МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 25

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

к.т.н., доцент		А. А. Бурков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ)

МОДЕЛИРОВАНИЕ АЛГОРИТМА СЛУЧАЙНОГО МНОЖЕСТВЕННОГО ДОСТУПА ПРИ ПЕРЕМЕННОЙ ИНТЕНСИВНОСТИ ВХОДНОГО ПОТОКА

по дисциплине: Основы построения инфокоммуникационных систем и сетей

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 2155		И. А. Макаренко
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1. Цель курсовой работы

Целью курсовой работы является создание имитационной модели для системы массового обслуживания, работающей по алгоритму случайного множественного доступа, при переменной интенсивности входного потока заявок. Так же целью работы является изучение характеристик системы, полученных на основе моделирования.

2. Описание моделируемой системы

2.1. Обслуживание пользователей

В качестве алгоритма случайного множественного доступа используется вероятностный алгоритм ALOHA, а также адаптивный вероятностный алгоритм ALOHA.

Система обслуживания (обслуживающий прибор) является синхронной системой, то есть выделяет (закрепляет) ресурс канала под определенного абонента в соответствии с единым временем. Если в канале передает только один абонент, то передача по каналу считается успешной. Если в канале оказалось два и более предающих пользователей, то в канале происходит конфликт и ни один из пользователей не предает сообщение. Если же в канале не оказалось абонентов, готовых передавать сообщение, то такая ситуация обозначается как пусто.

Вероятностный алгоритм ALOHA работает по следующему принципу: абонент, у которого появилось готовое для передачи сообщение, выставляет постоянную вероятность передачи в окне (для исследуемой системы используется вероятность $p = \frac{1}{M}$, где M - количество абонентов), то есть если

$$p_{\text{передачи}} = p - \text{абонент пытается передать сообщение}$$

$$p_{\text{передачи}} = 1 - p - \text{абонент ничего не передает и ждет следующее окно}$$

В ситуации “успех” (абонент оказался единственным передающим) запрос абонента обрабатывается системой. В ситуации “конфликт” ни один из

абонентов не обслуживается, все пользователи ожидают следующего окна для новой попытки передачи.

На рисунке 1 приведена схема системы, обслуживающей M абонентов. Обозначения на рисунке:

1) M – число абонентов в системе
2) λ – входная интенсивность потока; $\frac{\lambda}{M}$ – интенсивность входного потока одного абонента, соответственно
3) p – вероятность передачи сообщения абонентом
4) μ – время обслуживания абонента (время окна системы)

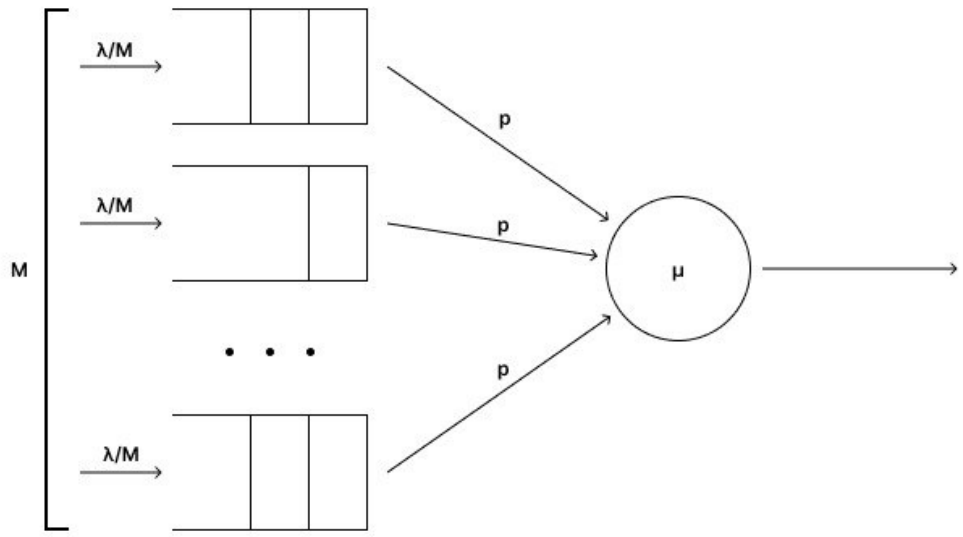

Рисунок 1 – Схема обслуживающего прибора с M абонентами.

Адаптивный вероятностный алгоритм ALOHA является модификацией обычного вероятностного алгоритма ALOHA. В алгоритме вероятность передачи сообщения абонентом является не постоянной величиной (например, 0.5), а динамической. Вероятность передачи сообщения абонентом изменяется в каждом кадре в соответствии с наблюдаемым событием в окне: “успех”, “конфликт” или “пусто”. Соответственно каждый пользователь, имеющий готовое для передачи сообщение начинает прослушивать канал, изменяя свою

вероятность передачи и пытается передать сообщение при новом значении вероятности. Каждый новоприбывший в систему пользователь имеет начальную вероятность передачи равную $p = 1$ (пользователь не прослушивал канал ранее). Вероятность передачи абонента изменяется по следующему принципу:

$$p_{t+1} = \begin{cases} \max\left(\frac{1}{M}, \frac{p_t}{2}\right), \text{при ситуации "конфликт"} \\ p_t, \text{при ситуации "успех"} \\ \min(1, 2p_t), \text{при ситуации "пусто"} \end{cases}$$

2.2. Переменная интенсивность входного потока

Для имитации перемены интенсивности входного потока в системе имеется две величины входного потока: λ_{max} и λ_{min} которые “переключаются” между собой в каждом окне с определенной вероятностью, определяемой соотношением количества окно с λ_{max} к окнам с λ_{min}. На рисунке 2 приведен пример конечного автомата состояний интенсивности входного потока. На рисунке имеются следующие обозначения:

1) $p_{\lambda max}$ – вероятность появления максимальной интенсивности в окне
2) $p_{\lambda max}$ – вероятность появления минимальной интенсивности в окне
3) λ_{max} и λ_{min} – максимальная и минимальная интенсивности, соответственно

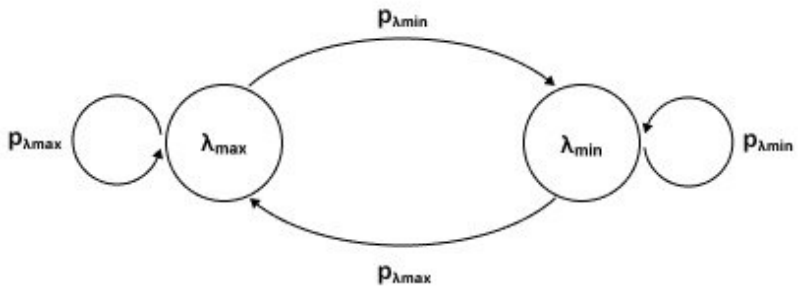

Рисунок 2 – Автомат состояний интенсивности входного потока.

2.3. Генерация заявок с помощью Пуассоновского потока

Для генерации новых заявок пользователей используется простой Пуассоновский поток заявок, определяемый по формуле:

$$P_k(\lambda, t_\Delta) = \frac{(\lambda t_\Delta)^k}{k!} e^{-\lambda t_\Delta}$$

где

λ – интенсивность входного потока пользователя;

t_Δ – время обслуживания одного абонента (время кадра)

k – количество заявок, которое может возникнуть на промежутке t_Δ

Для каждой интенсивности входного потока формируется дискретная случайная величина на основе вычисленных вероятностей появления сообщений. На рисунке 3 приведен пример того, как может выглядеть дискретная случайная величина для λ_{max} и λ_{min}.

Рисунок 3 – Пример дискретных случайных величин для разных интенсивностей.

Каждое значение (граница) дискретной случайной величины вычисляется следующим образом:

$$p_k = \sum_{i=0}^{k} p_i(\lambda, t_\Delta)$$

Соответственно, в каждом окне пользователь генерирует случайную величину, распределенную на отрезке от 0 до 1, которая будет соответствовать тому количеству новых сообщений в текущем окне, в вероятность которого попало сгенерированное значение.

3. Список допущений рассматриваемой модели

Для рассмотрения модели со случайным множественным доступом вводится ряд допущений:

1) Все сообщения у всех абонентов имеют одинаковую длину, время передачи одного сообщения принято за единицу времени. Все время передачи по каналу разбито на окна, длительность окна соответствует времени передачи одного сообщения. Абоненты точно знают моменты разделения и могут начать передачу только в начале окна.
2) В окне может быть 3 события:
    a. Событие «Конфликт». В окне одновременно передают два абонента или больше. Считается, что из-за наложения сигналов сообщения полностью искажаются и не могут быть приняты правильно.
    b. Событие «Успех». В окне передает один абонент, в этом случае считается, что абонент успешно передает сообщение.
    c. Событие «Пусто». В окне никто не передает.
3) Абоненты наблюдают выход канала в конце окна и достоверно определяют, какое из трех событий произошло.
4) В системе имеется M абонентов. В среднем у всех абонентов в одну единицу времени возникает λ сообщений (интенсивность входного потока) (Пуассоновский входной поток с параметром λ).

Интенсивность входного потока у всех абонентов в системе одинакова и у каждого абонента она равна $\frac{\lambda}{M}$.

Простейший Пуассоновский поток заявок имеет следующие особенности:

1) Отсутствие последствия – заявки приходят независимо.
2) Стационарность – вероятность поступления заявки за какое-то время зависит только от t_Δ и λ.

4. Описание моделирующей программы в виде блок-схемы

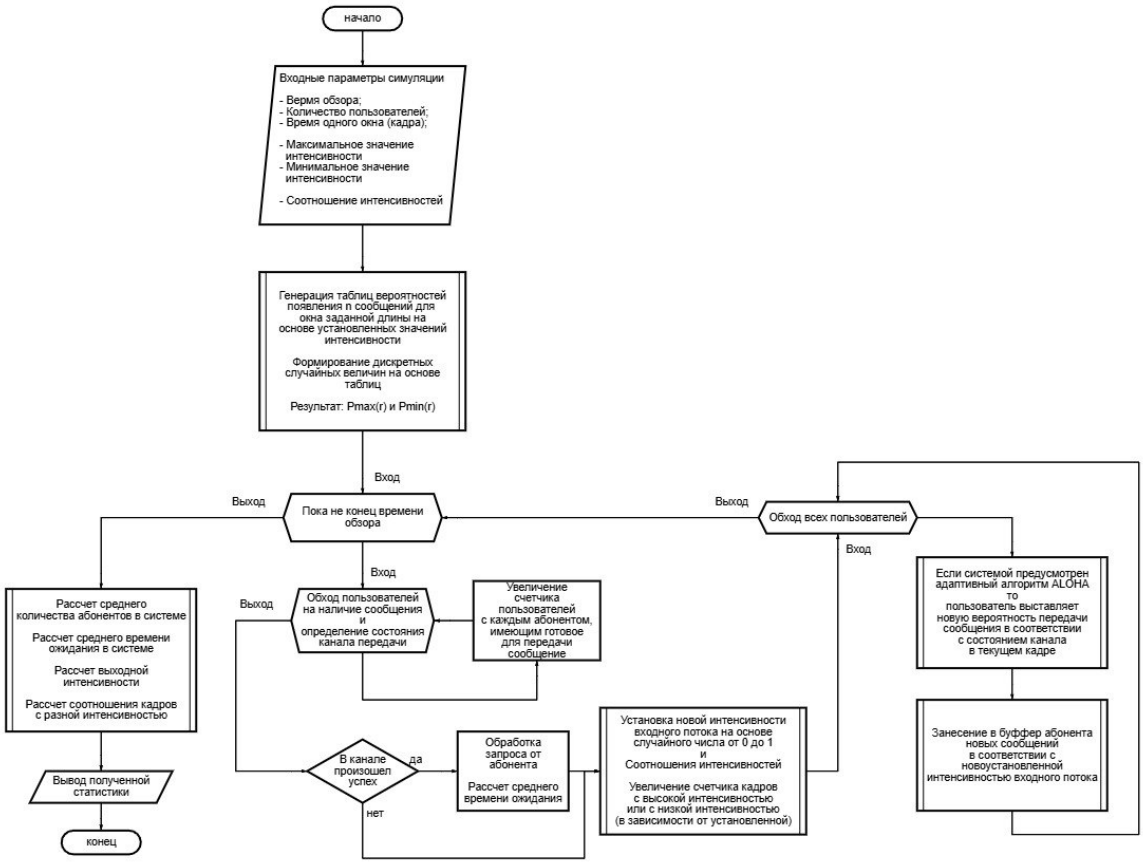

Рисунок 4 – Блок-схема моделирующей программы

5. Графики зависимостей характеристики системы и результаты вычислений в отдельных тестах

Основной задачей исследования моделируемой системы является сравнение результатов зависимостей: средней задержки от интенсивности входного потока; среднего числа абонентов в системе от интенсивности входного потока; интенсивность выходного потока от входного потока. Для каждого графика рассматривается система с 20-ю пользователями и временем обзора 1 млн. окон. Соотношения интенсивностей были выбраны 3 к 10, где меньшее соотношение относится к статичной интенсивности (неизменяющейся), статичная низкая интенсивность равна 0.01, статичная высокая интенсивность равна 0.8.

На рисунках 5-7 показаны графики зависимостей для вероятностного алгоритма, на рисунках 8-10 для адаптивного алгоритма и 11-13 тот же алгоритм, но на приближении в зоне пересечения графиков.

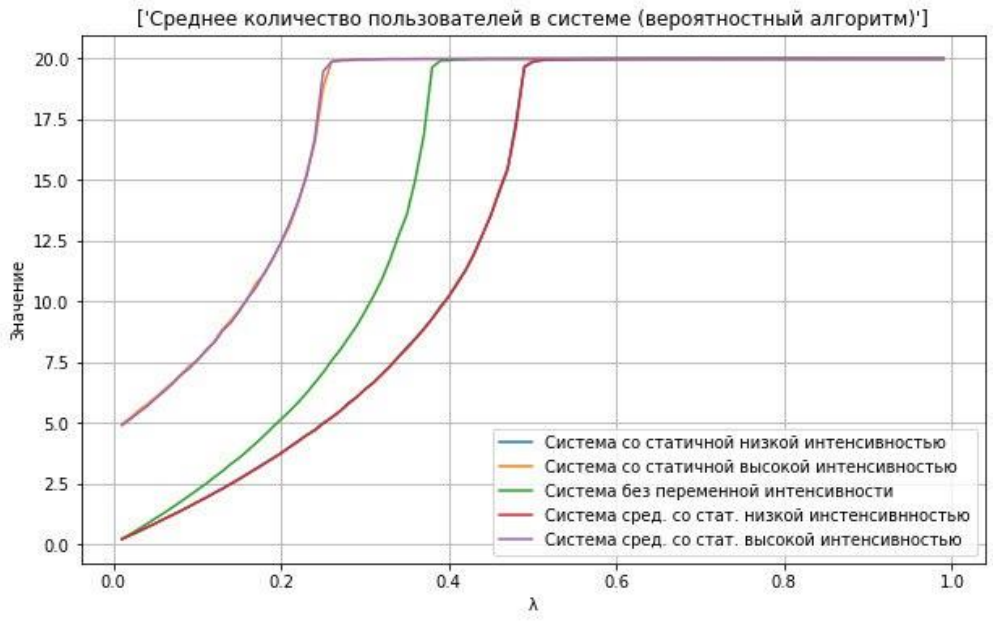

Рисунок 5 – График среднего количества пользователей для вероятностного алгоритма при переменной интенсивности входного потока

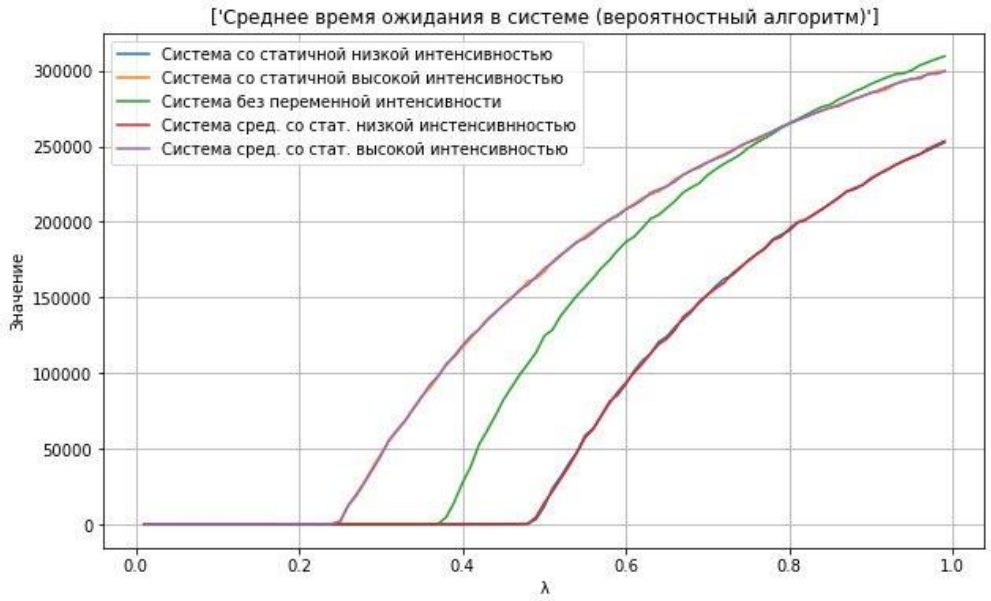

Рисунок 6 – График среднего времени ожидания (задержки) для вероятностного алгоритма при переменной интенсивности входного потока

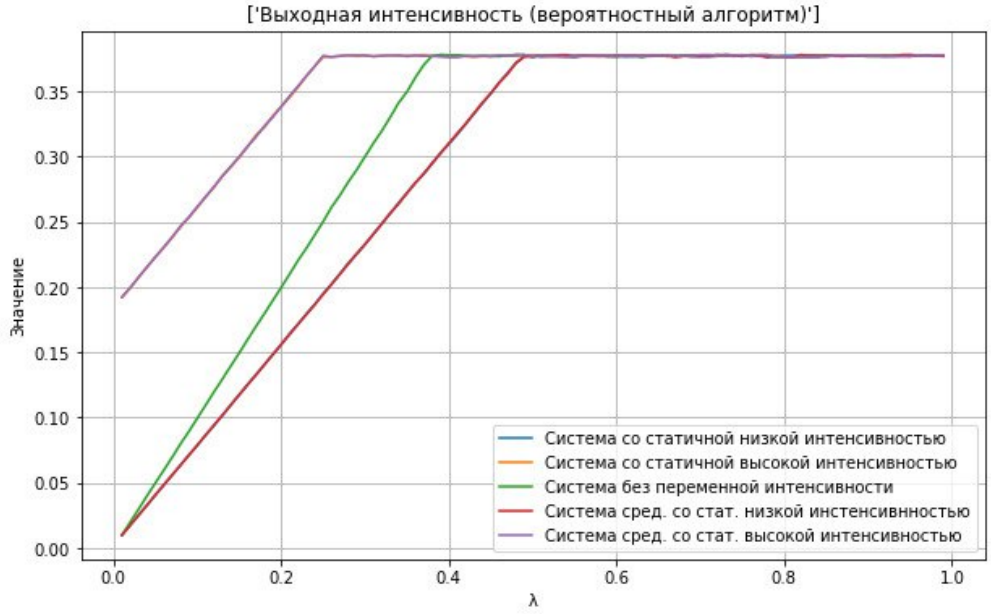

Рисунок 7 – График выходной интенсивности для вероятностного алгоритма при переменной интенсивности входного потока

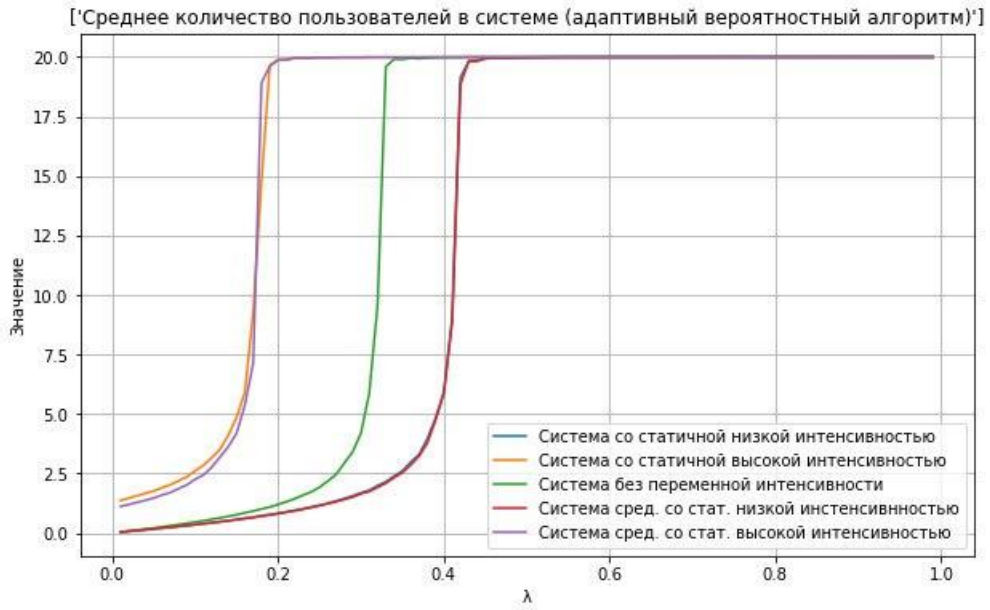

Рисунок 8 – График среднего количества пользователей для адаптивного вероятностного алгоритма при переменной интенсивности входного потока

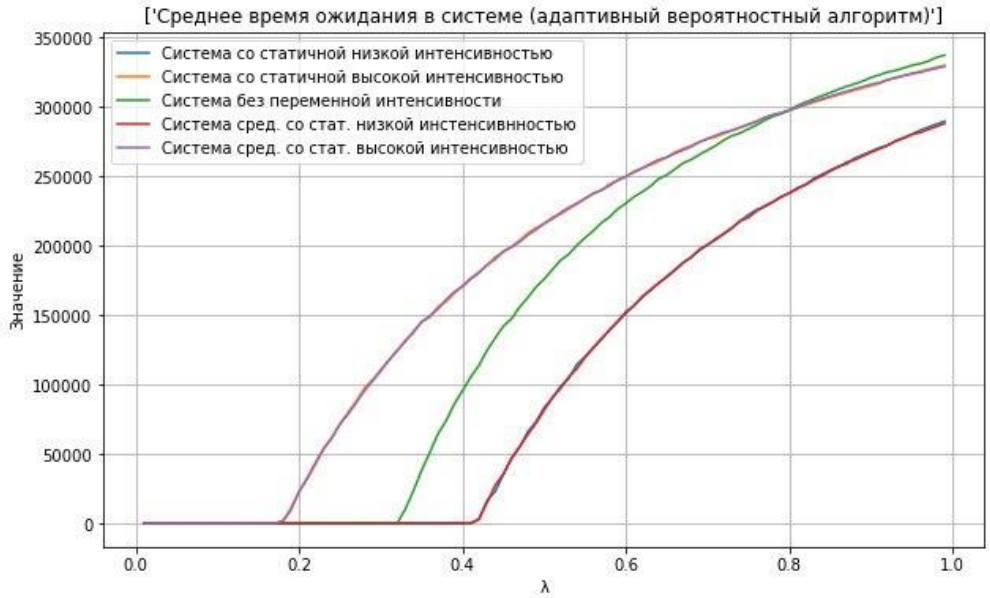

Рисунок 9 – График среднего времени ожидания (задержки) для адаптивного вероятностного алгоритма при переменной интенсивности входного потока

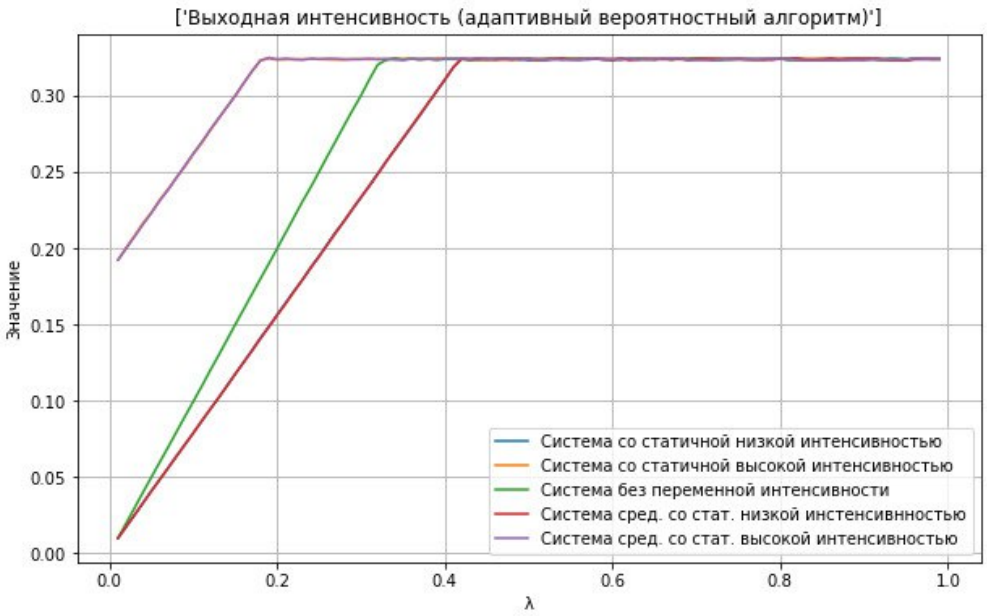

Рисунок 10 – График выходной интенсивности для адаптивного вероятностного алгоритма при переменной интенсивности входного потока

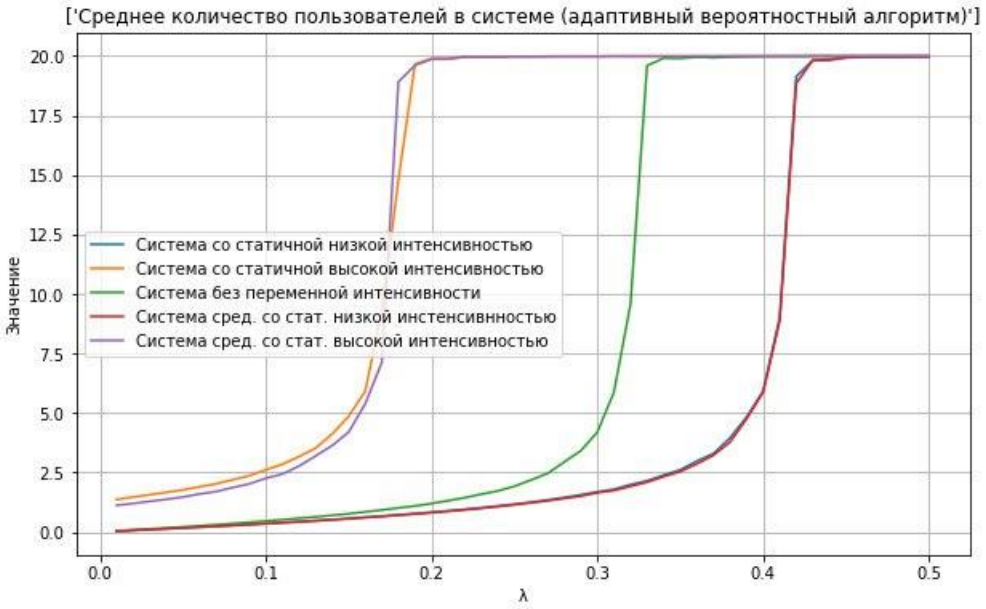

Рисунок 11 – График среднего количества пользователей для адаптивного вероятностного алгоритма при переменной интенсивности входного потока на промежутке от 0.0 до 0.5

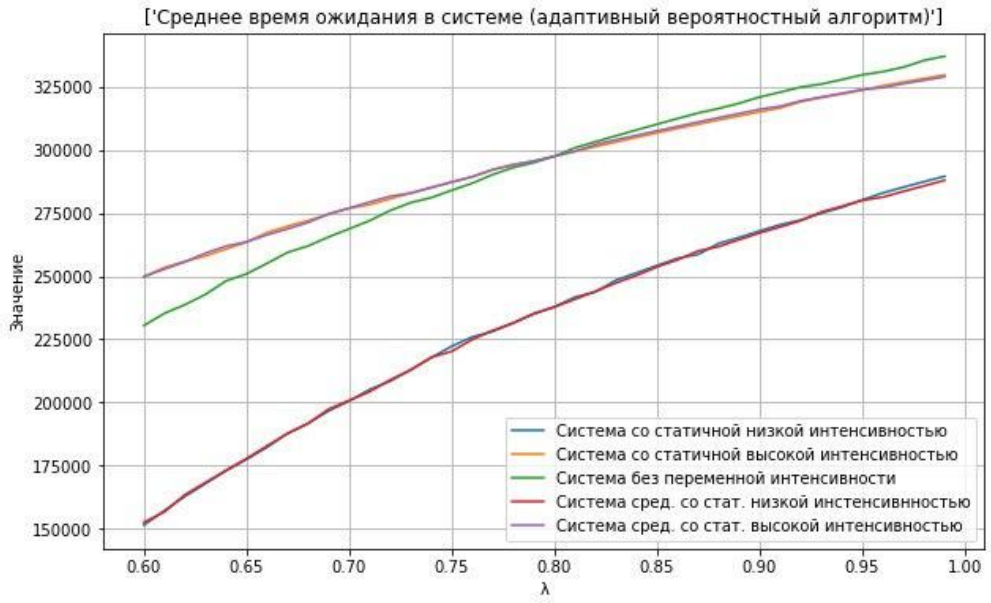

Рисунок 12 – График среднего времени ожидания (задержки) для адаптивного вероятностного алгоритма при переменной интенсивности входного потока на промежутке от 0.6 до 1.0

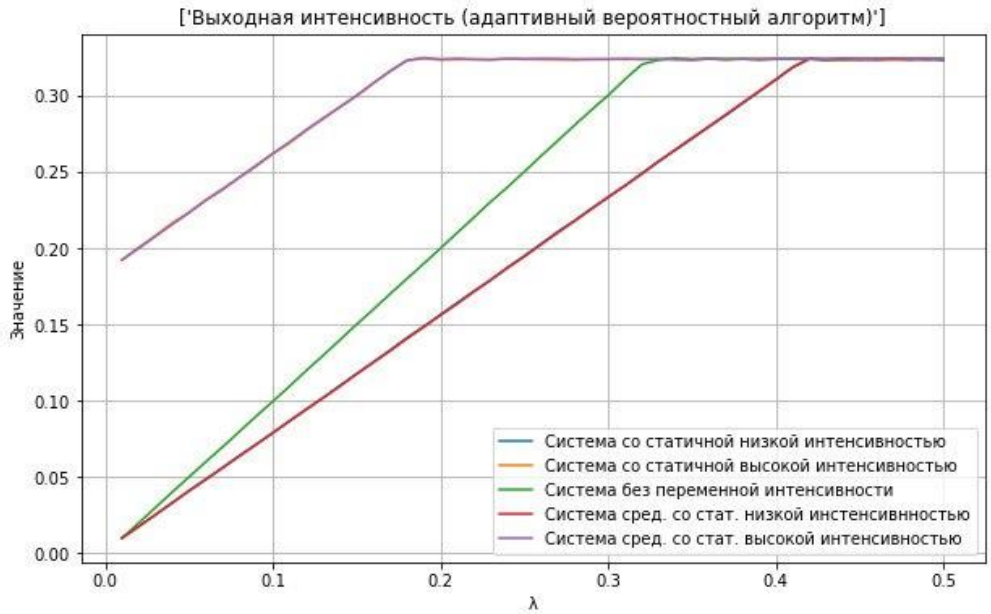

Рисунок 13 – График выходной интенсивности для адаптивного вероятностного алгоритма при переменной интенсивности входного потока на промежутке от 0.0 до 0.5

Из получившихся графиков видно, что при достижении интенсивности равной статической (неизменяемой) интенсивности, график начинает пересекаться графиком системы без переменной интенсивности. Эту зависимость можно увидеть при значении 0.01 в системе со статической низкой задержкой и при значении 0.8 а системе со статической высокой интенсивностью.

Далее были рассмотрены другие значения статических интенсивностей, на рисунках 14-16 показаны графики при статической интенсивности равной 0.3 (низкой) и 0.6 (высокой).

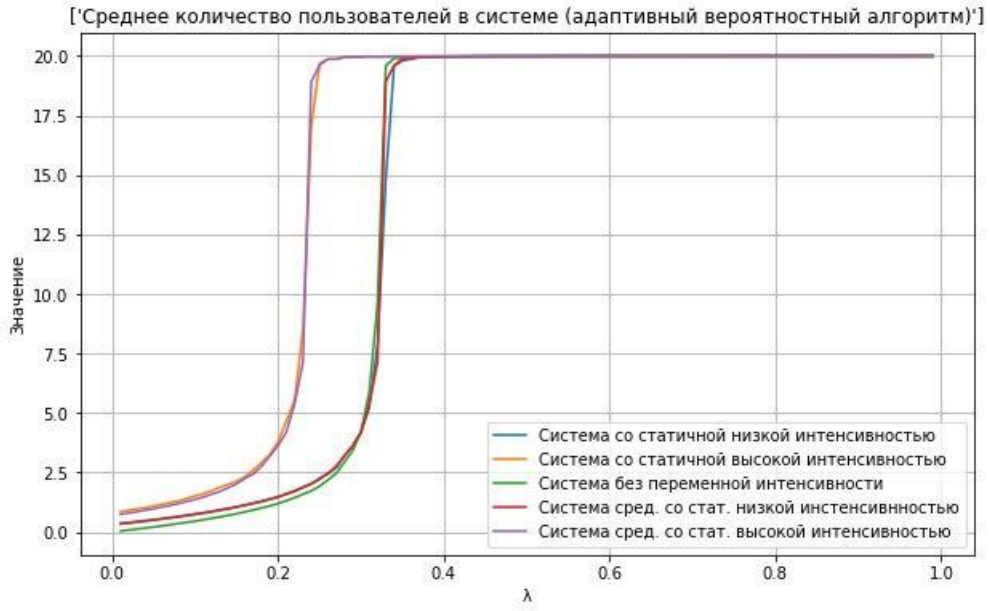

Рисунок 14 - График среднего количества пользователей для адаптивного вероятностного алгоритма при переменной интенсивности входного потока, при статических интенсивностях 0.3 и 0.6

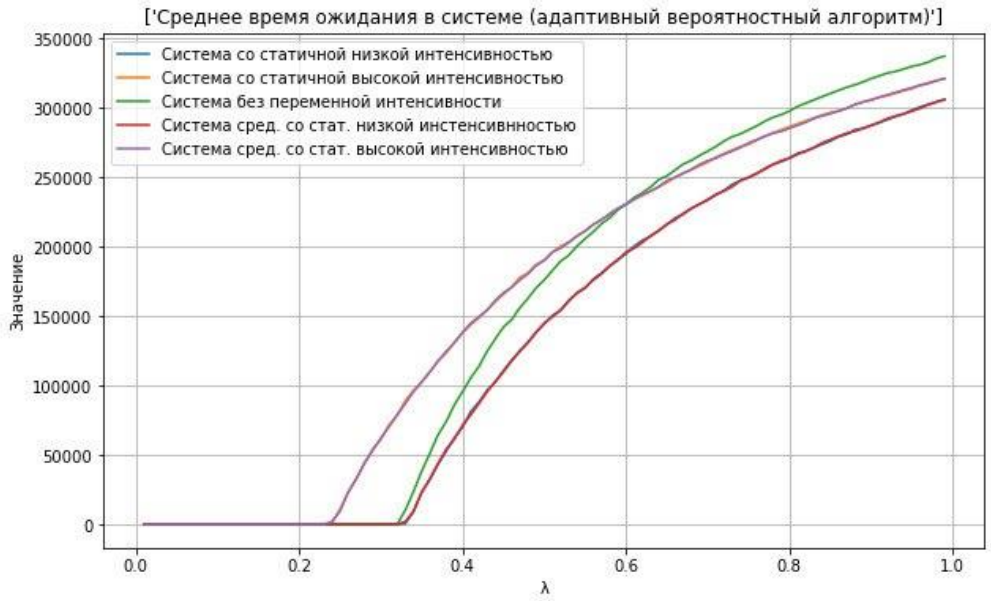

Рисунок 15 - График среднего времени ожидания (задержки) для адаптивного вероятностного алгоритма при переменной интенсивности входного потока, при статических интенсивностях 0.3 и 0.6

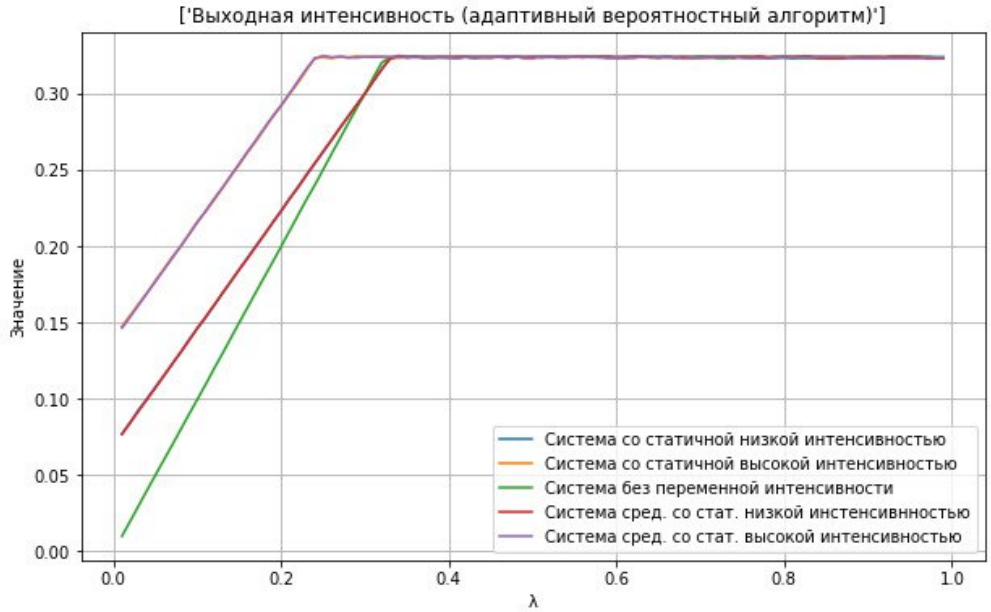

Рисунок 16 - График выходной интенсивности для адаптивного вероятностного алгоритма при переменной интенсивности входного потока, при статических интенсивностях 0.3 и 0.6

Получившиеся графики подтверждают вышесказанное – график с переменной интенсивностью пересекает график с линейно изменяющейся интенсивностью. Точка пересечения определяется статической интенсивностью для системы с переменной интенсивностью входного потока.

Далее были проведены дополнительные исследования моделируемой системы.

Для системы были проведены тесты с постоянными параметрами ($\lambda_{max}, \lambda_{min}, M, t_{\Delta}$, соотношение($\lambda_{max}$ и λ_{min})) для вероятностного алгоритма ALOHA и адаптивного вероятностного алгоритма ALOHA. На рисунке 17 приведен вывод характеристик систем с параметрами ($\lambda_{max} = 0.8,\ \lambda_{min} = 0.1, M = 30, t_{\Delta} = 0.5,$ с соотношением $1/19$).

```
ALOHA
SIMULATION STATISTICS:
AVG USERS: 2.1827
AVG WAITING TIME: 35.3978
OUTPUT INTENSITY: 0.0675705

FRAMES PASSED: 2000000
FRAMES WITH MAX INTENSITY(0.8): 100265
FRAMES WITH MIN INTENSITY(0.1): 1899735
INTENSITY RATIO: 1/19 = 0.0526316
REAL INTENSITY RATIO: 0.0527784

Adaptive ALOHA
SIMULATION STATISTICS:
AVG USERS: 0.108581
AVG WAITING TIME: 2.13048
OUTPUT INTENSITY: 0.067189

FRAMES PASSED: 2000000
FRAMES WITH MAX INTENSITY(0.8): 98860
FRAMES WITH MIN INTENSITY(0.1): 1901140
INTENSITY RATIO: 1/19 = 0.0526316
REAL INTENSITY RATIO: 0.0520004
```

Рисунок 17 – Симуляция для вероятностного и адаптивного вероятностного алгоритма ALOHA при одинаковых параметрах.

Из рисунка 17 видно, что система с не адаптивным алгоритмом получила: среднее количество абонентов 2.1827; среднее время ожидания

35.3978; интенсивность выходного потока – 0.0675705. Системой было пройдено 2 млн. кадров, при этом получилось 100265 окон с интенсивностью 0.8 и 1899735 окон с интенсивностью 0.1, что соответствует ранее выставленному соотношению 1/19. Для адаптивного алгоритма были выведены аналогичные характеристики.

На рисунке 18 – системы с параметрами ($\lambda_{max} = 0.3,\ \lambda_{min} = 0.2, M = 30, t_{\Delta} = 1.0,$ с соотношением $32/86$). Время обзора системы равно 1 млн.

```
ALOHA
SIMULATION STATISTICS:
AVG USERS: 9.27488
AVG WAITING TIME: 59.5099
OUTPUT INTENSITY: 0.227387

FRAMES PASSED: 1000000
FRAMES WITH MAX INTENSITY(0.3): 271221
FRAMES WITH MIN INTENSITY(0.2): 728779
INTENSITY RATIO: 32/86 = 0.372093
REAL INTENSITY RATIO: 0.372158

Adaptive ALOHA
SIMULATION STATISTICS:
AVG USERS: 0.938823
AVG WAITING TIME: 5.01585
OUTPUT INTENSITY: 0.227099

FRAMES PASSED: 1000000
FRAMES WITH MAX INTENSITY(0.3): 271537
FRAMES WITH MIN INTENSITY(0.2): 728463
INTENSITY RATIO: 32/86 = 0.372093
REAL INTENSITY RATIO: 0.372753
```

Рисунок 18 – Симуляция для вероятностного и адаптивного вероятностного алгоритма ALOHA при одинаковых параметрах

Для рассмотрения влияния различных значений λ_{max} и λ_{min} при различных соотношениях были построены 3D-графики для каждой системы. В графиках рассматривается влияние λ_{max}(от 0.01 до 1.0) и λ_{min}(от 0.01 до 1.0) при соотношениях 1/1, 2/8 и 8/2. 3D-графики для вероятностного алгоритма приведены на рисунках 19 – 27, для адаптивного 28 – 36.

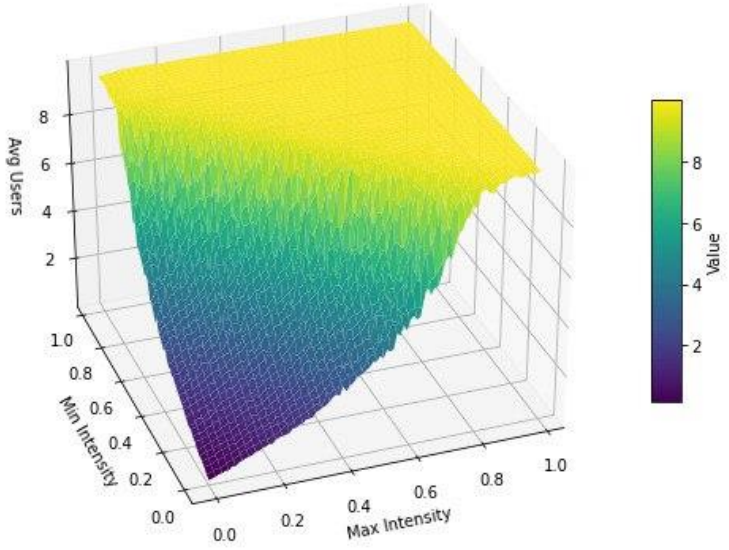

Рисунок 19 – 3D-график для среднего количество пользователей в вероятностном алгоритме с соотношением 1/1

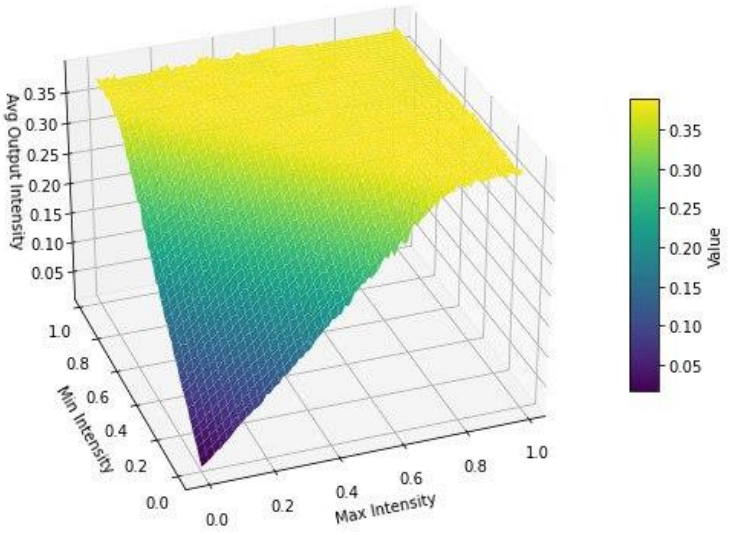

Рисунок 20 – 3D-график для выходной интенсивности в вероятностном алгоритме с соотношением 1/1

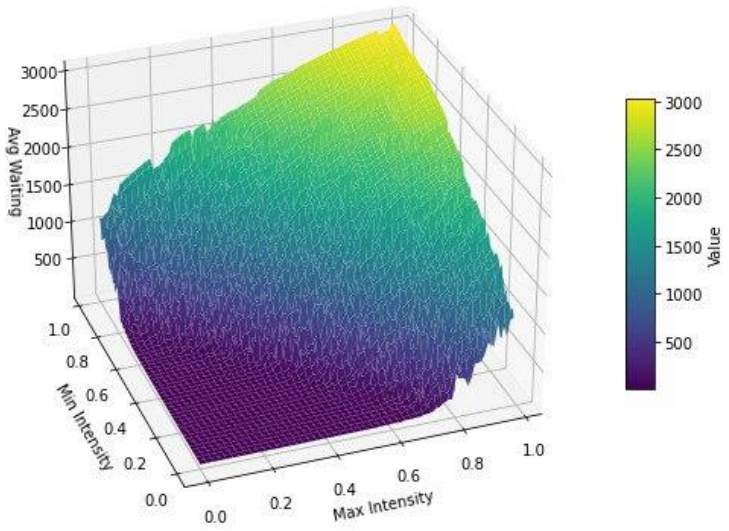

Рисунок 21 – 3D-график для среднего времени ожидания в вероятностном алгоритме с соотношением 1/1

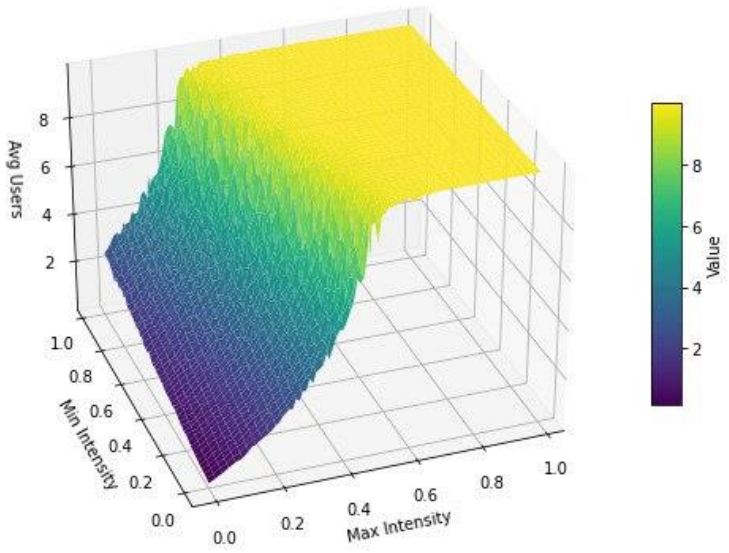

Рисунок 22 – 3D-график для среднего количество пользователей в вероятностном алгоритме с соотношением 2/8

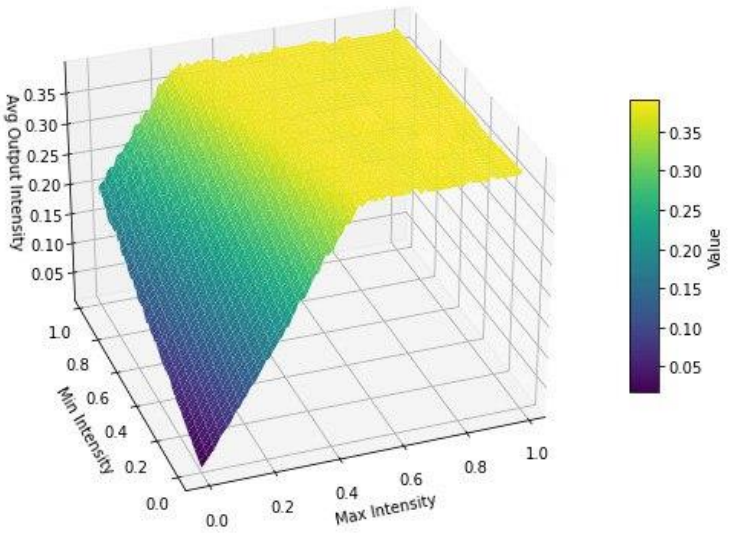

Рисунок 23 – 3D-график для выходной интенсивности в вероятностном алгоритме с соотношением 2/8

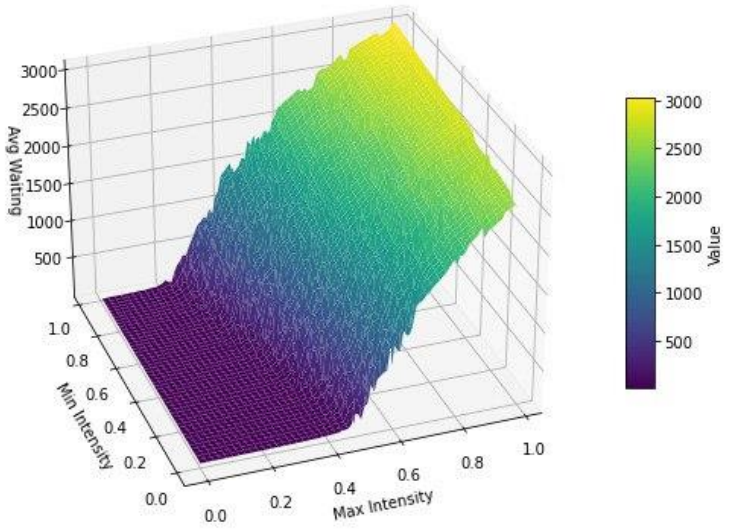

Рисунок 24 – 3D-график для среднего времени ожидания в вероятностном алгоритме с соотношением 2/8

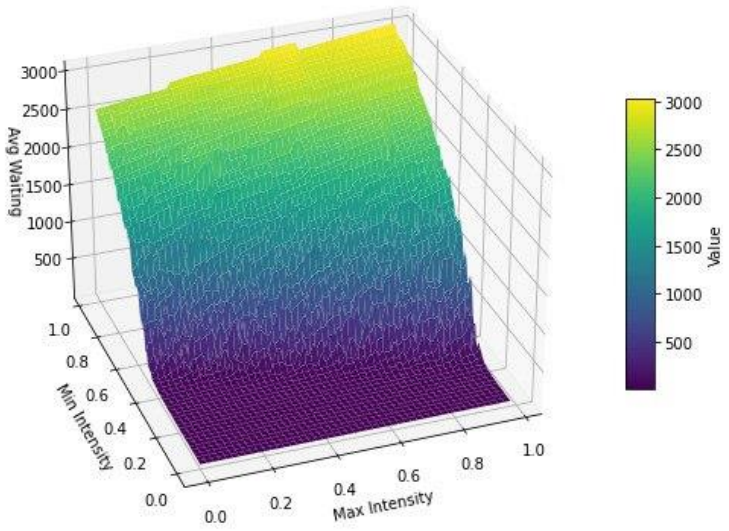

Рисунок 25 – 3D-график для среднего количество пользователей в вероятностном алгоритме с соотношением 8/2

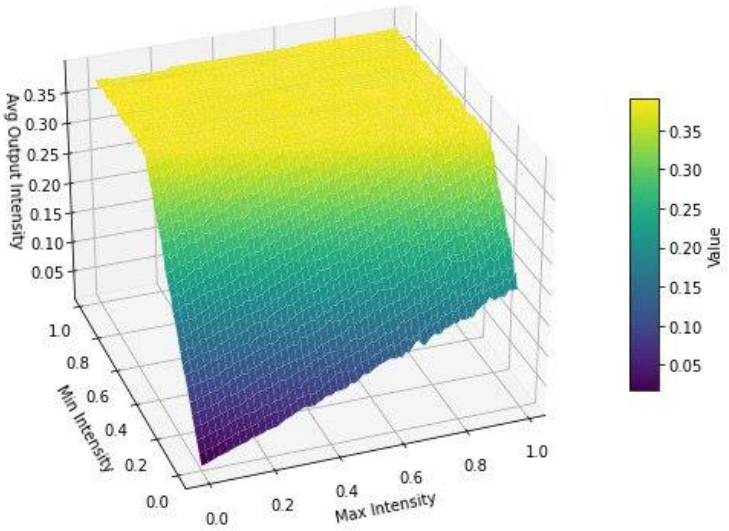

Рисунок 26 – 3D-график для выходной интенсивности в вероятностном алгоритме с соотношением 8/2

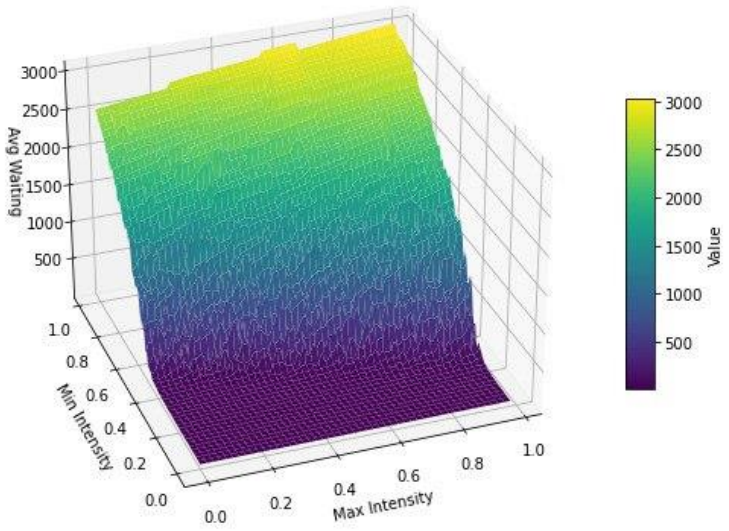

Рисунок 27 – 3D-график для среднего времени ожидания в вероятностном алгоритме с соотношением 8/2

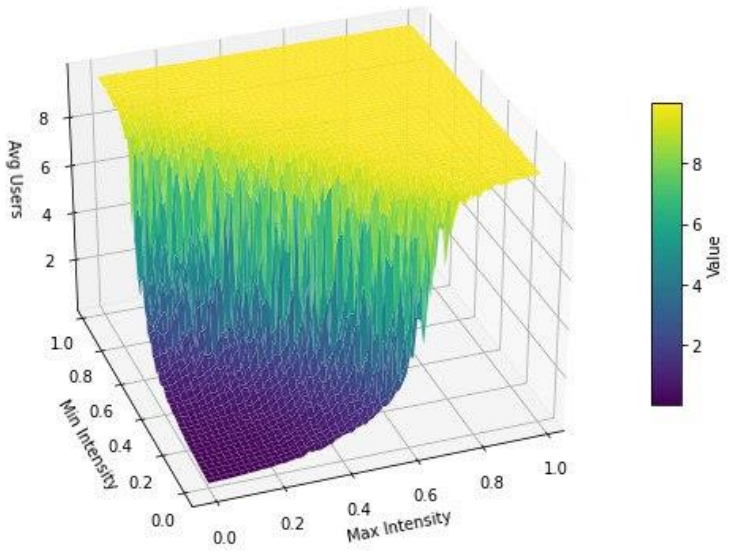

Рисунок 28 – 3D-график для среднего количество пользователей в адаптивном вероятностном алгоритме с соотношением 1/1

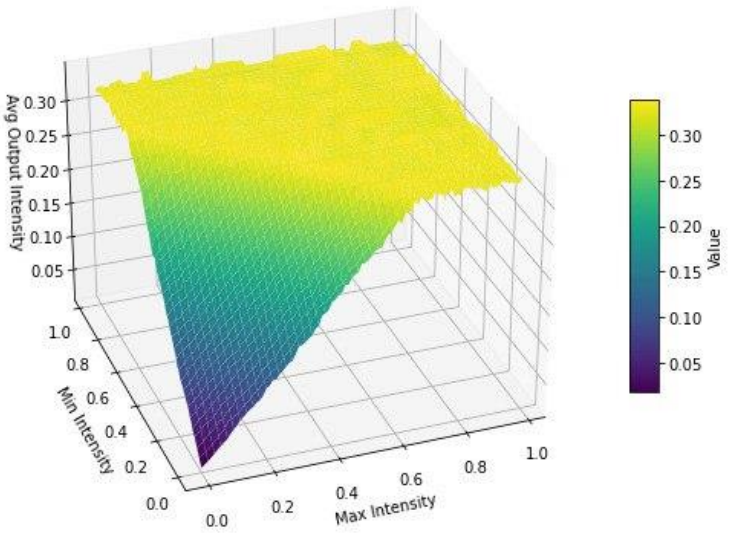

Рисунок 29 – 3D-график для выходной интенсивности в адаптивном вероятностном алгоритме с соотношением 1/1

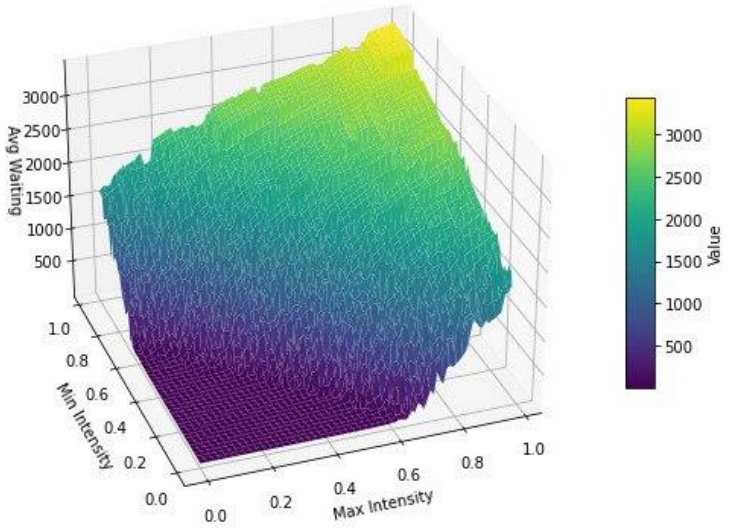

Рисунок 30 – 3D-график для среднего времени ожидания в адаптивном вероятностном алгоритме с соотношением 1/1

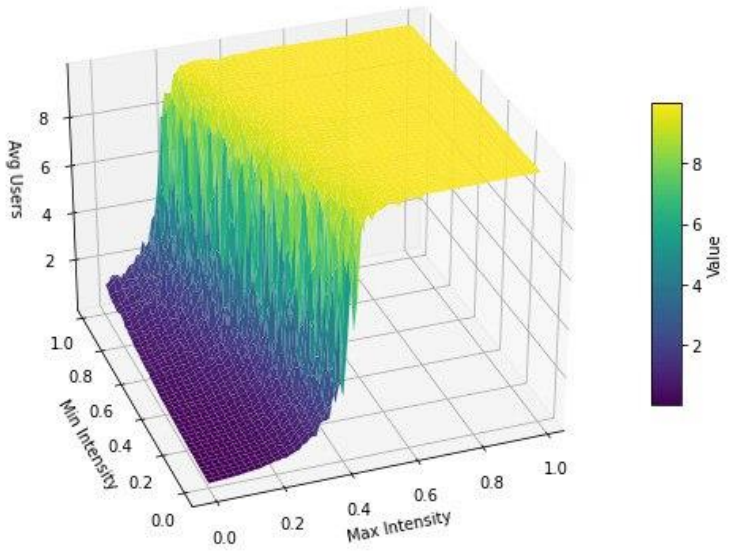

Рисунок 31 – 3D-график для среднего количество пользователей в адаптивном вероятностном алгоритме с соотношением 2/8

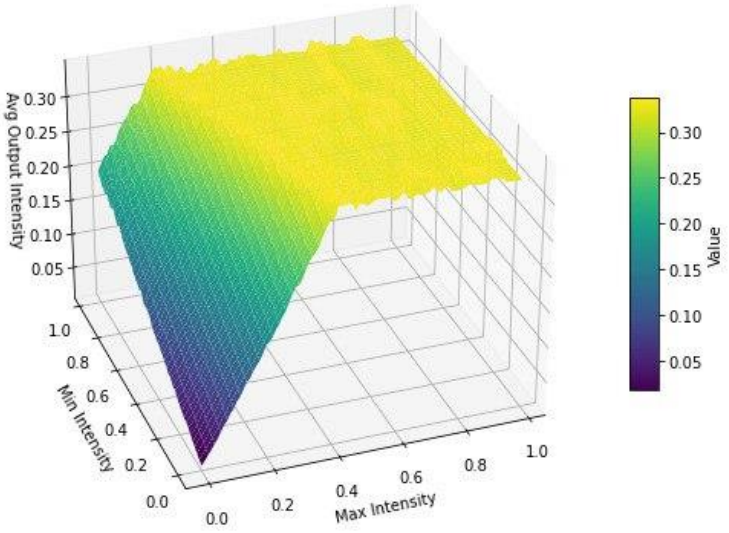

Рисунок 32 – 3D-график для выходной интенсивности в адаптивном вероятностном алгоритме с соотношением 2/8

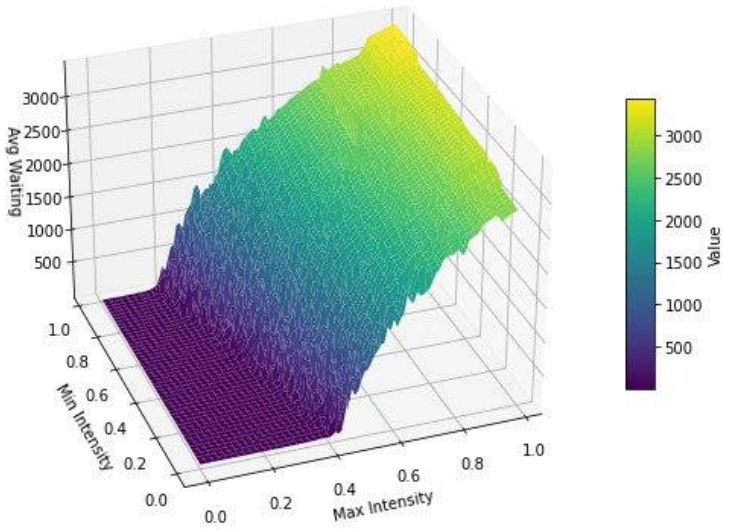

Рисунок 33 – 3D-график для среднего времени ожидания в адаптивном вероятностном алгоритме с соотношением 2/8

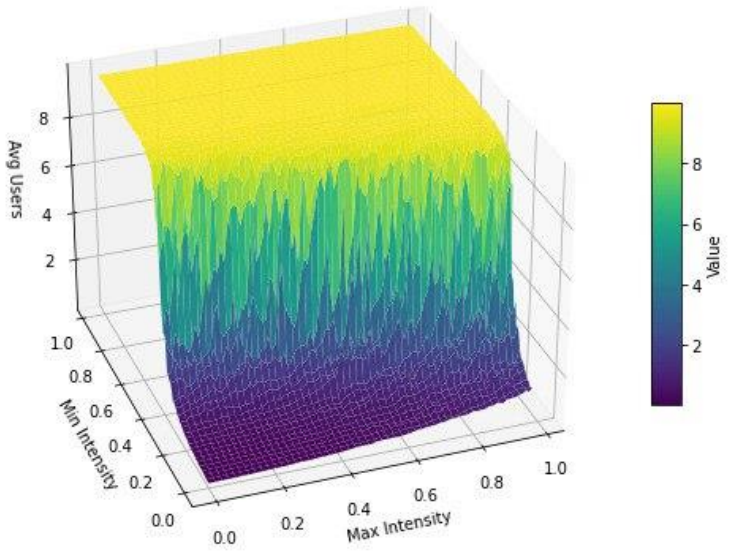

Рисунок 34 – 3D-график для среднего количество пользователей в адаптивном вероятностном алгоритме с соотношением 8/2

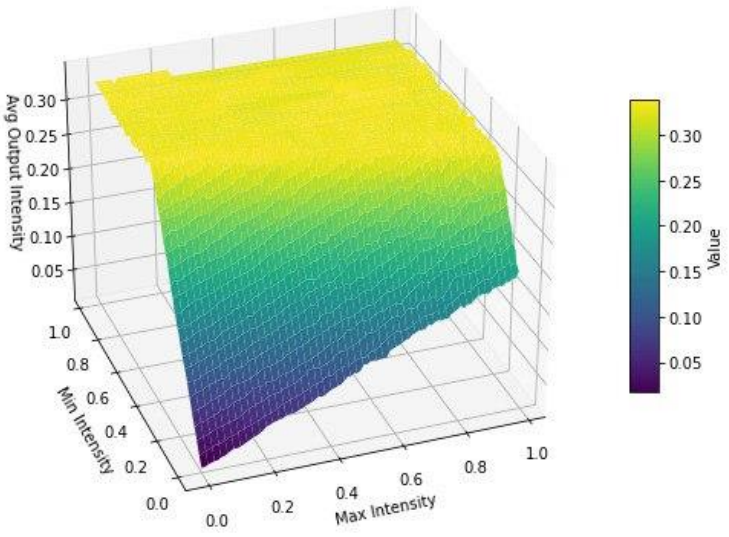

Рисунок 35 – 3D-график для выходной интенсивности в адаптивном вероятностном алгоритме с соотношением 8/2

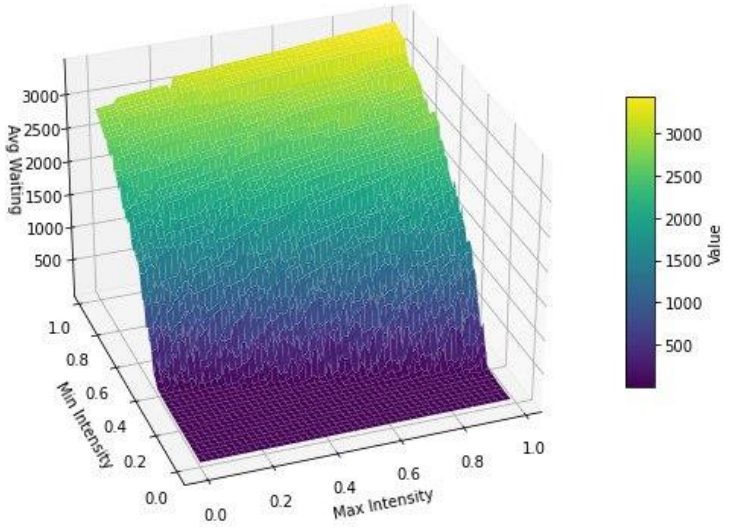

Рисунок 36 – 3D-график для среднего времени ожидания в адаптивном вероятностном алгоритме с соотношением 8/2

В 3D графиках видно, что чем больше значений с меньшей интенсивностью (по отношению к большей), тем больше (дальше) значение критической интенсивности. Из результатов можно выразить особенность: графики с обратными по значению соотношениями (2/8 и 8/2) получаются симметричными относительно друг друга.

6. Выводы по проделанной работе

В рамках курсовой работы была разработана программа, которая позволяет симулировать работу системы с алгоритмом случайного множественного доступа при переменной интенсивности потока. В качестве рассматриваемых алгоритмов были выбраны – вероятностный алгоритм ALOHA и адаптивный вероятностный алгоритм ALOHA.

Была исследована схожесть графиков системы с переменной интенсивностью входного потока и средней интенсивностью потока, рассчитанной с помощью соотношении для переменной интенсивности (рисунки 5 - 13).

Для результатов работы программы были построены 3D-графики, позволяющие рассмотреть такие зависимости как: среднее количество абонентов в системе, среднее время ожидания и выходную интенсивность от значения интенсивностей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- А. А. Бурков, А. М. Тюрликов ОСНОВЫ ПОСТРОЕНИЯ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ И СЕТЕЙ - Лабораторный практикум. - СПб: 2018
- Процесс Пуассона // Википедия URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D0%9F%D1%83%D0%B0%D1%81%D1%81%D0%BE%D0%BD%D0%B0 (дата обращения: 11.12.2024).

ПРИЛОЖЕНИЕ А

Листинг программы

main.cpp

```
#include <iostream>

#include "include/sim.h"
#include "include/file_tools.h"

const std::string directory = "../data/3d_data/default/8by2";

const std::string avg_users_file = directory + "/avg_users.txt";
const std::string avg_waiting_file = directory + "/avg_waiting.txt";
const std::string output_intensity_file = directory +
"/avg_output_intensity.txt";

const double observ_time = 1000000.0;
const int seed = 10;

const double duration = 0.01;
const uint64_t capacity = static_cast<uint64_t>(1.0 / duration) - 1;
const uint64_t capacity_3d = capacity * capacity;

void one_run_simulation_with_stats()
{
    SIM_PARAMS::default_user_probability = 0.5;
    SIM_PARAMS::number_of_users = 30;

    SIM_PARAMS::frame_time = 1.0;

    SIM_PARAMS::max_intensity = 0.3;
    SIM_PARAMS::min_intensity = 0.2;

    SIM_PARAMS::max_num_of_events = 10;

    SIM_PARAMS::max_intensity_part = 32;
    SIM_PARAMS::min_intensity_part = 86;
```

```
    sim simulation{};

    simulation.run(observ_time, seed);
    simulation.print_statistics();

    std::cout << std::endl;

    SIM_PARAMS::default_user_probability = 1.0;

    simulation.adaptive_run(observ_time, seed);
    simulation.print_statistics();
}

void data_adaptive_3d()
{
    check_for_directory(directory.c_str());
    check_for_file(avg_users_file.c_str());
    check_for_file(avg_waiting_file.c_str());
    check_for_file(output_intensity_file.c_str());

    SIM_PARAMS::default_user_probability = 1.0;
    SIM_PARAMS::number_of_users = 10;

    SIM_PARAMS::frame_time = 1.0;

    SIM_PARAMS::max_intensity = 0.1;
    SIM_PARAMS::min_intensity = 0.1;

    SIM_PARAMS::max_num_of_events = 10;

    SIM_PARAMS::max_intensity_part = 8;
    SIM_PARAMS::min_intensity_part = 2;

    std::vector<double> avg_users;
    std::vector<double> avg_waitings;
    std::vector<double> output_intensitys;
```

```
    avg_users.reserve(capacity_3d);
    avg_waitings.reserve(capacity_3d);
    output_intensitys.reserve(capacity_3d);

    sim simulation{};

    for (double max_intens = duration; max_intens < 1.0; max_intens
+= duration)
    {
        SIM_PARAMS::max_intensity = max_intens;

        for (double min_intens = duration; min_intens < 1.0;
min_intens += duration)
        {
            SIM_PARAMS::min_intensity = min_intens;

            simulation.adaptive_run(observ_time, seed);

            avg_users.push_back(simulation.get_average_users());
            avg_waitings.push_back(simulation.get_average_waiting());
            output_intensitys.push_back(simulation.get_output_intensi
ty());
        }
    }

    write_vector_file(avg_users_file, avg_users);
    write_vector_file(avg_waiting_file, avg_waitings);
    write_vector_file(output_intensity_file, output_intensitys);
}

void data_3d()
{
    check_for_directory(directory.c_str());
    check_for_file(avg_users_file.c_str());
    check_for_file(avg_waiting_file.c_str());
    check_for_file(output_intensity_file.c_str());

    SIM_PARAMS::default_user_probability = 0.5;
```

```
    SIM_PARAMS::number_of_users = 10;

    SIM_PARAMS::frame_time = 1.0;

    SIM_PARAMS::max_intensity = 0.1;
    SIM_PARAMS::min_intensity = 0.1;

    SIM_PARAMS::max_num_of_events = 10;

    SIM_PARAMS::max_intensity_part = 8;
    SIM_PARAMS::min_intensity_part = 2;

    std::vector<double> avg_users;
    std::vector<double> avg_waitings;
    std::vector<double> output_intensitys;

    avg_users.reserve(capacity_3d);
    avg_waitings.reserve(capacity_3d);
    output_intensitys.reserve(capacity_3d);

    sim simulation{};

    for (double max_intens = duration; max_intens < 1.0; max_intens
+= duration)
    {
        SIM_PARAMS::max_intensity = max_intens;

        for (double min_intens = duration; min_intens < 1.0;
min_intens += duration)
        {
            SIM_PARAMS::min_intensity = min_intens;

            simulation.run(observ_time, seed);

            avg_users.push_back(simulation.get_average_users());
            avg_waitings.push_back(simulation.get_average_waiting());
            output_intensitys.push_back(simulation.get_output_intensi
ty());
```

```
        }
    }

    write_vector_file(avg_users_file, avg_users);
    write_vector_file(avg_waiting_file, avg_waitings);
    write_vector_file(output_intensity_file, output_intensitys);
}

int main(/*int argc, char const *argv[]*/)
{

    //one_run_simulation_with_stats();

    //data_3d();
    //data_adaptive_3d();

    check_for_directory(directory.c_str());
    check_for_file(avg_users_file.c_str());
    check_for_file(avg_waiting_file.c_str());
    check_for_file(output_intensity_file.c_str());

    std::vector<double> avg_users;
    std::vector<double> avg_waitings;
    std::vector<double> output_intensitys;

    avg_users.reserve(capacity);
    avg_waitings.reserve(capacity);
    output_intensitys.reserve(capacity);

    SIM_PARAMS::default_user_probability = 1.0;
    SIM_PARAMS::number_of_users = 10;

    SIM_PARAMS::frame_time = 1.0;

    SIM_PARAMS::max_intensity = 0.1;
    SIM_PARAMS::min_intensity = 0.1;

    SIM_PARAMS::max_num_of_events = 10;
```

```
    SIM_PARAMS::max_intensity_part = 1;
    SIM_PARAMS::min_intensity_part = 1;

    sim simulation{};

    for (double intens = 0.01; intens < 1.0; intens += duration)
    {
        SIM_PARAMS::max_intensity = intens;
        SIM_PARAMS::min_intensity = intens;

        simulation.run(observ_time, seed);
        //simulation.adaptive_run(observ_time, seed);

        avg_users.push_back(simulation.get_average_users());
        avg_waitings.push_back(simulation.get_average_waiting());
        output_intensitys.push_back(simulation.get_output_intensity()
);
    }

    write_vector_file(avg_users_file, avg_users);
    write_vector_file(avg_waiting_file, avg_waitings);
    write_vector_file(output_intensity_file, output_intensitys);

    return 0;
}
```

sim.h

```
#ifndef _SIMULATION_
#define _SIMULATION_

#include "simulation_parameters.h"

class sim
{
private:
    double average_users;
    double average_waiting;
    double output_intensity;

    uint64_t max_intensity_frames;
    uint64_t min_intensity_frames;
    uint64_t frame_counter;

public:
    sim(/*uint64_t number_of_users, double max_intensity, double
min_intensity, uint64_t max_num_of_events, double
default_user_probability, double frame_time*/);
    ~sim();

    void run(double observ_timem, int seed);
    void adaptive_run(double observ_timem, int seed);
    void print_statistics();

    double get_average_users();
    double get_average_waiting();
    double get_output_intensity();
};

#endif // !_SIMULATION_
```

sim.cpp

```
#include "sim.h"

#include <vector>
#include <iostream>

#include "user.h"

sim::sim(/*uint64_t number_of_users, double max_intensity, double
min_intensity, uint64_t max_num_of_events, double
default_user_probability, double frame_time*/) :
    average_users   (0.0),
    average_waiting (0.0),
    output_intensity(0.0),
    max_intensity_frames(0),
    min_intensity_frames(0),
    frame_counter(0)
{
    // SIM_PARAMS::default_user_probability =
default_user_probability;

    // SIM_PARAMS::max_num_of_events = max_num_of_events;
    // SIM_PARAMS::number_of_users = number_of_users;

    // SIM_PARAMS::max_intensity = max_intensity;
    // SIM_PARAMS::min_intensity = min_intensity;

    // SIM_PARAMS::frame_time = frame_time;

    // SIM_PARAMS::init();
}

sim::~sim()
{
}

void sim::run(double observ_time, int seed)
```

```
{
    //
================================================================
===============
    // Проверка корректности параметров симуляции
    if (SIM_PARAMS::default_user_probability == 0.0)
    {
        std::cerr << "ERROR: sim() default_user_probability is 0.0"
<< std::endl;
        return;
    }

    if (SIM_PARAMS::frame_time == 0.0)
    {
        std::cerr << "ERROR: frame_time() frame_time is 0.0" <<
std::endl;
        return;
    }

    if (SIM_PARAMS::number_of_users == 0)
    {
        std::cerr << "ERROR: number_of_users() number_of_users is
0.0" << std::endl;
        return;
    }

    if (observ_time < SIM_PARAMS::frame_time * 2)
    {
        std::cerr << "ERROR: frame_time() frame_time is less than 2
frames" << std::endl;
        return;
    }

    if (SIM_PARAMS::max_intensity_part == 0 ||
SIM_PARAMS::min_intensity_part == 0)
    {
        std::cerr << "ERROR: on of intensity_part() is 0" <<
std::endl;
```

```
        return;
    }

    SIM_PARAMS::init();
    srand(seed);

    //
================================================================
===============
    // Создание массива абонентов
    std::vector<user> users;
    users.reserve(SIM_PARAMS::number_of_users);

    for (uint64_t iter(0); iter < SIM_PARAMS::number_of_users;
++iter)
    {
        users.push_back(user(SIM_PARAMS::default_user_probability));
    }

    //
================================================================
===============
    // Вспомогательные переменные

    SIM_PARAMS::discrete_probabilitys probs{};

    this->average_users = 0.0;
    this->average_waiting = 0.0;
    this->output_intensity = 0.0;

    this->max_intensity_frames = 0;
    this->min_intensity_frames = 0;
    this->frame_counter = 0;

    CHANNEL_STATUS ss = CHANNEL_STATUS::EMPTY;
    CURRENT_INTENSITY curr_intens = CURRENT_INTENSITY::MAX;

    user *user_in_channel = nullptr;
```

```
    double current_frame_time(0.0);

    uint64_t served_events(0);

    //
==========================================================================
===============
    // Моделирование

    while (current_frame_time < observ_time)
    {
        // Просмотр текущего окна на наличие позлователей для
передачи
        uint64_t users_in_frame(0);

        user_in_channel = nullptr;
        ss = CHANNEL_STATUS::EMPTY;

        for (user &usr : users)
        {
            if (!usr.queue.empty())
            {
                if (usr.try_to_send())
                {
                    if (user_in_channel == nullptr)
                    {
                        user_in_channel = &usr;
                    }
                    else
                    {
                        ss = CHANNEL_STATUS::CONFLICT;
                    }
                }

                users_in_frame++;
            }
        }
```

```
        average_users += users_in_frame;

        // Если в канале 1 передающий абонент
        if (ss == CHANNEL_STATUS::EMPTY && user_in_channel !=
nullptr)
        {
            ss = CHANNEL_STATUS::SUCCESS;
        }

        // Обслуживание абонента
        if (ss == CHANNEL_STATUS::SUCCESS)
        {
            event ev(user_in_channel->queue.front());
            user_in_channel->queue.pop();

            average_waiting += (current_frame_time +
SIM_PARAMS::frame_time) - ev.arrival_time;
            served_events += 1;
        }

        // Генерация заявок

        // Выбор новой интенсивности
        double intensity_lot = static_cast<double>(rand()) /
RAND_MAX;
        if (intensity_lot <= SIM_PARAMS::intensity_ratio)
        {
            curr_intens = CURRENT_INTENSITY::MAX;
            max_intensity_frames++;
        }
        else
        {
            curr_intens = CURRENT_INTENSITY::MIN;
            min_intensity_frames++;
        }
```

```
        double arrival_time = current_frame_time +
SIM_PARAMS::frame_time / 2;

        for (user &usr : users)
        {
            for(uint64_t events = probs.get_events(curr_intens);
events > 0; events--)
            {
                usr.queue.push(event(arrival_time));
            }
        }

        current_frame_time += SIM_PARAMS::frame_time;
        frame_counter++;
    }

    average_users /= frame_counter;
    if (served_events == 0)
    {
        // Установка максимального значения (время обзора)
        average_waiting = observ_time;
        // Установка минимльного значения (из системы ничего не
вышло)
        output_intensity = 0;
    }
    else
    {
        // При домножении на frame_time превращает в коэффициент от
frane_time
        average_waiting /= static_cast<double>(served_events) *
SIM_PARAMS::frame_time;
        output_intensity = static_cast<double>(served_events) /
frame_counter;
    }
}

void sim::adaptive_run(double observ_time, int seed)
{
```

```
    //
================================================================================
===============
    // Проверка корректности параметров симуляции
    if (SIM_PARAMS::default_user_probability == 0.0)
    {
        std::cerr << "ERROR: sim() default_user_probability is 0.0"
<< std::endl;
        return;
    }

    if (SIM_PARAMS::frame_time == 0.0)
    {
        std::cerr << "ERROR: frame_time() frame_time is 0.0" <<
std::endl;
        return;
    }

    if (SIM_PARAMS::number_of_users == 0)
    {
        std::cerr << "ERROR: number_of_users() number_of_users is
0.0" << std::endl;
        return;
    }

    if (observ_time < SIM_PARAMS::frame_time * 2)
    {
        std::cerr << "ERROR: frame_time() frame_time is less than 2
frames" << std::endl;
        return;
    }

    if (SIM_PARAMS::max_intensity_part == 0 ||
SIM_PARAMS::min_intensity_part == 0)
    {
        std::cerr << "ERROR: on of intensity_part() is 0" <<
std::endl;
        return;
```

```
    }

    SIM_PARAMS::init();
    srand(seed);

    //
==========================================================================
===============
    // Создание массива абонентов
    std::vector<user> users;
    users.reserve(SIM_PARAMS::number_of_users);

    for (uint64_t iter(0); iter < SIM_PARAMS::number_of_users;
++iter)
    {
        users.push_back(user(SIM_PARAMS::default_user_probability));
    }

    //
==========================================================================
===============
    // Вспомогательные переменные

    SIM_PARAMS::discrete_probabilitys probs{};

    this->average_users = 0.0;
    this->average_waiting = 0.0;
    this->output_intensity = 0.0;

    this->max_intensity_frames = 0;
    this->min_intensity_frames = 0;
    this->frame_counter = 0;

    CHANNEL_STATUS ss = CHANNEL_STATUS::EMPTY;
    CURRENT_INTENSITY curr_intens = CURRENT_INTENSITY::MAX;

    user *user_in_channel = nullptr;
```

```
    double current_frame_time(0.0);

    uint64_t served_events(0);

    //
================================================================================
===============
    // Моделирование

    while (current_frame_time < observ_time)
    {
        // Просмотр текущего окна на наличие позлователей для
передачи
        uint64_t users_in_frame(0);

        user_in_channel = nullptr;
        ss = CHANNEL_STATUS::EMPTY;

        for (user &usr : users)
        {
            if (!usr.queue.empty())
            {
                if (usr.try_to_send())
                {
                    if (user_in_channel == nullptr)
                    {
                        user_in_channel = &usr;
                    }
                    else
                    {
                        ss = CHANNEL_STATUS::CONFLICT;
                    }
                }

                users_in_frame++;
            }
        }
```

```
        average_users += users_in_frame;

        // Если в канале 1 передающий абонент
        if (ss == CHANNEL_STATUS::EMPTY && user_in_channel !=
nullptr)
        {
            ss = CHANNEL_STATUS::SUCCESS;
        }

        // Обслуживание абонента
        if (ss == CHANNEL_STATUS::SUCCESS)
        {
            event ev(user_in_channel->queue.front());
            user_in_channel->queue.pop();

            average_waiting += (current_frame_time +
SIM_PARAMS::frame_time) - ev.arrival_time;
            served_events += 1;
        }

        // Генерация заявок

        // Выбор новой интенсивности
        double intensity_lot = static_cast<double>(rand()) /
RAND_MAX;
        if (intensity_lot <= SIM_PARAMS::intensity_ratio)
        {
            curr_intens = CURRENT_INTENSITY::MAX;
            max_intensity_frames++;
        }
        else
        {
            curr_intens = CURRENT_INTENSITY::MIN;
            min_intensity_frames++;
        }

        double arrival_time = current_frame_time +
SIM_PARAMS::frame_time / 2;
```

```
        // Адаптация вероятности и добавление новых заявок
        for (user &usr : users)
        {
            if (!usr.queue.empty())
            {
                usr.adapt_probability(ss, users_in_frame);
            }
            else
            {
                usr.set_probability(SIM_PARAMS::default_user_probabil
ity);
            }

            for(uint64_t events = probs.get_events(curr_intens);
events > 0; events--)
            {
                usr.queue.push(event(arrival_time));
            }
        }

        current_frame_time += SIM_PARAMS::frame_time;
        frame_counter++;
    }

    average_users /= frame_counter;
    if (served_events == 0)
    {
        // Установка максимального значения (время обзора)
        average_waiting = observ_time;
        // Установка минимльного значения (из системы ничего не
вышло)
        output_intensity = 0;
    }
    else
    {
        // При домножении на frame_time превращает в коэффициент от
frane_time
```

```
        average_waiting /= static_cast<double>(served_events) *
SIM_PARAMS::frame_time;
        output_intensity = static_cast<double>(served_events) /
frame_counter;
    }
}

void sim::print_statistics()
{
    std::cout << "SIMULATION STATISTICS:" << std::endl;
    std::cout << "AVG USERS: " << average_users << std::endl;
    std::cout << "AVG WAITING TIME: " << average_waiting <<
std::endl;
    std::cout << "OUTPUT INTENSITY: " << output_intensity <<
std::endl << std::endl;

    std::cout << "FRAMES PASSED: " << frame_counter << std::endl;
    std::cout << "FRAMES WITH MAX INTENSITY(" <<
SIM_PARAMS::max_intensity << "): " << max_intensity_frames <<
std::endl;
    std::cout << "FRAMES WITH MIN INTENSITY(" <<
SIM_PARAMS::min_intensity << "): " << min_intensity_frames <<
std::endl;

    std::cout << "INTENSITY RATIO: " <<
SIM_PARAMS::max_intensity_part << "/" <<
SIM_PARAMS::min_intensity_part << " = "
                                    << (double)
SIM_PARAMS::max_intensity_part / (double)
SIM_PARAMS::min_intensity_part << std::endl;
    std::cout << "REAL INTENSITY RATIO: " << (double)
max_intensity_frames / (double) min_intensity_frames << std::endl;
}

double sim::get_average_users()
{
    return this->average_users;
}
```

```
double sim::get_average_waiting()
{
    return this->average_waiting;
}

double sim::get_output_intensity()
{
    return this->output_intensity;
}
```

user.h

```
#ifndef _USER_
#define _USER_

#include <queue>

#include "event.h"
#include "simulation_parameters.h"

class user
{
private:
    double probability;
public:
    std::queue<event> queue;

    user(double probability);
    ~user();

    bool try_to_send();
    void adapt_probability(CHANNEL_STATUS &status, std::size_t
&users_in_channel);
    void set_probability(double probability);
};

#endif // !_USER_
```

user.cpp

```
#include "user.h"

#include <random>

user::user(double probability) : probability(probability)
{
}

user::~user()
{
}

bool user::try_to_send()
{
    if (static_cast<double>(rand())/RAND_MAX <= probability)
    {
        return true;
    }
    return false;
}

void user::adapt_probability(CHANNEL_STATUS &status, std::size_t
&users_in_channel)
{
    switch (status)
    {
    case CHANNEL_STATUS::CONFLICT:
        this->probability = std::max((1.0 /
static_cast<double>(users_in_channel)), probability / 2.0);
        break;
    case CHANNEL_STATUS::SUCCESS:
        break;
    case CHANNEL_STATUS::EMPTY:
        this->probability = std::min(1.0, 2.0 * probability);
        break;
    }
```

```
}


void user::set_probability(double probability)
{
    this->probability = probability;
}
```

simulation_parameters.h

```
#ifndef _SYSTEM_PARAMS_
#define _SYSTEM_PARAMS_

#include <stdint.h>
#include <random>

enum class CHANNEL_STATUS
{
    CONFLICT,
    SUCCESS,
    EMPTY
};

enum class CURRENT_INTENSITY
{
    MAX,
    MIN
};

namespace SIM_PARAMS
{
    // Стандартная вероятность передачи
    extern double default_user_probability;

    // Максимальное число заявок от одного пользователя за кадр
    extern uint64_t max_num_of_events;
    // Время одного кадра
    extern double frame_time;

    // Максимальная интенсивность
    extern double max_intensity;
    // Минимальная интенсивность
    extern double min_intensity;

    // Число пользователей в моделировании
    extern uint64_t number_of_users;
```

```
    // Максимальная интенсивность потока пользователя
    extern double max_user_intensity;
    // Минимальная интенсиваность потока пользователя
    extern double min_user_intensity;

    // Количество кадров с интенсивностью MAX (для рассчета
соотношения)
    extern uint64_t max_intensity_part;
    // Количество кадров с интенсивностью MIN (для рассчета
соотношения)
    extern uint64_t min_intensity_part;
    /*
        Соотношение max_intensity / min_intensity
        **Вероятность выбора max_intensity
    */
    extern double intensity_ratio;

    void init();

    class discrete_probabilitys
    {
    private:
        uint64_t len;
        uint64_t *num_of_events;
        double *probabilitys_max;
        double *probabilitys_min;

        inline double count_probaility(uint64_t &num_of_events,
double &intensity);
        inline uint64_t get_events_by_lot(const double &lot, double
*array);

    public:
        discrete_probabilitys();
        ~discrete_probabilitys();

        uint64_t get_events(CURRENT_INTENSITY &curr_intens);
```

```
    };

} // namespace SIM_PARAMS

#endif // !_SYSTEM_PARAMS_
```

simulation_parameters.cpp

```
#include "simulation_parameters.h"

#include <math.h>
#include <iostream>

double SIM_PARAMS::default_user_probability(0.5);

uint64_t SIM_PARAMS::max_num_of_events(10);
double SIM_PARAMS::frame_time(1.0);

double SIM_PARAMS::max_intensity(0.5);
double SIM_PARAMS::min_intensity(0.5);

uint64_t SIM_PARAMS::number_of_users(10);

double SIM_PARAMS::max_user_intensity(max_intensity /
number_of_users);
double SIM_PARAMS::min_user_intensity(min_intensity /
number_of_users);

uint64_t SIM_PARAMS::max_intensity_part(1);
uint64_t SIM_PARAMS::min_intensity_part(1);

double SIM_PARAMS::intensity_ratio(0.5);

inline double
SIM_PARAMS::discrete_probabilitys::count_probaility(uint64_t
&num_of_events, double &intensity)
{
    double dividend(1.0);
    double divisor(tgamma(num_of_events + 1));
    if (num_of_events > 0)
    {
        dividend = pow((intensity * frame_time), num_of_events);
    }
```

```
    dividend *= exp(-intensity * frame_time);
    return dividend / divisor;
}

inline uint64_t
SIM_PARAMS::discrete_probabilitys::get_events_by_lot(const double
&lot, double *array)
{
    uint64_t iter(0);
    while (array[iter] < lot)
    {
        iter++;
    }

    return iter;
}

SIM_PARAMS::discrete_probabilitys::discrete_probabilitys()
{
    if (max_num_of_events == 0)
    {
        std::cerr << "ERROR: discrete_probabilitys constructor:
max_num_of_event is 0" << std::endl;
        return;
    }

    this->len = max_num_of_events + 1;

    this->num_of_events = new uint64_t[len];
    this->probabilitys_max = new double[len];
    this->probabilitys_min = new double[len];

    for (uint64_t iter(0); iter < len - 1; ++iter)
    {
        num_of_events[iter] = iter;
        probabilitys_max[iter] = (iter == 0) ? count_probaility(iter,
max_user_intensity) : probabilitys_max[iter - 1] +
count_probaility(iter, max_user_intensity);
```

```
        probabilitys_min[iter] = (iter == 0) ? count_probaility(iter,
min_user_intensity) : probabilitys_min[iter - 1] +
count_probaility(iter, min_user_intensity);

        //std::cout << "Prob(" << iter << "): MAX = " <<
probabilitys_max[iter] << "; MIN = " << probabilitys_min[iter] <<
std::endl;
    }

    num_of_events[max_num_of_events] = max_num_of_events;
    probabilitys_max[max_num_of_events] = 1.0;
    probabilitys_min[max_num_of_events] = 1.0;
    //std::cout << "Prob(" << max_num_of_events << "): MAX = " <<
probabilitys_max[max_num_of_events] << "; MIN = " <<
probabilitys_min[max_num_of_events] << std::endl << std::endl;
}

SIM_PARAMS::discrete_probabilitys::~discrete_probabilitys()
{
    delete [] num_of_events;
    delete [] probabilitys_max;
    delete [] probabilitys_min;
}

uint64_t
SIM_PARAMS::discrete_probabilitys::get_events(CURRENT_INTENSITY
&curr_intens)
{
    double lot = static_cast<double>(rand()) / RAND_MAX;
    uint64_t index = 0;

    switch (curr_intens)
    {
    case CURRENT_INTENSITY::MAX:
        index = get_events_by_lot(lot, probabilitys_max);
        break;

    case CURRENT_INTENSITY::MIN:
```

```
        index = get_events_by_lot(lot, probabilitys_min);
        break;
    }

    return num_of_events[index];
}

void SIM_PARAMS::init()
{
    max_user_intensity = max_intensity / number_of_users;
    min_user_intensity = min_intensity / number_of_users;

    intensity_ratio = (1.0 / (static_cast<double>(max_intensity_part
+ min_intensity_part))) * max_intensity_part;
}
```

event.h

```
#ifndef _EVENT_
#define _EVENT_

class event
{
public:
    double arrival_time;
    event();
    event(const double arrival_time);
};

#endif // !_EVENT_
```

event.cpp

```
#include "event.h"

event::event() : arrival_time(0)
{
}

event::event(const double arrival_time) : arrival_time(arrival_time)
{
}
```

file_tools.h

```
#ifndef _FILE_TOOLS_
#define _FILE_TOOLS_

#include <string>
#include <vector>
#include <fstream>
#include <iostream>

void check_for_directory(const char *directory);

void check_for_file(const char *path);

template <typename T>
void write_vector_file(std::string filename, std::vector<T>& data);

#endif // !_FILE_TOOLS_

template <typename T>
inline void write_vector_file(std::string filename, std::vector<T>
&data)
{
    std::ofstream out;
    out.open(filename, std::ios::out);

    if(!out.is_open()) {
        std::cout << "Error open file: " << filename << std::endl;
        return;
    }

    out << data.size() << std::endl << std::endl;

    for (auto& var : data) {
        out << var << std::endl;
    }

    out.close();
```

```
    std::cout << "File has been written: " << filename << std::endl;
    return;
}
```

file_tools.cpp

```
#include "file_tools.h"

#include <filesystem>

namespace fs = std::filesystem;

void check_for_directory(const char *directory)
{
    if(!fs::exists(directory))
    {
        if(fs::create_directory(directory))
        {
            std::cout << "Directory " << directory << " has been
created" << std::endl;
        }
        else
        {
            std::cerr << "Failed to create directory " << directory
<< std::endl;
        }
    }
}

void check_for_file(const char *path)
{
    if(!fs::exists(path))
    {
        std::ofstream file(path);
        if(file)
        {
            std::cout << "File has been created in " << path <<
std::endl;
        }
        else
        {
```

```
            std::cerr << "Failed to create file in " << path <<
std::endl;
        }
    }
}
```

plotmaker.py

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

# Функция для чтения данных из файла
def read_file_to_array(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()
        return [float(line.strip()) for line in lines[2:] if
line.strip()]

# Директория
directory_path = "../data/ten_users/smo"

# Пути к файлам
avg_users_file = directory_path + "/avg_users.txt"
avg_waiting_file = directory_path + "/avg_waiting.txt"
output_intens_file = directory_path + "/avg_output_intensity.txt"

# Чтение данных
data_avg_users = read_file_to_array(avg_users_file)
data_avg_waiting = read_file_to_array(avg_waiting_file)
data_output_intens = read_file_to_array(output_intens_file)

# Генерация значений lambdas
lambdas = np.arange(0.01, 1.0, 0.01)

'''
# Вычисление теоретических значений
data_t_events = [(l * (2 - l)) / (2 * (1 - l)) for l in lambdas]
data_ta_wait = [t / l for t, l in zip(data_t_events, lambdas)]
data_ts_wait = [(t / l) + 0.5 for t, l in zip(data_t_events,
lambdas)]
'''

x_limit = None
```

```
# Функция для построения графика
def plot_graph(x, ys, labels, title, xlabel="λ", ylabel="Значение",
x_limit=None):
    """
    Строит график с возможностью ограничения по оси X.

    :param x: Данные для оси X.
    :param ys: Список массивов для оси Y.
    :param labels: Список подписей для каждого массива.
    :param title: Заголовок графика.
    :param xlabel: Подпись оси X.
    :param ylabel: Подпись оси Y.
    :param x_limit: Максимальное значение для оси X.
    """
    plt.figure(figsize=(10, 6))

    # Ограничиваем данные, если задан x_limit
    if x_limit is not None:
        indices = [i for i, val in enumerate(x) if val <= x_limit]
        x = [x[i] for i in indices]
        ys = [[y[i] for i in indices] for y in ys]

    for y, label in zip(ys, labels):
        plt.plot(x, y, label=label, linewidth=1.5)

    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()
    plt.grid(True)
    plt.show()

plot_graph(lambdas,
           [data_avg_users],
           ['Среднее количество пользователей в системе'],
           ['Среднее количество пользователей в системе'],
           x_limit=x_limit
)
```

```
plot_graph(lambdas,
           [data_avg_waiting],
           ['Среднее время ожидания в системе'],
           ['Среднее время ожидания в системе'],
           x_limit=x_limit
)

plot_graph(lambdas,
           [data_output_intens],
           ['Выходная интенсивность'],
           ['Выходная интенсивность'],
           x_limit=x_limit
)
```

3d_plotter.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

directory = "../data/3d_data/adaptive/8by2"

files = os.listdir(directory)

print(files)

for f in files:
    basename, extension = os.path.splitext(f)

    if extension != '.txt':
        continue

    with open(os.path.join(directory, f), 'r') as acfile:
        size = int(acfile.readline().strip())
        dim = int(np.sqrt(size))

        acfile.readline()

        data = []
        for line in acfile:
            data.append(float(line.strip()))

        matrix = np.array(data).reshape((dim, dim))

        print(f"Matrix from file {f}:")
        print(matrix)

        # Масштабирование осей X и Y
        x = np.arange(matrix.shape[0]) * 0.01
        y = np.arange(matrix.shape[1]) * 0.01
        x, y = np.meshgrid(x, y)
```

```
        fig = plt.figure(figsize=(10, 7))
        ax = plt.axes(projection='3d')

        # Используем plot_surface вместо contour3D
        surf = ax.plot_surface(x, y, matrix, cmap='viridis',
edgecolor='none')

        # Добавляем цветовую полосу (color bar)
        cbar = fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)
        cbar.set_label('Value')

        # Преобразуем basename для zlabel
        formatted_basename = basename.replace('_', ' ').title()

        # Улучшаем угол обзора
        ax.view_init(30, -110)

        # Добавляем подписи осей
        ax.set_xlabel('Max Intensity')
        ax.set_ylabel('Min Intensity')
        ax.set_zlabel(formatted_basename)

        plt.show()
```