_list, big\_T\_2\_list, big\_D\_list, max\_error\_threshold, step)

x1 = np.array(x1)

x2 = np.array(x2)

y1 = np.array(y1)

y2 = np.array(y2)

z = np.array(z)

colors = np.array(colors)

fig = plt.figure(*figsize*=(10, 7))

ax = fig.add\_subplot(111, *projection*="3d")

img = ax.scatter(x1, x2, y1, *c*=colors, *cmap*=cm.viridis)

ax.set\_xlabel("tau (коэффициент)")

ax.set\_ylabel("k (коэффициент агента)")

ax.set\_zlabel("T\_1 (коэффициент системы)")

plt.title("График с цветовой кодировкой максимальной ошибки системы")

fig.colorbar(img, *ax*=ax, *label*="Ошибка системы")

plt.show()

**Выводы программы**

