Министерство науки и высшего образования Российской федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (ТГУ)
Институт прикладной математики и компьютерных наук

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ОПОП
д-р техн. наук, профессор

_____ А.В. Замятин

«_____»_____2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

РАЗРАБОТКА ВЫЧИСЛИТЕЛЬНО-ИМИТАЦИОННОГО КОМПЛЕКСА МОДЕЛИ УЗЛА ОБРАБОТКИ ЗАПРОСОВ РАЗНОГО ТИПА

по направлению подготовки 01.04.02 Прикладная математика и информатика,
направленность (профиль) «Интеллектуальный анализ больших данных»

Благинин Алексей Леонидович

Руководитель ВКР
канд. физ.-мат. наук

_____ И.Л. Лапатин

«_____»_____2023 г.

Автор работы
студент группы № 932128

_____ А.Л. Благинин

«_____»_____2023 г.

Томск — 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современных условиях развития информационных технологий в обществе значительно выросла роль как Интернета, так и в общем случае получения доступа к различного рода распределенным ресурсам. В качестве таких ресурсов могут выступать, например, электронные записи к врачу, которые должны предоставлять людям множественный доступ к публичной информации, онлайн–консультациям, обзору предстоящих событий, службе поддержки. Помимо проникших во всех сферы жизни общества информационных сервисов, надобность в распределенных ресурсах возникает в специализированных областях жизнедеятельности и науки — облачные вычисления, автоматизированное производство, сетевые протоколы передачи информации и др.

В таком случае для достижения эффективности доступа к ресурсам требуется решить ряд задач, связанных со спецификой самого ресурса, например, какое время требуется для обработки запроса, в какое время ресурс будет доступен, как сократить количество утерянных или неучтенных запросов. Ко всему неизвестно, как и когда именно запросы будут приходить. Именно таким классом задач занимается теория массового обслуживания — изучение и моделирование при помощи математического аппарата различных ситуаций и схем доступа к распределенных ресурсам [1]. Теория массового обслуживания появилась благодаря А.К. Эрлангу, занимавшемуся задачами оптимизации линий телефонной связи [2].

Теория массового обслуживания оперирует такими понятиями как заявка, буфер, прибор, орбита и другие. С их помощью описывается модель системы, как раз состоящая, в общем случае, из входящего потока заявок и обслуживающего прибора. Существует множество разновидностей систем [3, 4], описывающих, соответственно различные ситуации обслуживания или доступа к ресурсам в реальных жизненных ситуациях. В частности такие модели применяются для моделирования работы сетевых протоколов модели OSI (ICMP, CSMA, AMQP, Ethernet и др.) [5, 6, 7, 8] и используются для их оптимизации и устранения заторов заявок при их резком увеличении в течение краткого временного интервала. Для подобной ситуации в теории массового обслуживания используют входящий поток заявок с управляющей марковской цепью — MMPP [9, 10]. Она модулирует интенсивность поступления заявок во времени,

тем самым имитируя свойственную реальному миру неоднородность поступления заявок при работе в сети.

Наряду с математическим аппаратом, имитационное моделирование [11] является эффективным методом исследования как систем теории массового обслуживания, так и в широком смысле технологией системного анализа. Само имитационное моделирование возникло ввиду двух факторов: развитие вычислительной мощности ЭВМ, давшее возможность эффективно использовать численные метода анализа, и недостаточность математического аппарата для исследования систем, выражающейся в необходимости экспериментального подтверждения получаемых аналитических результатов, так как они были получены при определенном ограничивающем условии [12]. Хоть имитационное моделирование и базируется на математической конструкции и опирается на ее логику, но при решении практической задачи применяются более гибкие методы, приводящие непосредственно к результатам численным. В частности это касается выбора алгоритмов и различных архитектурных решений в модели.

Существует несколько методологий моделирования, среди которых основными являются дискретно-событийное моделирование [13, 14] и агентное моделирование [15]. Дискретно-событийное моделирование заключается в последовательной обработке событий, происходящих в системе, в хронологическом порядке. Из специфики и показателей обработки событий и складывается общая характеристика работы системы. Агентное моделирование, в свою очередь, представляет собой взаимодействие отдельных элементов и подсистем, называемых агентами, которые снабжены своей собственной логикой функционирования и исполняют ее асинхронно и самостоятельно. Для экспериментального исследования систем теории массового обслуживания, как правило, применяется дискретно событийный подход, так как заявки могут быть лаконично представлены в виде событий, происходящих в системе.

Численные методы позволяют в полной мере исследовать рассматриваемую модель, а именно ответить на вопросы, касающиеся определения границ области применимости и наблюдения функционирования системы при нестандартных параметрах. Однако для получения достоверных результатов требуется проведение большого числа численных экспериментов и наличие специализированных инструментов для их анализа. Зачастую это является основным

ограничением в ходе имитационного моделирования — имея ограниченный объем вычислительных ресурсов и времени необходимо провести внушительное количество вычислений. Поэтому данная работа посвящена оптимизации процесса экспериментального исследования систем массового обслуживания различного рода.

Целью работы является разработка и реализация программного комплекса с набором инструментов для анализа систем теории массового обслуживания, который будет включать в себя алгоритмы вычисления характеристик, имитационную модель и средства для её эффективного использования.

Решение разработать программное обеспечение для моделирования было принято ввиду ряда аспектов исследовательской работы, которые требуют создания и анализа большего объема данных. Для этого требуется автоматизировать существенную часть процесса исследования ввиду ограниченного времени и ресурсов. Помимо этого, трудоемкость на последующих этапах работы значительно снижается благодаря структурированности получаемой из модели информации.

В рамках цели данной работы поставлены следующие задачи:

1. выработать функциональные требования к программному комплексу, учитывающие быстродействие, параллельность вычислений и возможность точечной конфигурации каждой модели;
2. разработать архитектуру имитационной модели и логику ее функционирования;
3. реализовать модель с учетом разработанной архитектуры и требований;
4. реализовать метод ускоренного обращения характеристической функции на основе ее дискретизации;
5. разработать и реализовать вспомогательные программные инструменты для проведения численных экспериментов и статистической обработки их результатов;
6. провести численные эксперименты и предоставить результаты, иллюстрирующие возможности разработанного программного комплекса.

Работа содержит 92 страницы, 28 рисунков, 2 таблицы, 101 источник.

В первом разделе описано описан метод моделирования систем массового обслуживания и алгоритм для его проведения. Во второй главе описана архитектура, процесс разработки и особенности реализации программного комплекса. В третьей главе описаны подходы к обращению характеристических функций для работы с асимптотическими результатами исследования моделей. В четвертой главе описывается процесс работы с реализованным программным комплексом на примере исследования модели RQ—системы с повторными вызовами и обратной связью и использования машинного обучения, где в качестве обучающей выборки выступают результаты имитационного моделирования.

1 Применение имитационного моделирования в теории массового обслуживания

Имитационное моделирование [11] является мощным инструментом для экспериментального изучения систем массового обслуживания. Его актуальность как метода исследования обусловлена следующими факторами:

1. принцип, лежащий в основе моделирования, довольно прост для понимания, что делает этот инструмент доступным для широкого круга исследователей, в том числе для тех, кто не имеет глубоких познаний в теории случайных процессов;

2. при использовании имитационного моделирования в связке со сложными средствами визуализации можно детально наблюдать процесс работы системы в различных ее аспектах, что дает данные, которые невозможно получить при помощи исключительно численного подхода к анализу, более сфокусированного на различных показателях функционирования системы;

3. аналитические методы в большинстве случаев опираются на решение специально составленных систем уравнений (система уравнений Колмогорова). В свою очередь, имитационное моделирование предлагает методологию, которая помогает рассчитать те же показатели без изменения основного подхода [16];

4. имитационное моделирование зачастую является способом подтверждения аналитических результатов при исследовании системы массового обслуживания методом асимптотического анализа и определения их область применимости. Поскольку в качестве аналитического решения, как правило, выступает аппроксимация некой функции, то для установки ее качества требуется применение численных методов исследования, среди которых имитационное моделирование предоставляет широкий набор возможностей.

Система массового обслуживания описывает при помощи специального математического аппарата [1] порядок обработки заявок — абстрактных объектов, которые в реальном мире могут выступать в качестве телефонных звон-

ков, сетевых запросов или клиентов, требующих обслуживания. Для описании работы системы используется ряд основных элементов, которые проводят операции над заявками:

1. входящий поток — случайный поток событий (заявок), возникающих в системе и требующих обработки;

2. обслуживающий прибор — второй важный элемент системы массового обслуживания, представляющий собой случайных процесс обработки заявок. В системе может быть несколько приборов, при том их блок может также иметь различную топологию: они могут обслуживать заявки последовательно, либо параллельно. Также на конфигурацию системы влияет и выбор самого количества обслуживающих приборов;

3. Для описания класса систем массового обслуживания, в которых заявка может не суметь получить доступ к прибору, используется источник повторных вызовов, иначе орбита, которая агрегирует заявки, которые не смогли захватить прибор. Через некоторое случайное время они попытаются повторить попытку захвата;

4. Для регулирования поступления заявок на прибор используется очередь, иначе буфер, который накапливает входящие заявки прежде чем они попадут на прибор.

Все перечисленные элементы являются случайными процессами и могут иметь различные характеристики и распределение случайных величин. Задачей теории массового обслуживания является выбор наиболее эффективной структуры системы для ее наилучшей производительности. Данная задача решается путем нахождения и анализа характеристик ее функционирования. Среди аналитических методов решения этой задачи можно выделить метод асимптотического анализа [17] и метод диффузионного анализа [18]. Имитационное моделирование, в свою очередь, является как методом апробации полученных аналитических результатов при помощи численных экспериментов, так и подходом к исследованию систем массового обслуживания, заключающимся в имитации работы исследуемой модели системы массового обслуживания посредством компьютерных вычислений.

Как упомянуто выше, ключевое преимущество имитационного моделирования — прямолинейная реализация алгоритма, что экономит время и снижает трудоемкость исследования. Из относительно простой реализации алгоритмов вытекает и легкое конфигурирование и изменение структуры системы с возможностью быстрого получения результатов в последствии.

При моделировании систем массового обслуживания используется дискретно-событийный подход. Он заключается в генерации случайных величин, имеющих требуемое распределение, которые являются моментами наступления событий в системе. События могут быть различного рода: окончание обслуживания заявки, перемещение заявки на прибор, поломка прибора и т. д. В момент наступления события состояние системы меняется, что и отслеживается в процессе моделирования.

Рассмотрим подход подробнее на примере следующей системы:

на вход системы поступает пуассоновский поток заявок. Заявка занимает прибор в случае, если он свободен; тот, в свою очередь, начинает ее обслуживание в течение экспоненциально распределенного времени с параметром μ. В ситуации, когда прибор уже занят обслуживанием, входящая заявка, после неудавшейся попытки захватить прибор, мгновенно уходит на орбиту и осуществляет там случайную задержу в течение экспоненциально распределенного времени с параметром σ.

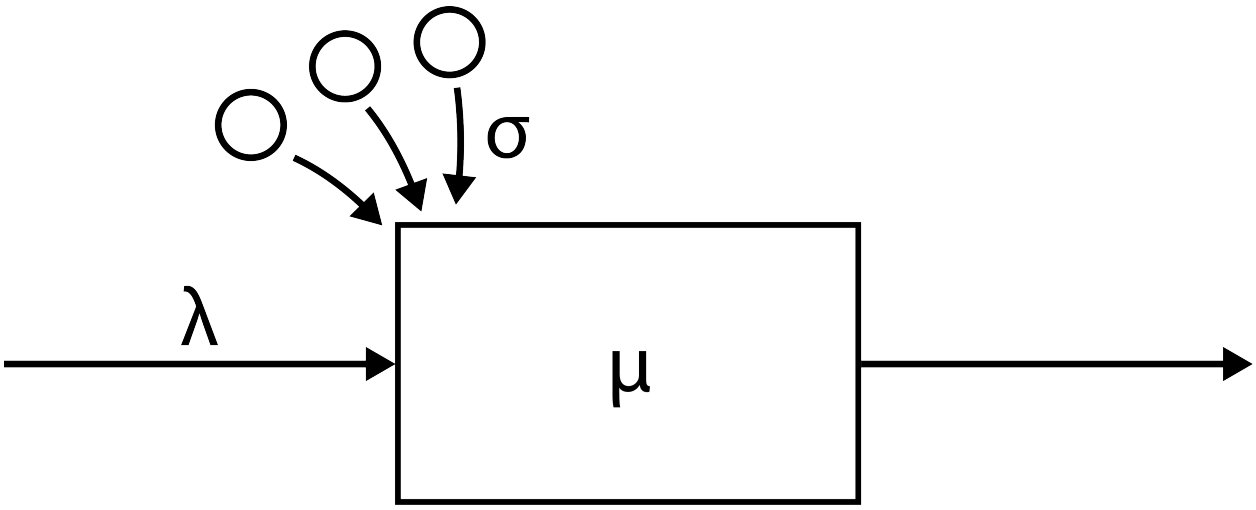

Рисунок 1 — Пример системы ТМО

Алгоритм моделирования функционирования представленной системы массового обслуживания при помощи очереди событий представлен ниже. Вве-

дем следующие обозначения:

1. Q — очередь, хранящая моменты наступления предстоящих событий в системе и функционирующая по принципу кучи, когда извлекаемый элемент является минимальным среди содержащихся в ней;
2. T_{curr} — текущее время моделирования;
3. T_{end} — момент окончания моделирования;
4. t_i — момент наступления события i в системе.

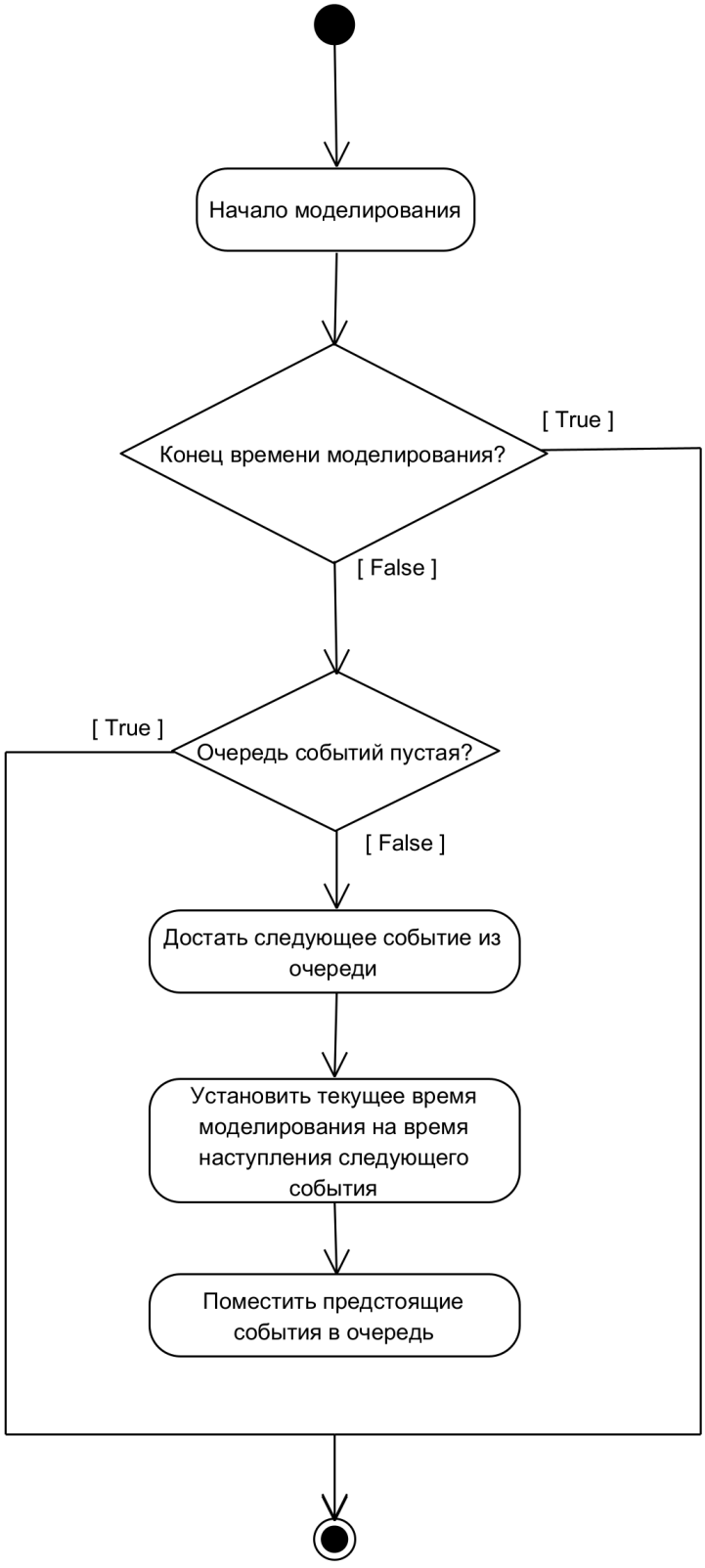

Рисунок 2 — Алгоритм моделирования

Так, при каждой итерации моделирования из Q достается момент наступления следующего события t_i, в свою очередь текущее время моделирования устанавливается как $T_{curr} = t_i$. Наступление события i позволит получить вре-

мя наступления последующего события t_j (например, заявка успешно поступила на прибор из входящего потока в момент t_i, и t_j будет являться момент окончания ее обслуживания), которое поместить в Q и итерация алгоритма завершится, и процесс будет повторятся, пока Q не опустеет, либо $T_{curr} = T_{end}$.

Однако для получения достоверных результатов при помощи имитационного моделирования требуется достаточный объем выборки данных, получаемый при работе алгоритма [19, 20]. Определение требуемого объема выборки является отдельной подготовительной задачей и может решаться эмпирическим путем подбора такого количества итераций алгоритма, при котором погрешность моделирования, выражаемая, к примеру, среднеквадратическим отклонением [21], становится несущественной.

Так как моделирование подразумевает сравнение получаемой выборки, в частности законов ее распределения, для проверки некой гипотезы о соответствии исследуемой математической модели, также требуется набор критериев и метрик для этого. В общем случае такой метрикой сравнения распределений может выступать расстояние Колмогорова, являющееся максимальной по модулю разницей между распределениями вероятностей:

$$\Delta = \max_{0<i<\infty} \left| \sum_{v=0}^{i} (P_0(v) - P_1(v)) \right|. \quad (1)$$

Данная формула также применима для случаев с многомерными распределениями вероятностей [22]. Помимо распределений можно находить и сравнивать разные числовые характеристики системы: коэффициент загруженности системы, среднее время обслуживания, соотношение потерянных заявок и обслуженных, соотношение времени работы системы и числа успешно обслуженных заявок. Перечисленные характеристики нетрудно получить при моделировании. Это дает возможность делать косвенные выводы о других аспектах работы системы, которые еще не получены аналитически, но доступны к изучению на основе наблюдений при моделировании. В таком случае, численные методы могут выступать как более приоритетные при решении задач, когда исследуемые характеристики наблюдаемы, но аналитически невыводимы.

2 Разработка программного комплекса

В данном разделе описывается процесс проектирования и разработки комплекса алгоритмов и программ для численного исследования процессов, протекающих в системах массового обслуживания, а также расчетов характеристик их функционирования. Ключевой составляющей данного комплекса является имитационная модель. Также программный комплекс содержит утилиты для вычисления дискретного преобразования Фурье, расчета расстояния Колмогорова, а также параметризованное решение системы уравнений Колмогорова для вычисления асимптотического приближения характеристической функции числа обслуженных заявок в системе массового обслуживания с повторными вызовами и обратной связью [23].

Проектирование программы начинается с разработки архитектуры по предварительно составленным требованиям. В данном случае, делается упор на универсальность модели и возможность встраивания ее в непосредственный процесс анализа. Также особое внимание уделяется процедуре проведения численных экспериментов. Численные результаты являются наиболее репрезентативными в случае, если было проведено достаточное количество экспериментов при наборах параметров, которые описывают функционирование модели в различных ситуациях. Такой подход позволяет удостовериться в верности теоретической стороны исследования посредством анализа объемной выборки данных о работе модели.

Архитектура программы была разработана с учетом следующих требований к ее работе и процессу взаимодействия с пользователем. Требования были составлены исходя из имеющегося опыта анализа систем массового обслуживания при помощи имитационного моделирования и опираясь на реализации имеющихся инструментов [24]:

1. производительность. Инструмент должен обладать достаточным быстродействием, чтобы в течение адекватного времени (от нескольких минут до пары часов) проводить имитационное моделирование даже на домашнем компьютере. Данное требование также обосновано необходимостью получать объемные выборки для получения достоверных результатов моделирования;

2. универсальность. Возможность моделирования широкого спектра моделей теории массового обслуживания;

3. Инструмент должен иметь возможность проведения масштабных экспериментов. Другими словами, программа должна быть оптимизирована для запуска большого количества моделей параллельно;

4. параметризация. Возможность настройки как работы элементов модели в отдельности так и модели в целом, что включает подбор типа распределения генерируемых задержек, самих параметров распределения и условного времени моделирования;

5. возможность производить сбор данных о работе модели в требуемом контексте. Иначе говоря, инструмент должен позволять собрать конкретные статистические данные имитационного моделирования в зависимости от поставленной задачи;

6. интеграция в процесс анализа. Другими словами, имитационное моделирование должно быть одним из инструментов среды, в которой проводится анализ. Из данного требования вытекает и выбор целевой программной среды для разработки, чтобы обеспечить легкость установки и начала использования программы в независимости от платформы;

7. легкая расширяемость. Исходный код программы должен предоставлять возможность добавления новой логики.

2.1 Предметная область

Исходя из выше представленных требований, инструмент должен иметь возможность производить моделирование различных систем массового обслуживания. Наиболее гибким вариантом реализации данного требования будет включить построение модели в процесс работы с инструментом. Так, у пользователя появляется возможность конфигурировать требуемую под задачу систему массового обслуживания.

Для достижения предложенного подхода предлагается использовать конструирование блоками [25] — инструмент будет включать в себя набор атомарных элементов систем массового обслуживания (обслуживающий прибор, вхо-

дящий поток и т.д.), которые будут работать по принципу черного ящика, однако связь между элементами и промежуточные этапы будут доступны для конфигурирования. Такой подход позволяет создать абсолютно любую вариацию системы массового обслуживания и используется не только в имитационном моделировании. Зачастую подобное архитектурное решение можно наблюдать в программных комплексах, предназначенных для программирования сложных взаимодействий между объектами. В 3D–графике зачастую используется схожий процесс работы, основанный на графах, где вершины представляют собой алгоритмы и объекты, а ребрами являются взаимодействия между ними. Например, в ПО для создания эффектов и симуляции взаимодействия физических тел Houdini используется именно такой подход [26].

Принцип работы имитационной модели базируется на обработке событий, происходящих в системе во время ее функционирования. Под событиями понимаются перемещения заявок по элементам системы. Обработка включает в себя регистрацию времени наступления события. Заявками же являются объекты, которые обрабатываются в ходе работе системе. Цикл обработки заявки, как правило, начинается с поступления ее в систему из входящего потока, далее поступление на прибор, обслуживание и последующее покидание системы. Именно из особенностей прохождения заявок через систему складываются характеристики ее функционирования.

Для разработки инструмента выбран объектно–ориентированный подход. Его преимуществом является возможность построения абстракций над логикой работы конкретных элементов системы, что позволяет удовлетворить требованию легкой расширяемости и позволить конструирование модели блоками в процессе анализа.

Объектно–ориентированный подход [27] к реализации предметной области подразумевает выделение из нее отдельных сущностей, которые могут быть представлены одним самостоятельным объектом, выполняющим конкретную задачу.

Процесс моделирования заключается в перемещении заявок от одного элемента к другому. В очередной условный момент времени T_{cur} местонахождение каждой заявки и ее характеристики могут меняться. По этой причине для каждой заявки вводится условный момент времени моделирования, когда она

изменит своё состояние — T_{shift}. В момент времени, когда $T_{shift} = T_{curr}$, заявка перемещается в очередной элемент модели, и T_{shift} обновляется им в зависимости от заданных параметров. Иначе процесс моделирования можно представить как последовательное изменение состояния системы, складывающегося из текущих состояний ее элементов. В свою очередь события, происходящее в системе могут быть поделены на несколько основных категорий:

1. прибытие заявки в систему;
2. начало обслуживания заявки на приборе;
3. окончание обслуживания заявки на приборе;
4. перемещение заявки в источник повторных вызовов (орбиту, при ее наличии в системе);
5. попытка повторного обращения из источника повторных вызовов.

В первую очередь, как ранее было указано, выделим элемент модели — объект, который содержит некую вычислительную логику и обрабатывает заявки в соответствии с этой логикой. Примером элемента модели является обслуживающий прибор.

Для того, чтобы соединить элементы в единую модель, потребуется определенном образом указать, как заявки будут перемещаться между элементами. Для этой цели выделим сущность — маршрутизатор. Однако, в зависимости от реализации элемента, он может иметь разное количество связей, с помощью которых будут перемещаться заявки. Таким образом, требуется ввести сущность, которая регулирует количество и качество возможных соединений элемента — слот.

Заявка также представлена сущностью, которая хранит в себе время смены ее состояния на новое, текущее состояние и другие опциональные атрибуты.

Поскольку элемент модели работает по принципу черного ящика, то не следует отягощать его логику сбором данным о моделировании. Для этой цели можно использовать маршрутизатор, отслеживая характеристики перемещения заявок в нужных участках модели. Исходя из этого, логику сбора статистики можно вынести в отдельную сущность, что позволит добавить в данный процесс разнообразную логику.

Итак, в предметной области программы были выделены следующие сущности:

- Элемент системы — объект, являющийся частью модели и выполняющий обработку заявок на протяжении ее работы.
- Заявка — объект, перемещающийся по системе и подлежащий обработке.
- Маршрутизатор — объект, образующий односторонний туннель для перемещения заявок от элемента системы к другому.
- Слот — объект, служащий для соединения двух элементов при помощи маршрутизатора.
- Сборщик статистики — объект, собирающий данные о перемещении заявок по маршрутизатору;
- Генератор задержки — служит для генерации случайных величин с требуемым распределением.

2.2 Объектная модель предметной области

На основе выделенных сущностей и описанного принципа работы программы была построена ее объектная модель, в последствии использованная для реализации. Для графического представления предметной области используется язык моделирования UML [28], в частности диаграмма классов [29], позволяющая графически изобразить выделенные в предметной области в качестве классов и взаимосвязь между ними.

Для обеспечения универсальности и возможности добавления новых разновидностей элементов модели был введен общий для них интерфейс Producer. Такое решение, в качестве побочного эффекта, также может позволять использовать различные элементы системы в несвойственных для них позициях в модели.

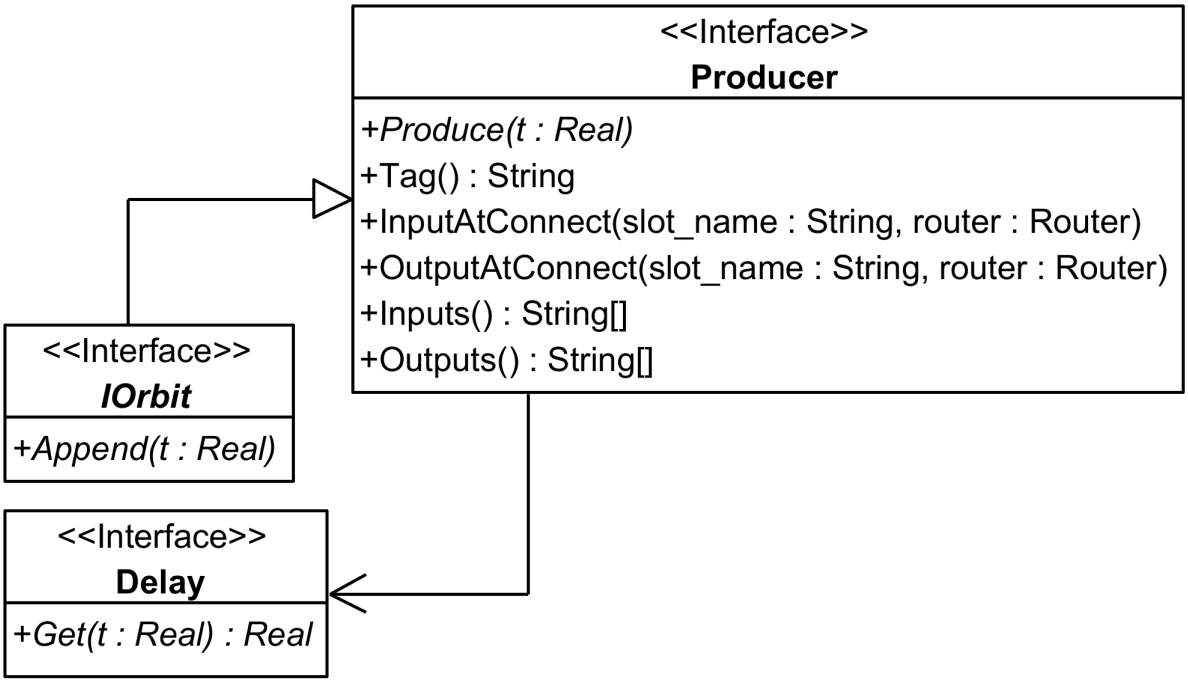

Рисунок 3 — Интерфейс Producer

В абстрактном методе Produce должна находится логика работы элемента системы, и именно этот метод будет вызываться в каждой итерации работы модели, возвращая вектор моментов времени наступления событий в элементе. Также в интерфейсе определены вспомогательные методы Tag, InputAtConnect, OutputAtConnect, Inputs и Outputs, служащие для соединения элементов между собой. Метод Tag возвращает константную строку, определенную для каждой реализации интерфейса Producer. Слоты элемента разделена на два класса — входящие, которые позволяют принимать элементу заявки, и выходящие, позволяющие ему их отправлять. Методы InputAtConnect и OutputAtConnect позволяют присоединять к маршрутизатору входящие и выходящие слоты соответственно, указывая название слота для соединения.

Другим важным элементом предметной области является объект, генерирующий время задержки заявок в процессе их перемещения и позволяющий проводить эксперименты, когда элементы системы могут задавать задержку для заявки разного рода. Для этой цели служит интерфейс Delay. Он имеет единственный метод Get, возвращающий сгенерированную случайную величину.

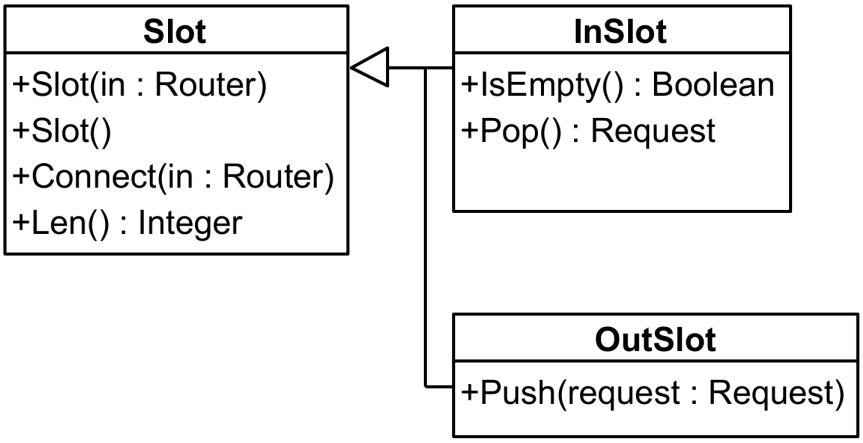

Рисунок 4 — Класс Slot и его наследники

Слоты используются в элементах системы для задания ограничения на возможные связи между другими элементами. К примеру, входящий поток может иметь только один выход для заявок, которые им генерируются, и не имеет входов, поскольку порождает заявки самостоятельно. В свою очередь, орбита должна иметь один вход для поступающих на нее заявок и один выход для отправления их обратно. Для разграничения логики работы слота на принимающий и отправляющий введено два наследника — InSlot, способный только принимать заявки при помощи метода Pop и OutSlot, предназначенный лишь для их отправления при помощи метода Push. Среда для переправки заявок между слотами обеспечивается за счет класса Router.

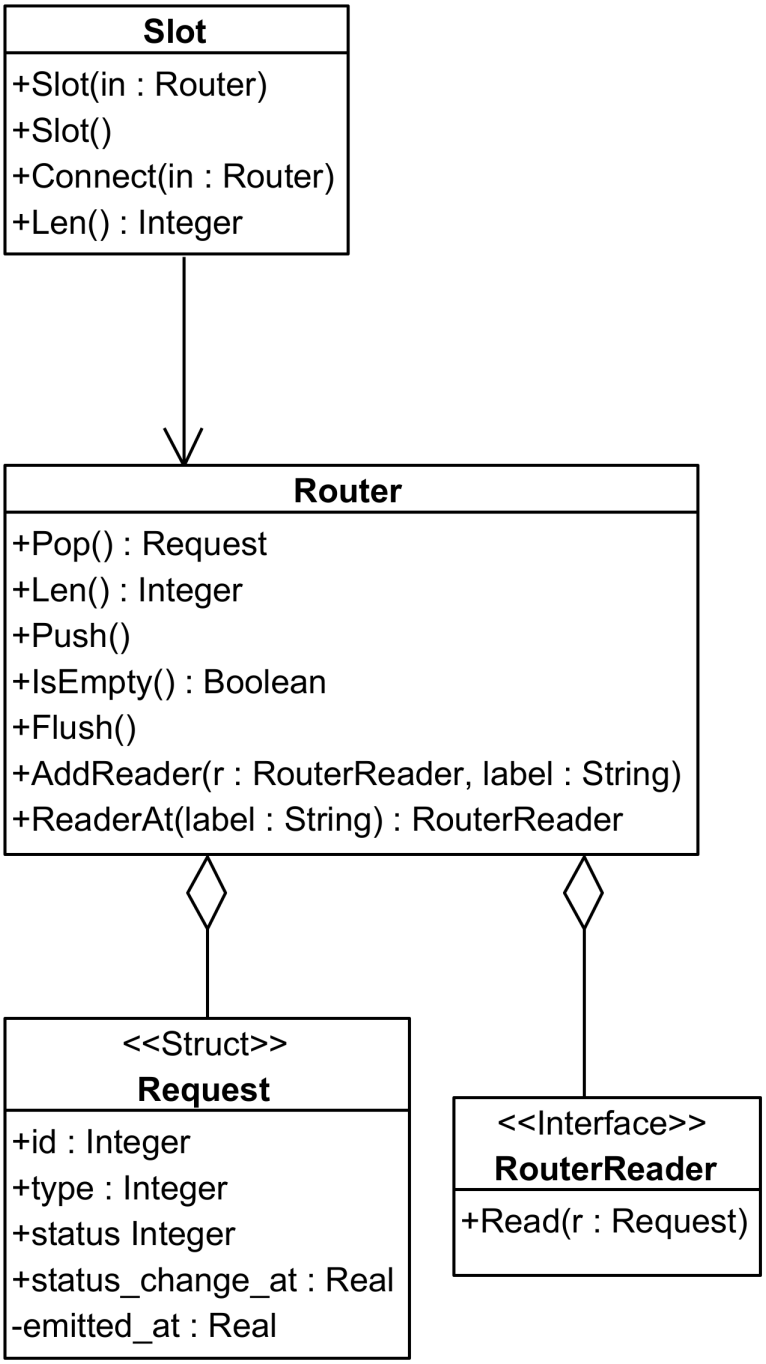

Рисунок 5 — Класс Router

Маршрутизатор представляет собой очередь, работающую по принципу FIFO. Таким образом, он позволяет элементам системы взаимодействовать с заявками между итерациями, когда очередная заявка еще ожидает обработки, но текущая итерация подошла к концу. Управление очередью заявок осуществляется при помощи методов Push (поместить заявку в очередь), Pop (достать заявок из очереди), Len (получить текущую длину очереди), IsEmpty (пуста

ли очередь). Методы AddReader и ReaderAt предназначены для управления набором сборщиков статистики. AddReader позволяет добавить новый сборщик статистики под указанным ярлыком, который позволит обращаться к нему в последствии при помощи метода ReaderAt для доступа к собранным данным.

Заявка представлена структурой Request, содержащей момент изменения ее состояния status_change_at (T_{shift}), тип, служащий для разделения способа происхождения заявки, статус, отражающий этап ее обработки (обслужена, в процессе обслуживания, в пути, покинула систему) и время, когда заявка была создана.

Ниже представлена полная объектная модель предметной области, объединяющая ранее представленные классы и интерфейсы.

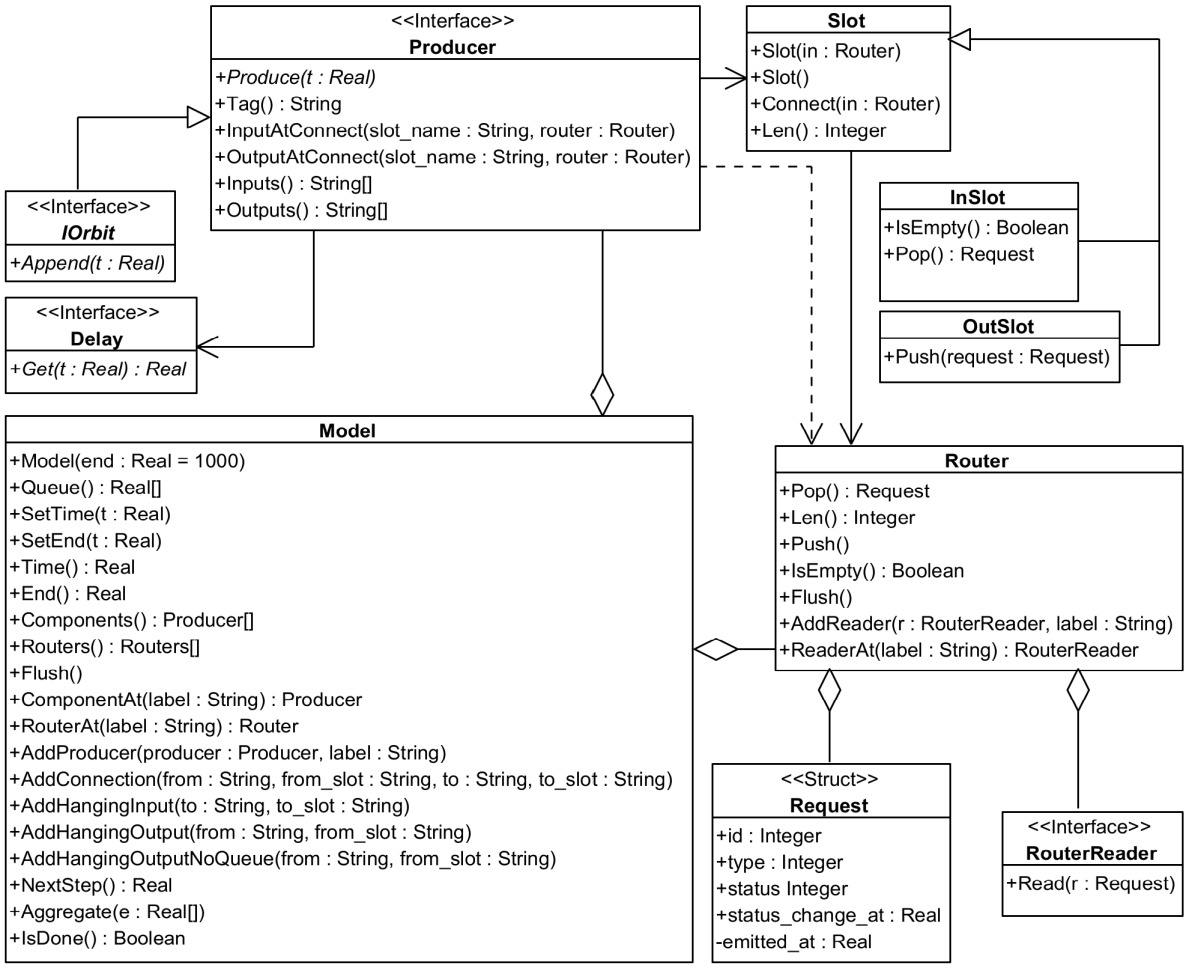

Рисунок 6 — Объектная модель предметной области

Как видно на рисунке 6, предметная область содержит специальный агрегирующий класс Model, предназначенный для управления процессом моделирования в рамках одной системы. Он позволяет построить модель при помощи

методов AddProducer (добавление нового элемента в модель) и AddConnection (добавление маршрутизатора между существующими элементами).
AddConnection имеет три частных случая: AddHangingInput — добавляет маршрутизатор на входящий слот одного элемента, AddHangingOutput — добавляет маршрутизатор на выходящий слот элемента, AddHangingOutputNoQueue — аналогичен AddHangingOutput, но заявки не сохраняются в маршрутизаторе. Три перечисленных метода позволяют вручную отправлять заявки в модель, а также устанавливать поток заявок между отдельными моделями.

Метод Queue возвращает текущий вектор моментов наступления событий в системе. SetTime, Time, SetEnd, End устанавливают либо возвращают текущее условное время и условное время окончания моделирования соответственно. Методы Components и Routers возвращают список имеющихся в модели элементов и маршрутизаторов соответственно. Методы ComponentAt и RouterAt предоставляют доступ к конкретному элементу или маршрутизатору. Метод Aggregate добавляет в вектор предстоящих событий новые, а метод NextStep сдвигает время моделирования на момент наступления ближайшего события и возвращает его. Метод IsDone позволяет узнать, закончилось ли моделирование (текущее время моделирования стало равно времени окончания моделирования).

Иначе содержимое модели можно представить графически следующим образом (рисунок 7):

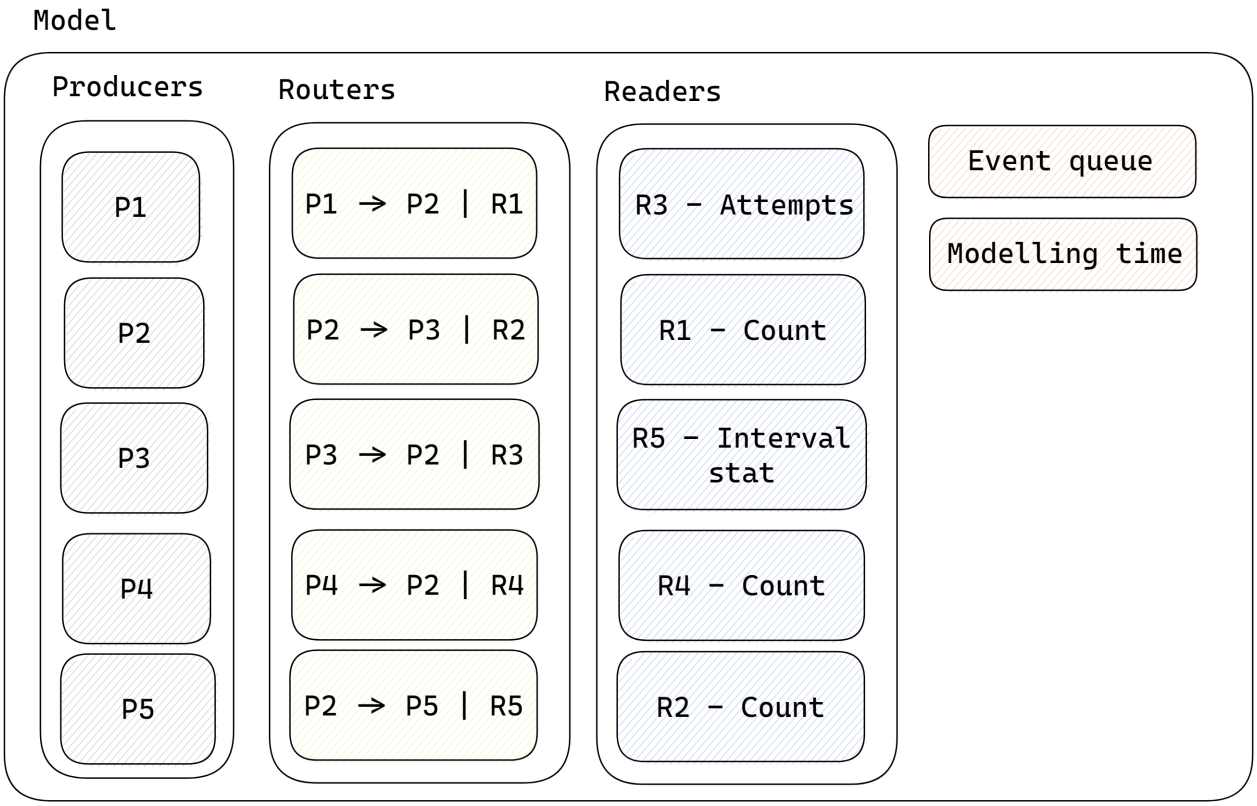

Рисунок 7 — Объектная модель предметной области

2.3 Реализации интерфейсов

В данном разделе представлены имеющиеся реализации интерфейсов и родительских классов, описанных ранее в предметной области, но не представленных на рисунке 6. Интерфейсы были выделены таким образом, чтобы предоставить возможность к легкому расширению имеющейся реализации. Поскольку логика работы каждого элемента модели, генератора задержек и сборщика статистики не зависит от реализации других элементов того же семейства классов, это позволяет без труда создавать новые элементы со сложной логикой без надобности изменения имеющихся реализаций. Наглядным примером данной особенности является интерфейс Producer, предназначенный для работы в качестве элемента модели. Для него существует три ряда реализаций, разделенных по предназначению.

Первым классом являются обслуживающие приборы. Среди них:

1. SimpleNode — реализация простейшего обслуживающего прибора, который обслуживает входящие заявки и передает их далее. В случае, если заявка не может получить доступ к прибору, она теряется;

2. RQNode — реализация обслуживающего прибора с орбитой. Функционирует аналогично простейшему прибору, однако заявки, не сумевшие попасть на прибор перемещаются на орбиту, откуда снова пытаются захватить его;

3. RQTNode — аналогичен прибору с орбитой, но обслуживает два типа заявок: входящие и приходящие из источника повторных вызовов.

Далее описаны источники заявок — объекты, имеющие только выход для порожденных заявок:

1. SimpleInput — реализация пуассоновского потока;

2. MMPPInput — служит для имитации работы Марковского модулированного пуассоновского процесса (MMPP). Для этого имеет поля, содержащие матрицу инфинитезимальных характеристик, вектор интенсивностей для каждого состояния, текущие состояние управляющей цепи, момент времени смены состояния управляющей цепи, а также метод для смены ее состояния shift. Позволяет моделировать ситуации, когда в системе периодически может случаться резкий всплеск заявок, например в реальной жизни пример работы MMPP будет являться резкий наплыв пользователей на интернет ресурс в час–пик.

Для орбиты имеется лишь одна реализация через наследованный от Producer интерфейс IOrbit, позволяющий при помощи отдельного метода вводить заявки на орбиту и рассчитывать их время нахождения там.

Отметим, что каждая из реализаций имеет в качестве параметра генераторы случайных задержек, что позволяет существенно менять характеристики функционирования элемента. Для этого интерфейса (Delay) имеются следующие реализации:

1. ExponentialDelay — генератор экспоненциально распределенного времени задержки;

2. UniformDelay — генератор равномерно распределенного времени задержки;

3. GammaDelay — генератор задержки, имеющей гамма распределение;

4. WeibullDelay — генератор задержки, имеющей распределение Вейбулла [30];
5. LognormalDelay — генератор задержки, имеющей логнормальное распределение.

Для маршрутизатора, помимо основной реализации, имеются два особых случая:

1. NoneRouter — очередь всегда пуста, а заявки, попадающие в нее, уничтожаются. Данный класс предназначен для тех случаев, когда в заявки надо направить в тупик после обслуживания и их учет не требуется;
2. OutputRouter — отличается от NoneRouter тем, что учет проходящих по очереди заявок ведется, однако они по-прежнему не хранятся.

Сборщики статистики:

1. IntervalRouterReader — сборщик статистики, класс для сбора информации о выходящих процессах системы. Имеет методы и поля для расчета характеристик ее работы, таких как среднее число обслуженных вызванных пришедших заявок, расчет двумерного, суммарного распределения, а также распределений выходящего процесса, обслуживающего входящие заявки и процесса, обслуживающие вызванные заявки. Также имеются методы, позволяющие рассчитать вариацию длин интервалов между моментами завершения обслуживания заявок для каждого из выходящих процессов;
2. AttemptCounter — сборщик, учитывающий количество раз, которое каждая заявка прошла через маршрутизатор, а также суммарное время, потраченное на перемещение до цели;
3. TimeCounter — сборщик, учитывающий моменты времени, когда заявки проходили через маршрутизатор.

2.4 Особенности реализации

В данном разделе описаны подробности реализации имитационной модели.

Исходный код, реализующий предметную область, выполнен на языке программирования C++ стандарта 2020 года (C++2a). Выбор в сторону C++ был сделан исходя из требований к инструменту, которые включают производительность и возможность интеграции непосредственно в процесс исследования, о чем подробнее изложено ниже. C++ отлично подходит для написания производительного программного обеспечения, а также обеспечивает гибкость в плане архитектурных решений и управления памятью. Помимо этого C++ является объектно–ориентированным языком программирования, что важно для реализации предметной области. Однако с некоторыми различиями, такими, как отсутствие интерфейсов в явном виде. Вместо них C++ поддерживает абстрактные классы с чистыми виртуальными функциями [31]. Помимо этого важными аспектами являются:

1. ручное управление памятью. С одной стороны, необходимость разработчику постоянно следить за выделяемой памятью и разрушением экземпляров объектов является недостатком, так как осложняет процесс разработки и способствует возникновению ошибок в управлении памятью, с другой — дает возможность организовать подходящий под задачу процесс выделения памяти;

2. компиляция. Компилируемые языки всегда превосходят в производительности интерпретируемые, поскольку проверка типов, предупреждение возникших исключений и других ошибок происходят на этапе компиляции, а не в процессе работы программы;

3. статическая типизация. Заранее известные типы объектов ускоряют как компиляцию, так и работу программы ввиду отсутствия проверки типов;

4. объявление без инициализации. C++ позволяет объявить переменную, не инициализируя ее некоторым значением, что экономит время работы.

Для сборки программы использовался компилятор GCC [32], поскольку он поддерживается всеми платформами, а также имеет широкий спектр параметров для оптимизации кода на этапе компиляции [33].

Также стоить отметить подход к генерации случайных величин во время моделирования. Поскольку получаемые величины по своей природе являются псевдослучайными, то генерируются они с использованием в качестве пара-

метров такта центрального процессора или системного времени. При использовании такого генератора во время имитационного моделирования получаемые данные будут недостоверны, поскольку генерация будет производится крайне часто — для каждого элемента модели в каждой итерации, что вызовет появление повторяющихся значений в генерируемой последовательности чисел. Для решения данной проблемы используется вихрь Мерсенна [34] — алгоритм генерации псевдослучайных чисел, основывающийся на свойствах простых чисел Мерсенна. Алгоритм гораздо менее предсказуем, имеет большой период повторения и полное отсутствие статистической закономерности.

В стандартной библиотеке C++ имеется реализация вихря Мерсенна под названием MT19937 [35], где число в названии связано с величиной его периода — 2^{19937}, которой хватит на генерацию псевдослучайных чисел без повторения в течение продолжительного время моделирования.

2.5 Реализация программы в качестве программного пакета для Python

Ключевой особенностью реализуемого ПО является возможности работы с ним в среде, где проводится остальные операции по исследованию и анализу системы массового обслуживания. По этой причине инструмент является программным пакетом (библиотекой) для языка Python. Данный язык крайне удобен и популярен для проведения статистического анализа, машинного обучения и других научных исследований, а имеющая оболочка для интерактивной работы IPython позволяет объединить все аспекты работы в общей среде исполнения.

Python реализован на C++, что позволяет разрабатывать к нему расширительные пакеты не только на самом языке, но и на языке реализации, что дает преимущество в производительности и вариативность. Примером подобного пакета, является фундаментальная библиотека для вычислений линейной алгебры и работы с матрицами NumPy, которая лежит в основе многих других пакетов и написана на C++ [36].

Исходный код, относящийся к предметной области строго изолирован, чтобы предоставить максимальную расширяемость имеющихся интерфейсов. За привязку исходного кода с библиотеке Python отвечает библиотека Pybind11

[37]. Она позволяет скомпилировать код, написанный на языке C++ в пакет для Python, при этом код изначально может быть изначальное не предзначен для этого. Таким образом, становится возможным строго разделить интерфейс, с которым взаимодействует пользователь и внутреннюю логику работы программы.

Ниже приведен пример привязки исходного кода класса Router. Тело методов опущено для лучшей читаемости.

Код на C++:

```
class Router
{
 public:
 std::vector<Request> q;
 std::unordered_map<std::string, RouterReader &> readers;
 friend class InSlot;
 friend class OutSlot;
 friend class Slot;

 Router()
 void AddReader(RouterReader &r, std::string label)
 std::unordered_map<std::string, RouterReader &> &Readers()
 RouterReader &ReaderAt(std::string label)
 virtual Request Pop()
 virtual int Len()
 virtual void Push(Request request)
 virtual bool IsEmpty()
 virtual void Flush()
};
```

Листинг 1 — Исходный код класса Router

Как видно из листинга 1, маршрутизатор содержит очередь q, словарь ссылок на сборщиков статистики readers, а также объявления о дружественных классах для семейства классов Slot, поскольку они тесно связаны между собой. Ниже представлена привязка данного класса при помощи Pybind11, позволяющая инстанциировать объекты в интерпретаторе Python и работать с ними, но управление памятью и вызов функций будет происходить на стороне C++. Другими словами, с помощью такого подхода создается оболочка для реализации предметной области программы на C++:

```
 py::class_<Router>(m, "Router", "Request queue")
 .def(py::init())
 .def("len", &Router::Len, "Returns number of requests contained
    ")
 .def("push", &Router::Push, "request"_a, "Push request in queue
    ")
 .def("pop", &Router::Pop, "Pop request from queue")
```

```
  .def("is_empty", &Router::IsEmpty, "Check if queue is empty")
  .def("flush", &Router::Flush, "Clears router queue")
  .def("add_reader", &Router::AddReader, "reader"_a, "label"_a,
    py::keep_alive<1, 2>(), "Add request reader")
  .def("readers", &Router::Readers, py::return_value_policy::
    reference, "dictionary of readers")
  .def("reader_at", &Router::ReaderAt, py::keep_alive<1, 0>(), py
    ::return_value_policy::reference, "get reader")
  .def_readonly("pushed_count", &Router::pushed_count, "number of
     pushed requests")
  .def_readonly("popped_count", &Router::popped_count, "number of
     popped requests")
  .def_readwrite("__readers__", &Router::readers)
  .def_readonly("__q__", &Router::q);
```

Листинг 2 — Привязка класса Router при помощи Pybind11

Так, Pybind11 позволяет переименовать сигнатуры функций, в частности, указать новые названия методов, аргументов, а также сразу указывать описание и документацию к каждому методу и классу. То же самое возможно сделать для всего модуля в целом, что позволить совместить его в одном пакете с имеющимся кодом на Python, либо создать вокруг нее еще один слой интерфейса:

```
PYBIND11_MODULE(simulation, m)
{
  m.attr("__name__") = "q_analysis.simulation";
  m.doc() = "Python library for retrial queuing system modeling";
  ...
}
```

Листинг 3 — Объявление модуля Pybind11

После описания документации, например, для заявки (Request), в среде IPython она будет доступна при использовании этого класса, что упрощает процесс первичного ознакомления с пакетом и его использование:

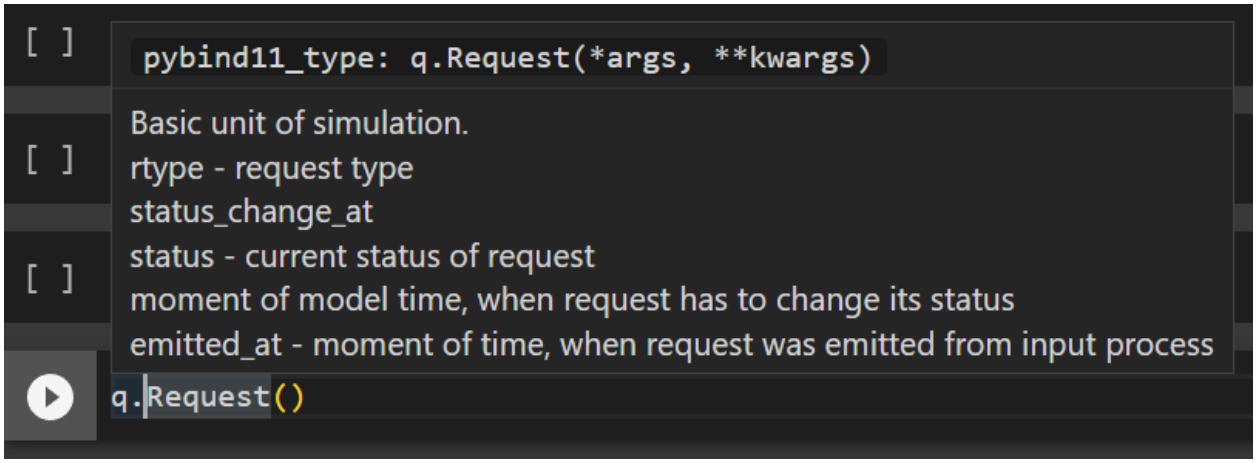

Рисунок 8 — Пояснение к структуре Request

Помимо этого, важно отметить, что при привязке требуется указать, как именно будет происходить распоряжение памятью при создании и удалении объектов. В Python все переменные являются указателями, которые содержать в себе счетчик ссылок, означающий количество ссылающихся указателей на область в памяти. Как только счетчик становится равным нулю, объект уничтожается интерпретатором, что может повлечь за собой различного рода ошибки доступа к памяти и ее утечку. Во избежание перечисленных ошибок Pybind11 позволяет указать время жизни объекта, являющегося аргументом и возвращаемым значением по принципу Nurse–Patient:

```
py::class_<Router>(m, "Router", "Request queue")
.def("add_reader", &Router::AddReader, "reader"_a, "label"_a,
   py::keep_alive<1, 2>(), "Add request reader")
.def("reader_at", &Router::ReaderAt, py::keep_alive<1, 0>(), py
   ::return_value_policy::reference, "get reader")
```

Листинг 4 — Указание времени жизни объекта и тип возвращаемого значения в Pybind11

В листинге 4 для метода add_reader указано выражение py::keep_alive<1, 2>(), что означает следующее: аргумент метода reader (индекс 2) не должен удалятся, пока не удален вызывающий метод объект (индекс 1). Данное выражение предотвратит ситуацию, когда аргумент reader может быть удален из памяти еще до того, как попадет в функцию add_reader. Иными словами, выражение keep_alive<i, j> указывает, что объект j должен прожить хотя бы столько же, сколько проживет объект i. Индексом, равным единице, всегда является вызывающий объект; последующие индексы относятся к аргументам; индекс, равный нулю, означает возвращаемое значение функции. Пример использования указания времени жизни возвращаемого значения можно наблюдать в привязке метода reader_at: поскольку данный метод является аналогом оператора индексации, крайне важно, чтобы ссылка на объект, возвращаемая методом не была удалена при обращении. Также для этого же метода указана политика возвращаемого значения — reference. Это означает, что при возращении значения из функции управление данным объектом не переходит к Python, и ответственным за своевременное удаление объекта из памяти является код на стороне C++.

Для корректной строковой репрезентации объектов пакета используется атрибут Python–объектов под названием __repr__, так как по умолчанию объекты классов, принадлежащие стороне C++ будут отображаться в формате <module.class_name object at <адрес в памяти> >.

2.6 Интеграция в процесс исследования

После успешной компиляции проекта становится возможным использовать программу как нативный пакет Python, что и позволяет встроить ее непосредственно в среду, где проводятся исследования. Зачастую аналитические результаты, полученные при помощи математического аппарата, проверяются при помощи моделирования, однако производится оно на крайне малой выборке — 2-3 запуска имитационной модели для получения совпадений в показателях. Причиной тому в том числе является подход к процессу запуска моделей. Как правило, их нужно сконфигурировать в специализированном ПО, провести моделирование, экспортировать результаты, импортировать их в среду для анализа и только далее приступить непосредственно к анализу. Также для него могут понадобится и сами конфигурации моделей. Подобный трудоемкий процесс является препятствием для получения достоверных результатов численными методами и отнимает большое количество времени на утилитарные действия. Наличие всех необходимых инструментов в одной среде значительно упрощает задачу проведения крупномасштабных экспериментов. Python используется [38] во многих областях науки как инструмент исследования. В особенности это касается анализа данных, так как для него реализован ряд зарекомендовавших себя программных пакетов — Pandas [39], позволяющий работать с данными в табличном виде и производить различные манипуляции и расчеты на их основе; NumPy [36] — пакет функций линейной алгебры и утилит для работы с многомерными данными; scikit-learn [40] — пакет для машинного обучения и статистического анализа; Scipy [41] — пакет, содержащий различные вычислительные алгоритмы и функции, и множество других. Таким образом, проведение моделирования на основе программного пакета Python предоставляет ряд преимуществ в удобстве, скорости и вычислительных возможностях.

С технической точки зрения данное решение также избавляет от необходимости реализации специальных классов для инстанциирования объектов с применением порождающих шаблонов проектирования (таких, как абстрактная фабрика [27]. В свою очередь процесс работы был вдохновлен известной библиотекой для машинного обучения PyTorch [42], где модель, в данном случае нейронная сеть, предварительно составляется из набора слоев требуемой конфигурации и уже в последствии обучается. Попытка использовать подоб-

ный принцип была сделана и в случае имитационной модели, где аналогичным образом для начала создается пустая модель, не содержащая элементов, затем ими наполняется и конфигурируется:

In [1]:
```
from q_analysis import simulation as q
model = rq.Model()
model.set_time(0)
model.set_end(1000000)
```

Добавление элементов:

In [2]:
```
model.add_producer(rq.SimpleInput(rq.ExponentialDelay(1.1),0,0),"input")
model.add_producer(rq.SimpleInput(rq.ExponentialDelay(0.6),1,0),"call")
model.add_producer(rq.Orbit(rq.ExponentialDelay(0.81)),"orbit")
model.add_producer(
            rq.RqtNode(rq.ExponentialDelay(1.3),
            rq.ExponentialDelay(1)),"node")
```

In [3]:
```
nodein = model.add_connection("input","out_slot","node","in_slot")
model.add_connection("call","out_slot","node","call_slot")
model.add_connection("orbit","out_slot","node","orbit_slot")
output = model.add_hanging_output_noqueue("node","out_slot")
model.add_connection("node","orbit_append_slot","orbit","in_slot")
```

In [4]:
```
model.router_at(output).add_reader(rq.TimeCounter(),"count")
```

В ячейке [1] производится создание модели и задание времени моделирования. Далее, в ячейке [2], производится добавление элементов в модель:

- простейший поток с экспоненциально распределенными моментами порождения заявок, интенсивностью 1.1 и меткой «input»;
- простейший поток с экспоненциально распределенными моментами порождения заявок, интенсивностью 0.6 и меткой «call» в качестве источника вызываемых заявок;
- орбита с интенсивностью повторного обращения заявок 0.81 и меткой «orbit»;
- обслуживающий прибор с повторными обращениями и обратной связью, обрабатывающий входящие заявки в течение экспоненциально распределенной задержки с интенсивностью 1.3 и вызываемые заявки в течение экспоненциально распределенной задержки с интенсивностью 1. Имеет метку «node».

Важно отметить, что для начала использования программного пакета достаточно установить его при помощи пакетного менеджера Python — pip [43]. После этого он готов к работе, и его потребуется лишь импортировать, как это и проиллюстрировано в ячейке `[1]`. Такая простота установки обеспечивается специальном форматом предварительно собранных пакетов Python — wheel [44], экземпляр которого создается в результате компиляции и сборки программного пакета при помощи утилиты setuptools [45]. В последствии, для установки пакета из формата wheel не потребуется наличие исходного кода, компилятора или всей цепочки инструментов, использовавшихся при сборке, что позволяет легко распространять пакет и использовать его.

В ячейке `[3]` производится маршрутизация заявок между элементами модели:

- заявки из входящего потока «input» попадают на прибор «node»;
- заявки из источника вызываемых заявок «call» попадают на прибор «node»;
- заявки, находящиеся на орбите «orbit» пытаются попасть обратно на прибор «node»;
- заявки, не сумевшие захватить прибор «node» отправляются на орбиту «orbit»;
- заявки, успешно обработанные прибором «node» отправляются в тупик.

В ячейке `[4]` к маршрутизатору, который представлен как тупик для обслуженных заявок, добавляется сборщик моментов прохождения заявок через маршрутизатор «count». После этого модель готова к запуску и графически может быть представлена как изображено на рисунке 9.

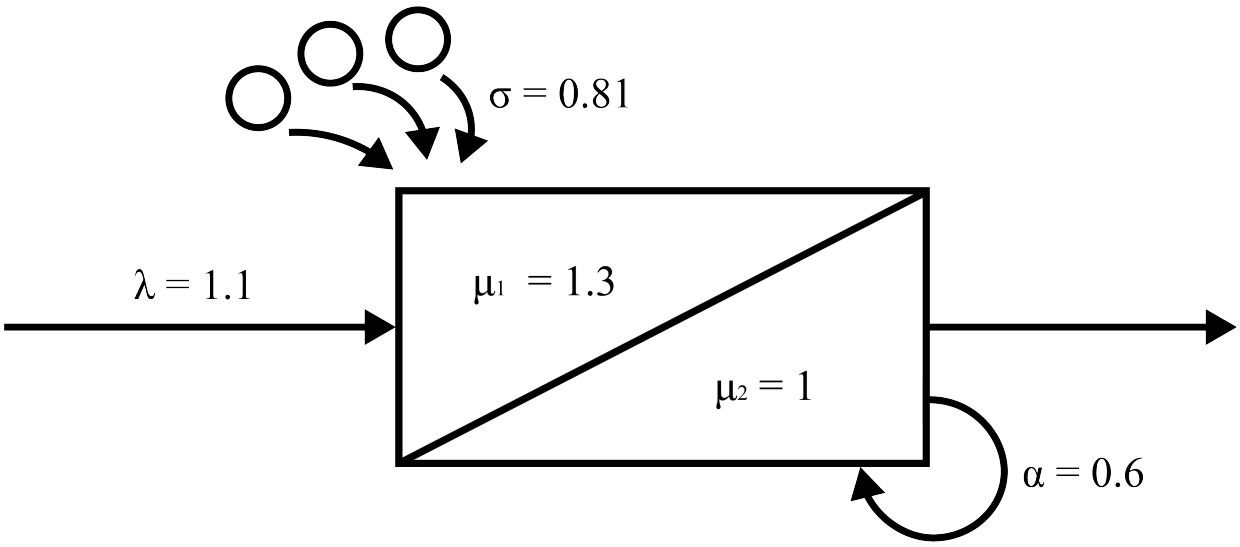

Рисунок 9 — Пример составленной модели

Таким образом, реализация программного пакета в качестве библиотеки для Python позволяет проводить численные эксперименты непосредственно в среде, где проводится статистический анализ, апробация результатов и остальные аспекты исследования.

3 Вычисление асимптотических результатов

Имитационное моделирование позволяет нам получать большое количество информации о работе системы, и во многих задачах оно используется для определения области применимости асимптотических результатов. В зависимости от задачи они могут быть получены и вычисляться легко, но в других случаях они могут быть представлены в виде характеристической функции, а для анализа проводится поточечное сравнение распределений вероятности исследуемой характеристики системы. Такая функция может содержать довольно трудно поддающиеся вычислению элементы, среди которых находятся интегралы, матричные экспоненты и др., поскольку получена она была при помощи математического аппарата без первостепенной задачи оптимизации ее вычисления. И на этапе, когда требуется получить значения такой функции, процесс оказывается длительным и трудоемким, так как переход от характеристической функции к распределению вероятностей производится при помощи обратного преобразования Фурье, использующее интегрирование. В ситуации, когда характеристическая функция имеет два, три и более аргументов, вычислительная сложность возрастает во много раз в виду вложенных интегралов. К проблеме трудоемкости также добавляется надобность производить большое количество вычислений, например для пятисот запусков имитационной модели, результатом которых будет являться двумерное распределение вероятностей размерностью 10 на 10 точек, потребуется 500 раз вычислить то же самое распределение посредством аналитических формул.

В данном разделе предлагается использовать теорию Фурье-анализа сигналов для обращения характеристических функций дискретных распределений. Идея заключается в применении принципов теоремы Котельникова [46, 47] для дискретизации характеристических функций и применении дискретного преобразования Фурье, которое вместо интегрирования использует суммирование и умножение, что гораздо эффективнее. В рамках разрабатываемого программного комплекса этот подход также применяется для обращения характеристической функции и упрощения вычислений, связанных с этим процессов. Данное решение может быть использовано как для анализа систем массового обслуживания, так и в других раздела теории вероятностей, где требуются множественные вычисления.

Как известно, любая случайная величина может быть однозначно определена функцией распределения вероятностей или характеристической функцией [48]. При этом характеристическая функция может быть определена для любой случайной величины (дискретной или непрерывной).

Объектом исследования теории массового обслуживания являются случайные величины и случайные процессы, которые могут быть как непрерывными (например, время ожидания заявки до начала обслуживания, период занятости системы, объем занятого ресурса), так и дискретными (например, число заявок в очереди, число событий, наступивших в потоке за некоторый промежуток времени, число занятых каналов обслуживания). При этом эти величины по своей природе принимают в основном неотрицательные значения.

Для неотрицательных случайных величин характеристическая функция является частным случаем преобразования Лапласа-Стилтьеса от функции распределения вероятностей с мнимым аргументом. Соответственно, характеристическая функция обладает свойствами преобразования Лапласа-Стилтьеса. С другой стороны, в общем случае взаимное соответствие характеристической функции и функции распределения вероятностей соответствует теории Фурье-анализа, безусловным преимуществом которого является свойство двойственности, то есть схожесть переходов от одной функции к другой. Это позволяет, решая задачу для характеристических функций, в последствии по явным интегральным формулам переходить к функциям распределения.

Для дискретных распределений принято использовать не характеристическую функцию, а производящую [49], но во многих работах для дискретных распределений используется тоже характеристическая функция. Это позволяет получать формулы перехода к распределению в терминах рядов Фурье через интегрирование характеристической функции по периоду 2π. С точки зрения записи формул никаких проблем не возникает, но при проведении численных экспериментов, в зависимости от сложности вида характеристической функции, численное интегрирование является задачей либо очень трудоемкой, либо вообще невозможной.

3.1 Связь характеристических функций с сигналами и их спектрами

Характеристической функцией $h(u)$ случайной величины ξ называется

$$h(u) = M\{e^{ju\xi}\}. \tag{2}$$

Характеристической функцией $h(u)$ дискретной случайной величины ξ с распределением вероятностей p_i называется

$$h(u) = M\{e^{ju\xi}\} = \sum_{i=-\infty}^{\infty} e^{ju\xi_i} p_i. \tag{3}$$

Характеристической функцией $h(u)$ дискретной неотрицательной целочисленной случайной величины $\xi(\xi = 0, 1, 2, \dots)$ с распределением вероятностей $p_i(i = 0, 1, 2, \dots)$ называется

$$h(u) = M\{e^{ju\xi}\} = \sum_{i=0}^{\infty} e^{jui} p_i. \tag{4}$$

При этом для нахождения распределения вероятностей $p_i(i = 0, 1, 2, \dots)$ по характеристической функции $h(u)$ используется формула обращения

$$p_i = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-jui} h(u) du. \tag{5}$$

Можно заметить, что формулы (3) и (4) являются формулами разложения сигнала $h(u)$ в ряд Фурье в комплексной форме [50], а формула (5) нахождения вероятностей p_i совпадает с формулой вычисления коэффициентов ряда Фурье [50]. Таким образом, в терминах теории сигналов характеристическая функция (3) является непрерывным комплекснозначным периодическим сигналом, а распределение вероятностей $p_i(i = 0, 1, 2, \dots)$ является спектром этого сигнала.

3.2 Дискретизация характеристической функции

Основные вычислительные сложности в задачах теории массового обслуживания возникают как раз при интегрировании по формуле (5) для нахождения распределения вероятностей p_i. При усложнении вида полученной характеристической функции $h(u)$ в зависимости от вычислительных ресурсов компьютера вероятности p_i либо считаются долго, либо их вовсе не удается вычислить.

Для уменьшения вычислительной сложности нахождения вероятностей предлагается воспользоваться аппаратом Дискретного преобразования Фурье [51], которое в соответствие дискретному сигналу h_k (последовательность N значений) ставит в соответствие его спектральные отсчеты p_i^* (последовательность N значений)

$$p_i^* = \left| \frac{1}{N} \sum_{k=0}^{N-1} h_k \cdot e^{-j \cdot i \cdot \frac{2\pi}{N} k} \right|, i = 0, 1, \ldots, (N-1). \tag{6}$$

Таким образом, необходимо дискретизировать характеристическую функцию $h(u)$ на периоде 2π таким образом, чтобы дискретное преобразование Фурье p_i^* от него было максимально близко к p_i

$$\Delta u = \frac{2\pi}{N}, h_k = h(k * \Delta u), k = 0, 1, \ldots, N-1. \tag{7}$$

При дискретизации сигнала (характеристической функции $h(u)$) с шагом Δu его спектр (распределение вероятностей p_i^*) начинает дублироваться с периодом Ω

$$\Omega = \frac{2\pi}{\Delta u} = N. \tag{8}$$

В этом случае, если распределение вероятностей p_i на интервале от 0 до N равно или близко к 1, то периодизация спектра не внесет в него значимых изменений на периоде N. Если же интервал от 0 до N не содержит всей или почти всей информации о распределении, то при дискретизации характеристической функции и периодизации спектра будет происходить наложение копий с периодом N, что приведет к расхождению вычисленного p_i^* и истинного p_i. Факт дублирования спектра при периодизации сигнала и возможного его наложения лежит в основе теоремы Котельникова, которая определяет выбор частоты дискретизации, позволяющее избежать такого наложения спектров.

3.3 Иллюстрация работы подхода на биномиальном распределении

Рассмотрим пример дискретизации характеристической функции на примере биномиального распределения с параметрами n (количество испытаний) и p (вероятность наступления события).

Характеристическая функция будет иметь вид

$$h(u) = (q + pe^{iu})^n. \tag{9}$$

Продемонстрируем, что обращение характеристической функции можно провести как при помощи интегрального обратного преобразования Фурье для дискретных случайных величин (5), так и при помощи дискретного преобразования Фурье (6). Результат вычислений представлен на графике

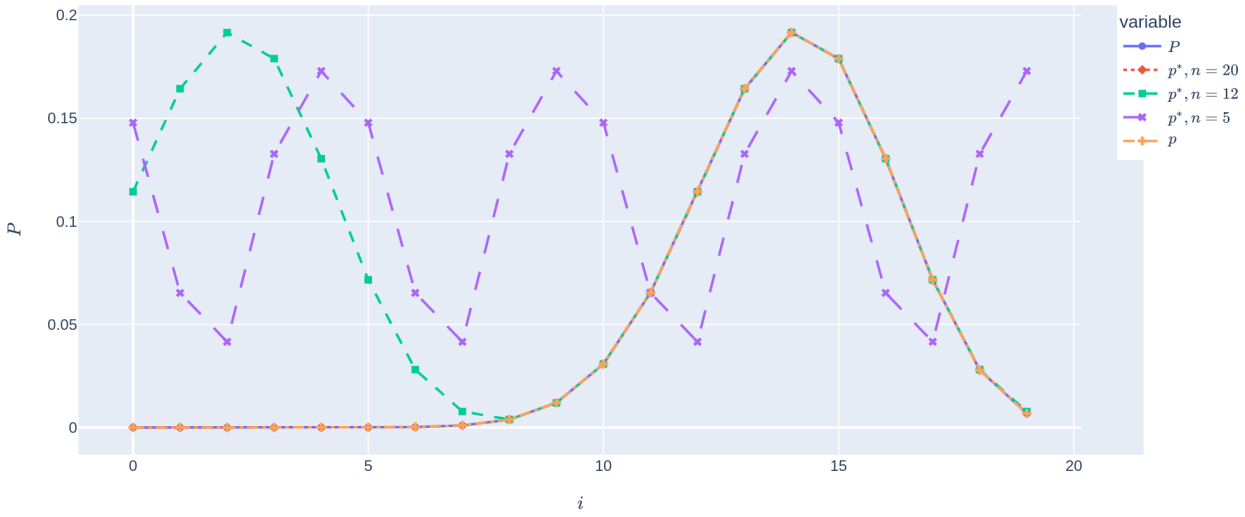

Рисунок 10 — Обращение характеристической функции

Вычисление было произведено на языке Python с заданными параметрами

$$n = 20, p = 0.7, N = \{20, 12\}.$$

Из рисунка 10 видно, что теоретическое биномиальное распределение P, результат обратного интегрального p (5) и дискретного p^* (6) преобразований совпадают при $N = 20$. Однако, как показывает график, при неверном выборе шага дискретизации (в данном случае $N < n, N = 12$) распределение начинает искажаться, так как условие теоремы Котельникова [46] не выполняется.

3.4 Иллюстрация работы подхода на задаче теории массового обслуживания

В данном разделе продемонстрируем эффективность предлагаемого подхода при реализации численных расчетов в задаче из [23]. В данной работе получено асимптотическое приближение характеристической функции $h(u, t)$ числа обслуженных заявок в системе с повторными обращениями и вызываемыми заявками

$$h(u, t) = \boldsymbol{R} e^{\boldsymbol{G}(u)t} \boldsymbol{E}, \tag{10}$$

которая при фиксированном t является функцией только от u.

Здесь матрица $\boldsymbol{G}(u)$ содержит коэффициенты системы дифференциальных уравнений Колмогорова. Ее элементы выражаются через параметры модели. $\boldsymbol{R}$ — вектор-строка, $\boldsymbol{E}$ — единичный вектор-столбец.

Для вычисления характеристической функции необходимо вычислять матричную экспоненту $e^{\boldsymbol{G}(u)}$, что безусловно делает задачу трудоемкой. Здесь матричную экспоненту вычисляем при помощи преобразования подобия матриц [52]:

$$e^{\boldsymbol{G}(u)} = \boldsymbol{T}(u) \cdot \boldsymbol{GJ}(u) \cdot \boldsymbol{T}(u)^{-1},$$

где $\boldsymbol{T}(u)$ — матрица собственных векторов $\boldsymbol{G}(u)$, $\boldsymbol{GJ}(u)$ — диагональная матрица собственных чисел Λ_n матрицы $\boldsymbol{G}(u)$.

Вычисление распределения вероятностей числа обслуженных заявок в системе за некоторое фиксированное время t через интегрирование с помощью формулы (5) является, как уже было упомянуто, достаточно трудоемкой вычислительной задачей. Для решения этой проблемы предлагается воспользоваться формулой (6) ДПФ.

На рисунке 11 видно, что результат дискретного преобразования Фурье (p^*) полностью совпадает с результатом вычисления при помощи интегрирования (p)

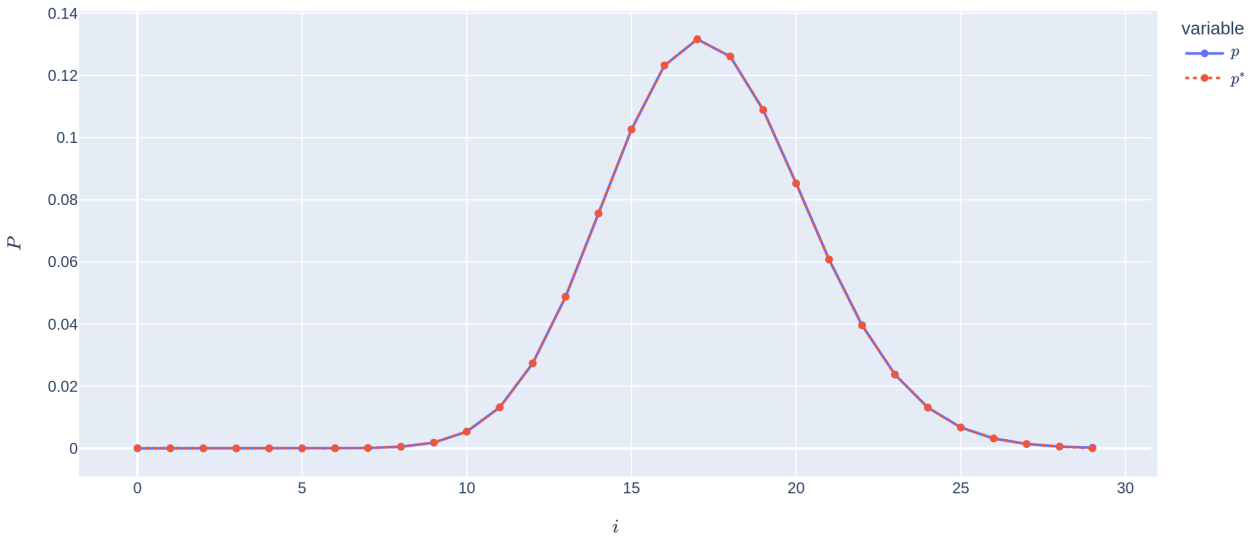

Рисунок 11 — Сравнение распределений вероятностей, полученных с помощью интегрирования и ДПФ

Для проверки скорости работы данного подхода к вычислению был проведен ряд тестов (400 запусков) со сравнением скорости работы алгоритмов и

точности получаемого распределения при помощи расстояния Колмогорова (1).

В среднем ДПФ быстрее интегрирования более, чем в 100 раз, а среднее расстояние Колмогорова — $4.16 \cdot 10^{-}6$.

Итак, предложенный метод позволяет в общем случае обращать характеристические функции гораздо более эффективно в сравнение с преобразованием Фурье, что в контексте разработанного программного комплекса позволяет быстрее вычислять асимптотические результаты и анализировать их, например, путем точечного сравнения распределений вероятности при помощи расстояния Колмогорова.

4 Работа с имитационной моделью

В данном разделе изложены подходы к работе с реализованным пакетом программ, показывающие течение процесса исследования при помощи имитационного моделирования и дополнительных утилит для расчета характеристик. В частности будет описан процесс моделирования с целью нахождения характеристик работы узла с двумя типами заявок и сравнения результатов с асимптотическими при помощи метода, предложенного в прошлой главе. Также представлен метод параллельного запуска множества моделей с разной конфигурацией для создания большой выборки.

4.1 Вычисление характеристик выходящих процессов модели RQ–системы с повторными вызовами и обратной связью

В главе 3.4 описаны асимптотические результаты, в частности приближение характеристической функции числа обслуженных заявок разного типа за время t. Последующей задачей ставится определение области применимости полученного приближения.

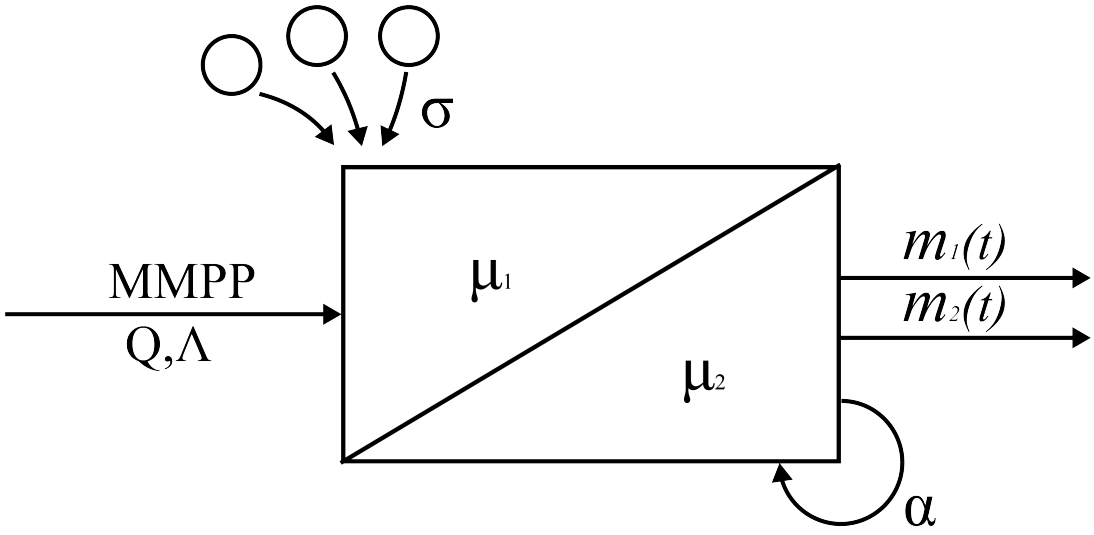

Рисунок 12 — Модель RQ–системы с повторными вызовами и обратной связью

Поскольку решение было получено при асимптотическом условии бесконечной задержки заявок в источнике повторных вызовов, требуется определить, при каких параметрах модели асимптотические результаты становятся недостоверными путем сравнения распределений вероятностей. Для это цели как раз отлично подходит имитационное моделирование.

В первую очередь, импортируем модуль q_analysis, который и является разработанным программным комплексом. Он, в свою очередь, содержит модуль simulation для имитационного моделирования.

In [5]:
```
import q_analysis.simulation as q
```

Создадим модель и установим время моделирования, равное 1000000 условных единиц.

In [6]:
```
model = rq.Model()
model.set_time(0)
model.set_end(100000)
model
```

Out[6]:
```
Model{ time: 0, end : 1000000, event_queue_len : 0, num_components : 0,
    num_connections : 0}
```

Далее добавим элементы модели (метод add_producer).

В модель добавятся следующие элементы:

- MMPP поток с матрицей Λ следующего вида

$$\Lambda = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.12 & 0 \\ 0 & 0 & 0.45 \end{bmatrix}$$

и матрицей Q

$$Q = \begin{bmatrix} -0.4 & 0.3 & 0.1 \\ 0.5 & -0.6 & 0.1 \\ 0.3 & 0.6 & -0.9 \end{bmatrix}$$

;

- Орбита с экспоненциальной задержкой с интенсивностью 1;
- Простейший поток с экспоненциальной задержкой с интенсивностью 1 (будет использоваться как источник вызываемых заявок прибора);
- Обслуживающий прибор с экспоненциальным временем обслуживания входящих и вызываемых заявок с интенсивностью 1.15.

Чтобы обращаться к элементам модели, им даются ярлыки - input, call, orbit, node.

In [7]:
```
model.add_producer(rq.MMPPInput(
[1,1.12,0.45],
[[ -0.4, 0.3, 0.1],
[0.5,-0.6,0.1],
[0.3,0.6,-0.9]],0,0),"input")
model.add_producer(rq.SimpleInput(rq.ExponentialDelay(1),1,0),"call")
model.add_producer(rq.Orbit(rq.ExponentialDelay(1)),"orbit")
model.add_producer(rq.RqtNode(rq.ExponentialDelay(1.15),rq.ExponentialDelay(
    1.15)),"node")
print("Components:",model.components())
```

Out[7]:
```
Components: {'node': RQTNode, 'orbit': Orbit, 'input': MMPP, 'call':
    SimpleInput}
```

Далее требуется соединить элементы модели при помощи маршрутизаторов. Для этого используется метод add_connection. В параметрах указывается ранее заданный ярлык элемента модели, потом его вход или выход, функция возвращает ярлык нового соединения.

In [8]:
```
model.add_connection("input","out_slot","node","in_slot")
```

В данном случае мы указали входящий поток input и его выход out_slot как источник заявок, и прибор node и его вход in_slot как вход для приходящих заявок. Получим следующий ключ, описывающий соединение:

Out[8]:
```
'input:out_slot:node:in_slot'
```

С помощью полученной строки в последствии можно обращаться к соответствующими маршрутизатору при помощи метода reader_at.

Далее добавляются остальные соединения. Для обслуживающего прибора маршрутизатор не будет сохранять в памяти обслуженные заявки. Это достигается использованием метода add_hanging_output_noqueue:

In [9]:
```
model.add_connection("call","out_slot","node","call_slot")
model.add_connection("node","orbit_append_slot","orbit","in_slot")
orb = model.add_connection("orbit","out_slot","node","orbit_slot")
output = model.add_hanging_output_noqueue("node","out_slot")
print("Connections",model.routers())
```

Out[9]:
```
Connections {
'onq:node:out_slot': Router{ queue_len: 0},
'orbit:out_slot:node:orbit_slot': Router{ queue_len: 0},
'node:orbit_append_slot:orbit:in_slot': Router{ queue_len: 0},
'input:out_slot:node:in_slot': Router{ queue_len: 0},
'call:out_slot:node:call_slot': Router{ queue_len: 0}
}
```

Ярлыки маршрутизаторов называются по следующей схеме:

- *ярлык выходного элемента : имя выхода : ярлык входного элемента : имя входа* ;
- для аккумулирующего входного маршрузитора (model.add_hanging_input) — *i : ярлык входного элемента : имя входа*
- для аккумулирующего выходного маршрузитора (model.add_hanging_output) — *o : ярлык выходного элемента : имя выхода*
- для выходного маршрузитора (model.add_hanging_output_noqeuue) — *onq : ярлык выходного элемента : имя выхода*

Добавим сбор статистики с интервалом в 20 условных единиц, что будет означать, что мы измеряем, сколько заявок было обслужено в системе за интервал времени, равный 20:

In [10]:
```
model.router_at(output).add_reader(rq.IntervalRouterReader(20),"stat")
model.router_at(output).readers()
```

Out[10]:
```
'stat': IntervalRouterReader
```

Теперь когда модель создана, нам требуется описать, как именно будет проходить ее запуск. Маршрутизация определяет топологию передачи заявок по системе, однако порядок взаимодействия элементов модели указывается в качестве отдельного алгоритма, так как, в зависимости от специфики задачи, он может варьироваться. Процесс моделирования сводится к итеративному вызову метода produce у элементов модели с указанием в качестве параметра текущего времени моделирования. Результат работы produce (список моментов наступления предстоящих событий) передается в функцию модели aggregate для того, чтобы далее смещать время на нужные моменты. Именно порядок

вызовов produce составляет алгоритм моделирования. В данно случае он будет следующим:

1. генерация очередной заявки входящих потоком;
2. возвращение заявок с орбиты;
3. генерация очередной заявки источником вызываемых заявок;
4. обслуживание заявки;
5. сбор заявок, не сумевших захватить прибор.

Построим цикл:

In [11]:
```
from tqdm import tqdm
e = model.end()
with tqdm(total=int(e)) as pbar:
t = c = 0
while True:
    c+=1
    told = t
    t = model.next_step()
    pbar.update(t-told)
    model.aggregate(model.component_at("input").produce(t))
    model.aggregate(model.component_at("orbit").produce(t))
    model.aggregate(model.component_at("call").produce(t))
    model.aggregate(model.component_at("node").produce(t))
    model.aggregate(model.component_at("orbit").append(t))
    if model.is_done():
        break
print("Time: ",model.time())
print("Iters: ",c)
```

Out[11]:
```
Time: 1000000.0437646629
Iters: 21710021
```

По окончании моделировании мы можем проверить, какое распределение вероятностей числа обслуженных заявок за интервал 20 получилось. Для этого обратимся к сборщику статистики маршрутизатора с ярлыком output и вызовем метод get_distribution_2d. Этот метод построит распределение вероятности числа обслуженных заявок двух типов — входящих и вызванных.

In [12]:

```
distr = model.router_at(output).reader_at('stat').get_distribution_2d()
import plotly.graph_objects as go
fig = go.Figure(data=[go.Surface(z=distr) ])

fig.update_traces(contours_z=dict(show=True, usecolormap=False,
highlightcolor="limegreen", project_z=True))

fig.update_layout(title='Model 2d distribution', autosize=False,
width=1000, height=1000,
margin=dict(l=65, r=50, b=65, t=90))

fig.show()
```

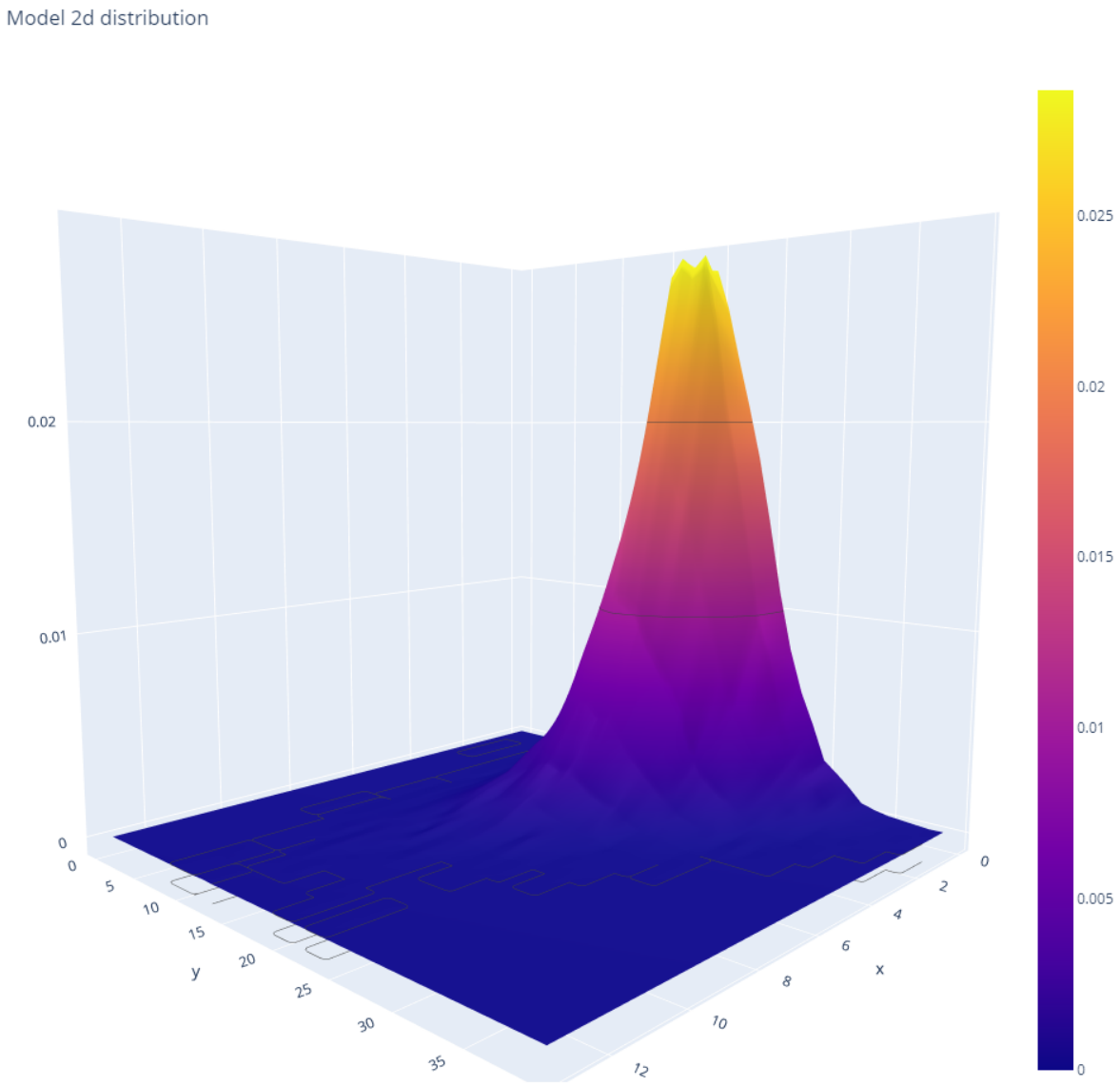

Рисунок 13 — Распределение числа обслуженных заявок по окончании имитационного моделирования

Далее получим характеристики выходящих процессов:

In [13]:
```
print("Mean called: {0},\nMean input: {1},\nVariation called: {2},\
    nVariation input: {3}".format(
model.router_at(output).reader_at('stat').get_mean_called(),
model.router_at(output).reader_at('stat').get_mean_input(),
model.router_at(output).reader_at('stat').get_variation_called(),
model.router_at(output).reader_at('stat').get_variation_input()))
```

Out[13]:
```
Mean called: 1.476929538590772,
Mean input: 19.793515870317407,
Variation called: 0.9226633215974042,
Variation input: 0.9463029320729057
```

В ячейке [14] представлены значения среднего числа обслуженных вызванных и входящих заявок, вариации числа обслуженных вызванных и входящих заявок соответственно.

Также мы можем экспортировать результаты в файл для работы в другой среде при помощи NumPy:

In [14]:
```
np.array(distr).tofile('example_distr.txt')
```

Таким образом мы смогли получить характеристики выходящих процессов рассматриваемой модели системы и визуализировать полученный результат, используя ту же среду выполнения. Далее нам потребуется вычислить ранее полученные асимптотические результаты для этой модели системы и провести сравнение. Для указанной системы в программном пакете уже имеется алгоритм вычисления значений характеристической функции числа обслуженных заявок. Для начала импортируем его и модуль с утилитами, в котором содержится функция для подсчета расстояния Колмогорова

In [15]:
```
from q_analysis import rq_system as rq
from q_analysis import utils as u
```

Далее зададим такие же параметры системы как при моделировании и рассчитаем распределение:

In [16]:

```
Lambda = np.mat([[1, 0, 0],
[0,1.12, 0],
[0, 0,0.45]])

Q = np.mat([[ -0.4, 0.3, 0.1],
[0.5,-0.6,0.1],
[0.3,0.6,-0.9]])
alpha = 1
mu1 = 1.15
mu2 = 1.15

s = rq.RQSystem(mu1, mu2, Lambda, Q, alpha)
n = 43
m = 13
distr_asymp =  s.icfft2(n,m,20)
```

При визуализации полученного распределения на рисунке 14 видно, что оно практически полностью совпадает с распределением, полученным в результате имитационного моделирования (рисунок 13)

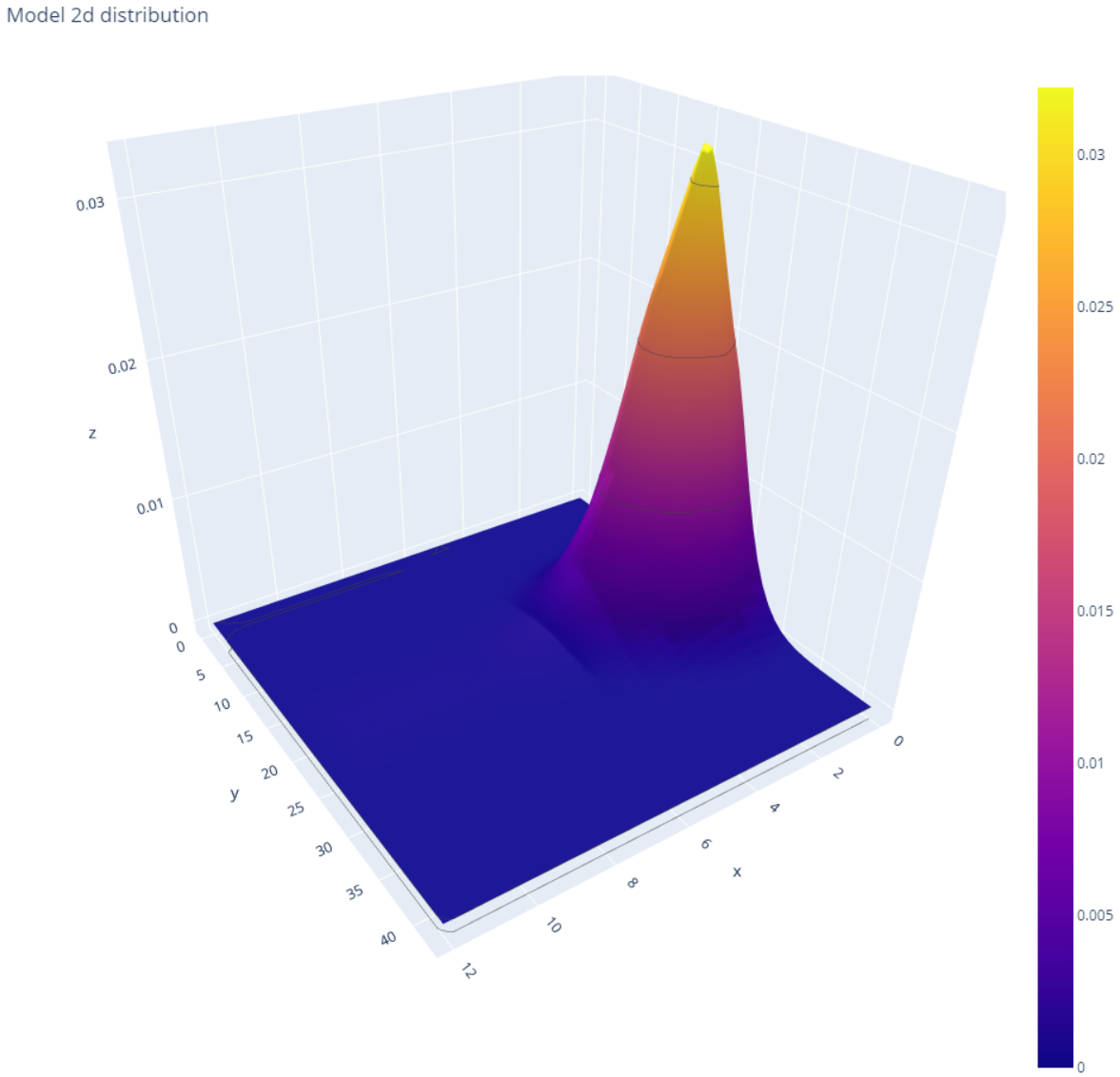

Рисунок 14 — Распределение числа обслуженных заявок

Для точечного сравнения распределений воспользуемся расстоянием Колмогорова:

```
In [17]: u.k_distance2(distr,distr_asymp)
Out[17]: 0.004355157840943953
```

Исходя из значения расстояния Колмогорова можно заключить, что при выбранных параметрах системы асимптотические результаты применимы, так как оно не превышает условного порога в $5 \cdot 10^{-3}$.

Таким образом, мы можем использовать данный инструмент как для по-

лучения наблюдаемых численных характеристик при отсутствии аналитических решений, так и использовать его для сравнения с ними при их наличии посредством вычисления различных показателей и средств визуализации.

4.2 Сбор и анализ данных о зависимости распределения заявок на орбите и времени ожидания

В данном разделе рассматривается применение имитационной модели для решения задачи, требующей большой выборки данных. Суть ее составляет подтверждение наличия зависимости между числом заявок на орбите и временем ожидания заявки до ее обслуживания на приборе в системе, изображенной на рисунке 12. Данная задача важна для понимания функционирования источника повторных вызовов и его влияния на эффективность обслуживания прибора. Поскольку требуется рассмотреть модель с различными параметрами орбиты и входящего потока, будет проводится ряд запусков, на основе которых в последствии будет производится поиск корреляции.

На число заявок на орбите, как и на время их обслуживания непосредственно влияют следующие параметры модели:

1. интенсивность входящего потока;
2. частота возвращения заявок с орбиты;
3. интенсивность обслуживания прибора.

Для конфигурации моделей интенсивность обслуживания будет зафиксирована, а интенсивность входящего потока и частота возвращения заявок с орбиты будет варьироваться.

Для хранения данных о моделировании и собираемых характеристиках работы моделей будем использовать словарь:

In [18]:
```
models = []
models_desc = []
```

Поскольку запусков будет много, будем запускать их в разных потоках для максимальной эффективности. Для этого объявим функции для генерации модели и запуска алгоритма. Это позволит нам использовать одну и ту же логику при параллельном запуске. Для упрощения задачи моделирования будет

использоваться простейший входящий поток заявок, также зафиксируем интенсивность вызова заявок на значении 1. Для подсчета времени ожидания нам достаточно обновлять для каждой прошедшей заявки момент времени, в который он уходила с орбиты. В свою очередь для подсчета количества заявок на орбите нам потребуется учитывать, сколько заявок пришло на орбиту, и сколько ушло. Для этой цели прикрепим к маршрутизаторам, по котором заявки проходят через орбиту, счетчики TimeCounter.

In [19]:
```
def sim_task(i):
   t = c = 0
   while True:
      c+=1
      t = models[i].next_step()
      models[i].aggregate(models[i].component_at("input").produce(t))
      models[i].aggregate(models[i].component_at("orbit").produce(t))
      models[i].aggregate(models[i].component_at("call").produce(t))
      models[i].aggregate(models[i].component_at("node").produce(t))
      models[i].aggregate(models[i].component_at("orbit").append(t))
      if models[i].is_done():
         print('#',end='')
         models[i].flush()
         break
```

In [20]:
```
def create_model(inp_i,orb_i,node_i):
model = rq.Model()
model.set_time(0)
model.set_end(100000)

model.add_producer(rq.SimpleInput(rq.ExponentialDelay(inp_i),1,0),"input")
model.add_producer(rq.SimpleInput(rq.ExponentialDelay(1),1,0),"call")
model.add_producer(rq.Orbit(rq.ExponentialDelay(orb_i)),"orbit")
model.add_producer(rq.RqtNode(rq.ExponentialDelay(node_i),rq.
    ExponentialDelay(1.15)),"node")

model.add_connection("input","out_slot","node","in_slot")
model.add_connection("call","out_slot","node","call_slot")
orb_i = model.add_connection("node","orbit_append_slot","orbit","in_slot")
orb_o = model.add_connection("orbit","out_slot","node","orbit_slot")
output = model.add_hanging_output_noqueue("node","out_slot")

model.router_at(orb_i).add_reader(rq.AttemptCounter(),"attempt_count")
model.router_at(orb_i).add_reader(rq.TimeCounter(),"count")
model.router_at(orb_o).add_reader(rq.TimeCounter(),"count")
return model
```

Далее сгенерируем модели. Интенсивность входящего потока будет находится в границах $[0.1, 1]$ с шагом 0.1, орбита — $[0.05, 3]$ с шагом 0.1, а интенсивность обслуживания в интервале $[1.6, 2]$ с шагом 0.1. Нижняя граница последнего интервала больше интенсивности входящего потока, так как условием стационарного режима для данной система является. Время моделирования установим в 200000 условных единиц, чего хватит для получения достоверных значений исследуемых характеристик. Для генерации моделей применим тройной вложенный цикл, итерации которого используют ранее установленные диапазоны:

In [21]:
```
input_intensity_range = np.arange(0.1, 1.0001, 0.1)
orbit_intensity_range = np.arange(0.05, 3.0001, 0.1)
node_intensity_range = np.arange(1.6001, 2.0001, 0.1)
m_index = 0

for inp in list(input_intensity_range):
   for orb in list(orbit_intensity_range):
      for nod in list(node_intensity_range):
         m = {}
         m['model_index'] = m_index
         m_index+=1
         m['input_intensity'] = inp
         m['orbit_intensity'] = orb
         m['node_intensity'] = nod
         models_desc.append(m)
         models.append(create_model(inp,orb,nod))
```

Для каждого запуска будем хранить заданные параметры в словаре, который в последствии также наполнится другими данными о моделировании, что позволит нам создать полноценную выборку для анализа:

In [22]:
```
for m in models_desc:
   print(m)
```

```
{'model_index': 0, 'input_intensity': 0.8, 'orbit_intensity': 0.8, '
    node_intensity': 1.55}
{'model_index': 1, 'input_intensity': 0.8, 'orbit_intensity': 0.8, '
    node_intensity': 1.65}
{'model_index': 2, 'input_intensity': 0.8, 'orbit_intensity': 0.8, '
    node_intensity': 1.75}
{'model_index': 3, 'input_intensity': 0.8, 'orbit_intensity': 0.8, '
    node_intensity': 1.85}
...
```

Итак, получилось 1500 моделей с различными параметрами. Запустим их в 12-ти процессах при помощи объекта ThreadPool из встроенной библиотеки Python multiprocessing [53], предназначенной для конкурентной работы:

In [23]:
```
print('#'*(len(models)-1))
from multiprocessing.pool import ThreadPool
r = list(range(0,len(models)-1))
with ThreadPool(processes=12) as p:
  p.map(sim_task, r )
```

Для последующего анализа рассчитаем ряд показателей:

- среднее число заявок на орбите;
- среднее время ожидания заявки до обслуживания;
- коэффициент вариации числа заявок на орбите;
- среднеквадратическое отклонение числа заявок на орбите;
- среднеквадратическое отклонение времени ожидания заявок;
- коэффициент вариации времени ожидания заявок.

Пример полученный таблицы с характеристиками:

Таблица 1 — Показатели для оценки зависимости числа заявок на орбите и времени ожидания заявки

λ	μ	σ	os_mean	wt_mean	os_std	wt_std	os_var	wt_var
0.100	1.600	0.050	2.044	20.395	1.487	35.386	0.728	1.735
0.100	1.700	0.050	2.035	20.144	1.484	34.628	0.729	1.719
0.100	1.800	0.050	1.994	20.198	1.489	34.612	0.747	1.714
0.100	1.900	0.050	1.969	19.485	1.480	33.644	0.752	1.727
0.100	2.000	0.050	1.998	19.317	1.466	33.433	0.734	1.731
0.100	1.600	0.150	0.734	7.293	0.898	12.226	1.223	1.676
0.100	1.700	0.150	0.688	7.049	0.861	12.026	1.252	1.706
0.100	1.800	0.150	0.688	7.028	0.871	11.913	1.267	1.695
0.100	1.900	0.150	0.689	6.956	0.871	11.813	1.264	1.698
0.100	2.000	0.150	0.689	6.903	0.866	11.907	1.258	1.725
0.100	1.600	0.250	0.436	4.349	0.686	7.321	1.574	1.683
0.100	1.700	0.250	0.450	4.524	0.700	7.633	1.557	1.687
0.100	1.800	0.250	0.440	4.411	0.691	7.292	1.569	1.653
0.100	1.900	0.250	0.428	4.349	0.681	7.338	1.590	1.687
0.100	2.000	0.250	0.429	4.320	0.684	7.223	1.593	1.672

После того, как выборка создана, рассмотрим распределение каждой из характеристик. На рисунке **??** представлены распределения вероятностей числа заявок на орбите во время моделирования. Как видно, для некоторых наборов параметров наблюдаются скопления похожих распределений. Также видно, что с изменением параметров математическое ожидание заметно смещается:

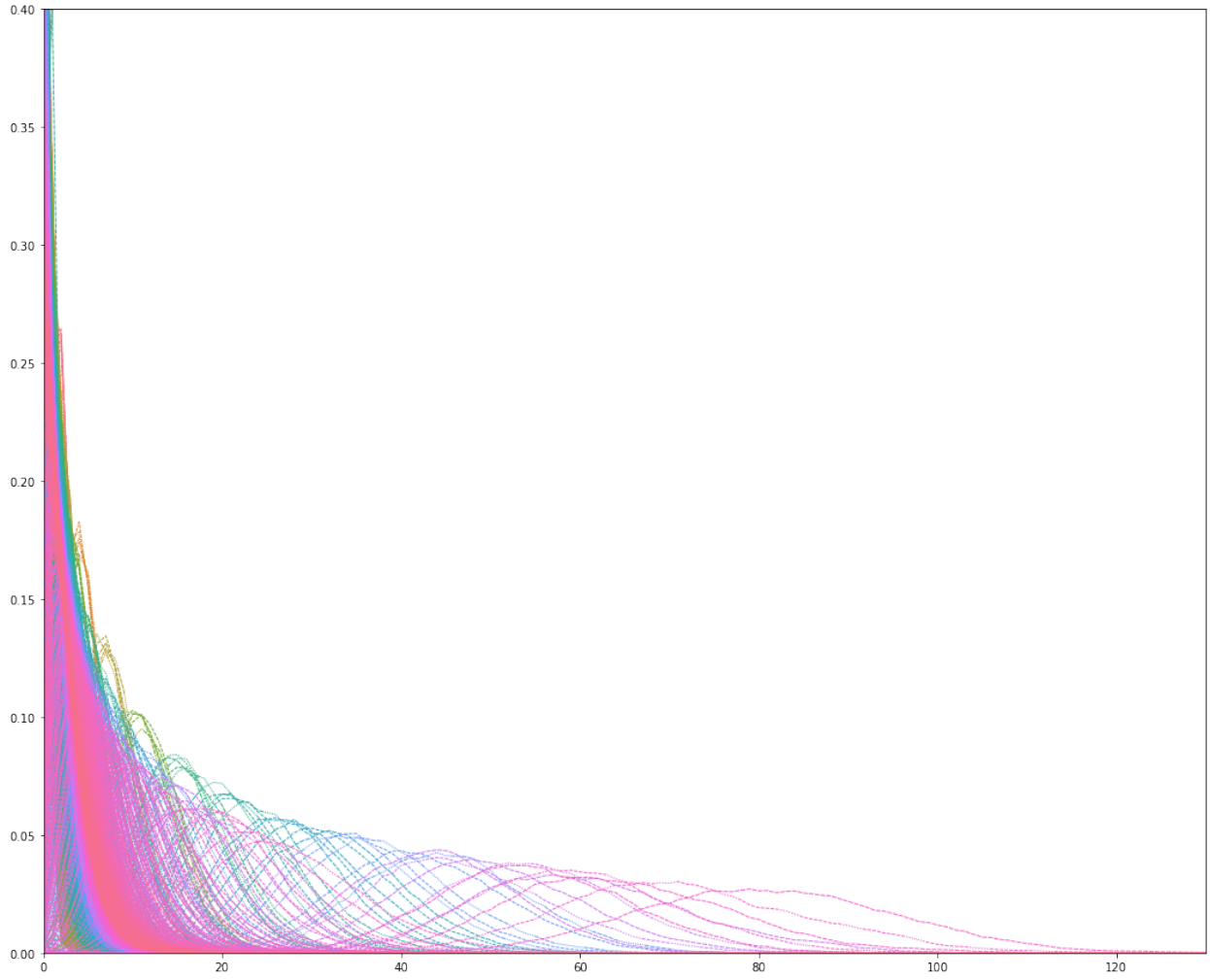

Рисунок 15 — Распределения числа заявок на орбите

На рисунке **??** представлены распределения времени до обслуживания заявки:

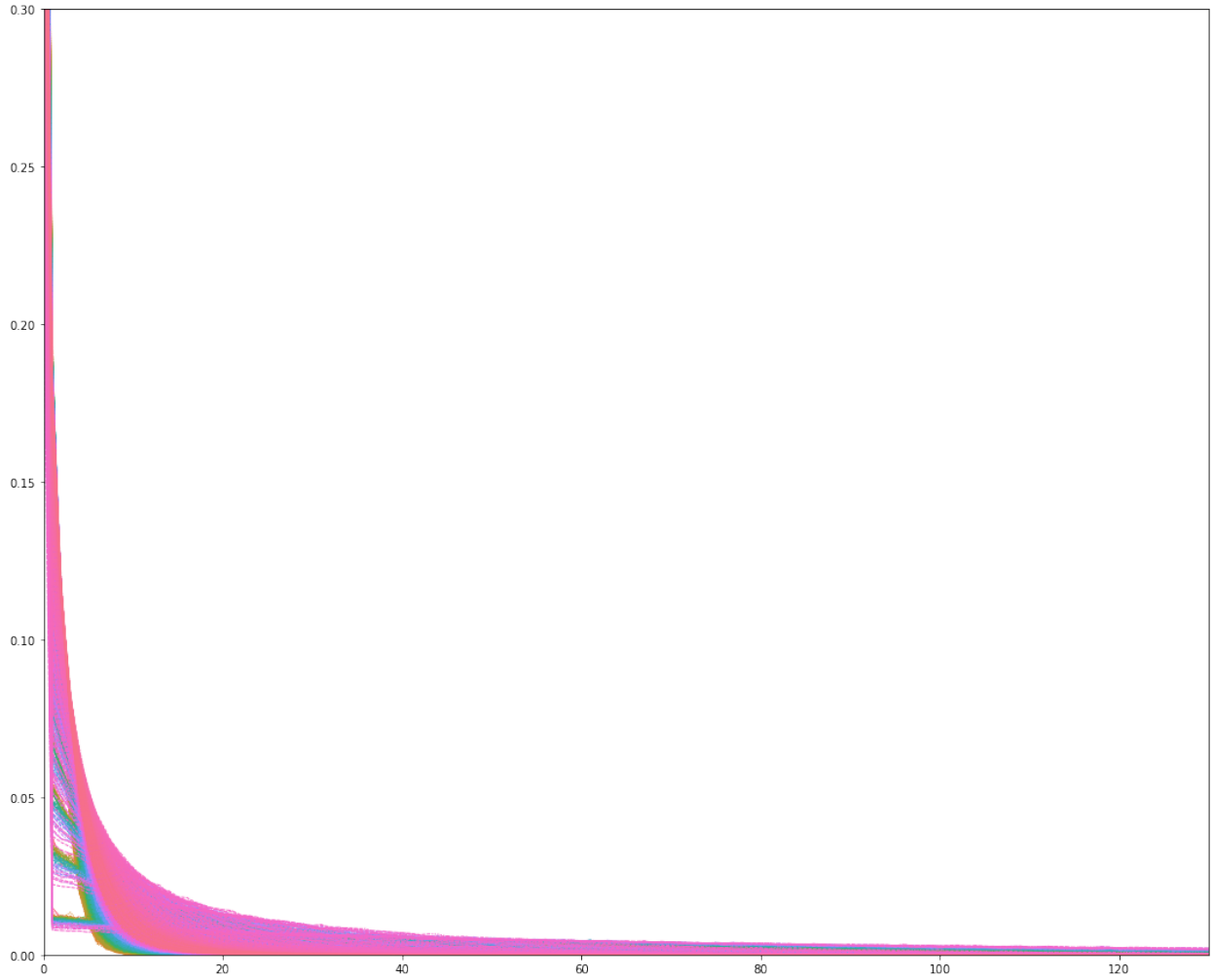

Рисунок 16 — Распределения числа заявок на орбите

Для того, чтобы выявить зависимости между показателями воспользуемся графиками рассеяния, где каждому наблюдению будет соответствовать точка на декартовой плоскости. По каждой из осей отложены рассматриваемые показатели, и в случае, если между ними имеется корреляционная зависимость, на диаграмме это будет представлено в виде определенной закономерности, согласно которой расположены точки. В случае положительной корреляции оба значения будут расти и наоборот:

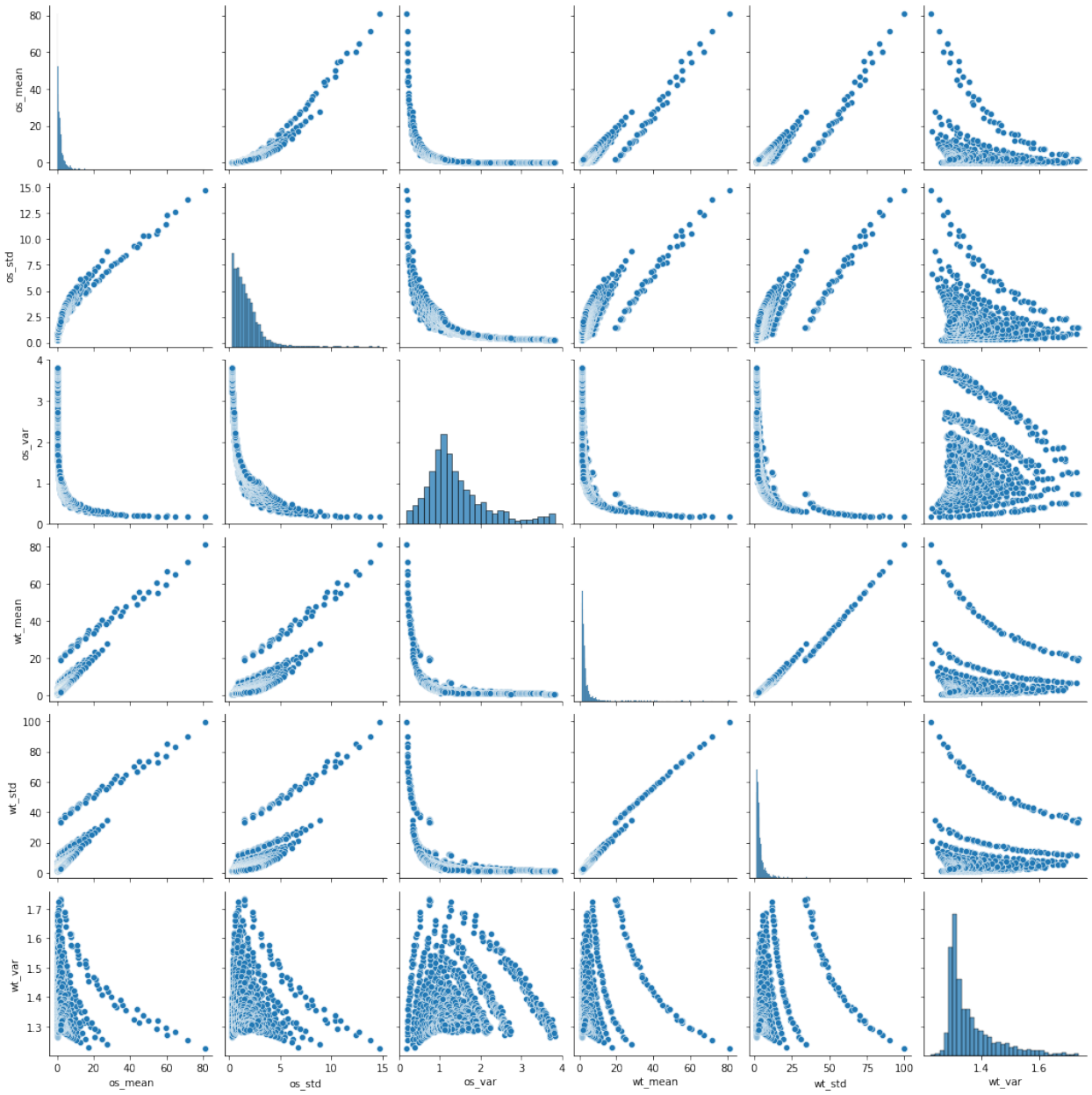

Рисунок 17 — График рассеяния

Для всех метрик, касающихся размера орбиты установлен префикс os_, для метрик времени ожидания заявкой обслуживания — wt_. Как видно из графика рассеяния (например пара os_var — wt_mean) зависимость между характеристиками есть. Это также можно видеть на паре os_mean — wt_std или os_mean – wt_mean, где очевиден линейный ее характер. Однако также на графике видны некие разрывы в наблюдений. Причиной тому может быть способ задания параметров моделей, при котором не были учтены некоторые интервалы значений, в результате не попавшие в наблюдения. Чтобы более ясно увидеть зависимость между двумя исследуемые характеристиками проведем еще два численных эксперимента: в первом случае будем варьировать только интенсивность

обращения с орбиты, во втором — интенсивность входящего потока. При таком подборе параметров получаемые наблюдения будут более гладкими и пригодны к анализу.

Также в рамках проводимых экспериментов воспользуемся несколько иным методом проведения параллельных вычислений. Он заключается в выделении кода, отвечающего за непосредственный запуск имитационной модели, в отдельный скрипт, что позволить его запускать при помощи интерпретатора Python в качестве отдельного процесса. Данный подход имеет два основных преимущества: во–первых, Python снабжен механизмом под названием Global Interpeter Lock или GIL [54], который предотвращает использование интерпретатора за раз больше чем в одном процессе. При использовании вложенных процессов же будет запущен отдельный интерпретатор на каждый процесс. Данное ограничение может являться проблемой при параллельном запуске, так как фактически программа будет работать в однопоточном режиме. Во–вторых, при данном подходе появляется возможность удобно контролировать объем потребляемых программой вычислительных ресурсов (оперативная память и время процессора). Это является важным аспектом при проведении моделирования, например, на домашнем компьютере, не обладающим достаточным количеством ресурсов для запуска большого количество процессов за раз.

Реализация данного подхода представлена ниже:

In [24]:

```
import subprocess
import json
import numpy as np
res = []
procs = []
input_intensity_range = np.arange(0.1, 1.0001, 0.1)
orbit_intensity_range = np.arange(0.05, 3.0001, 0.1)
node_intensity_range = np.arange(1.6001, 2.0001, 0.1)

completed_count = 0
m_index = 0
for inp in list(input_intensity_range):
   for orb in list(orbit_intensity_range):
      for nod in list(node_intensity_range):
         procs.append(subprocess.Popen(' '.join(['python','run.py',str(inp),
             str(orb),str(nod)]),shell=True))
         m_index +=1
         if m_index>100:
            procs[0].wait()
            m_index-=1
            procs.pop(0)
            completed_count+=1
            print(f'Completed: {completed_count}',end="\r",flush=True)


for p in procs:
   p.wait()
   completed_count+=1
   print(f'Completed: {completed_count}',end="\r",flush=True)
```

В ячейке [24] в тройном вложенном цикле в качестве вложенного процесса запускается скрипт run.py для каждого набора параметров. В нем содержится вся логика моделирования, а также код для сохранения результатов в требуемый формат (в данном случае используется JSON). Переменная m_index позволяет нам учитывать количество запущенных процессов. При каждом запуске мы инкрементируем ее, при каждом завершении процесса - декрементируем. А для ограничения количества запущенных процессов мы можем сравнивать значение m_index на каждой итерации с заданным максимальным числом запущенных процессов (в данном случае 100). В случае, если число запущенных процессов достигло порога, мы вызываем метод wait у объекта процесса, тем самым дожидаясь пока один из процессов завершится, чтобы мы могли запустить новый. Таким образом, всегда будет запущено лишь максимально

допустимое число процессов. Определение нужного значения порога определяется эмпирически, например посредством запуска одной модели и замером потребляемой памяти при помощи встроенного пакета psutil [55].

In [25]:
```
from os import listdir
from os.path import isfile, join
import pandas as pd

res = []

for f in [f for f in listdir('results/') if isfile(join('results/', f))]:
   with open(f'results/{f}','r',encoding='utf-8') as ff:
   res.append(json.load(ff))

df = pd.DataFrame(res)
```

По окончании моделирования мы можем прочитать все файлы в директории с результатами (ячейка [25]) и на их основе, например, создать объект DataFrame, позволяющий работать с данными в табличном виде.

Описанным методом были получены результаты для двух дополнительных экспериментов:

- σ варьируется в интервале $[0.05, 10]$ с шагом 0.05. λ и μ зафиксированы на значениях 1 и 1.5 соответственно. Проведено 100 запусков;
- λ варьируется в интервале $[0.1, 1.7]$ с шагом 0.01. σ и μ зафиксированы на значениях 1 и 1.8 соответственно. Проведено 160 запусков.

Для иллюстрации влияния параллельного моделирования на производительность были проведены 100 запусков для первого набора параметров с использованием алгоритма, изложенного в ячейке [24]: в первом случае, когда модели запускались последовательно, общее время моделирования составило 16 минут 4 секунды, во втором, когда было запущено до 10 моделей параллельно — 4 минуты 29 секунд.

В итоге были получены более пригодные к анализу выборки, на основе которых построены графики рассеяния 18 и 19:

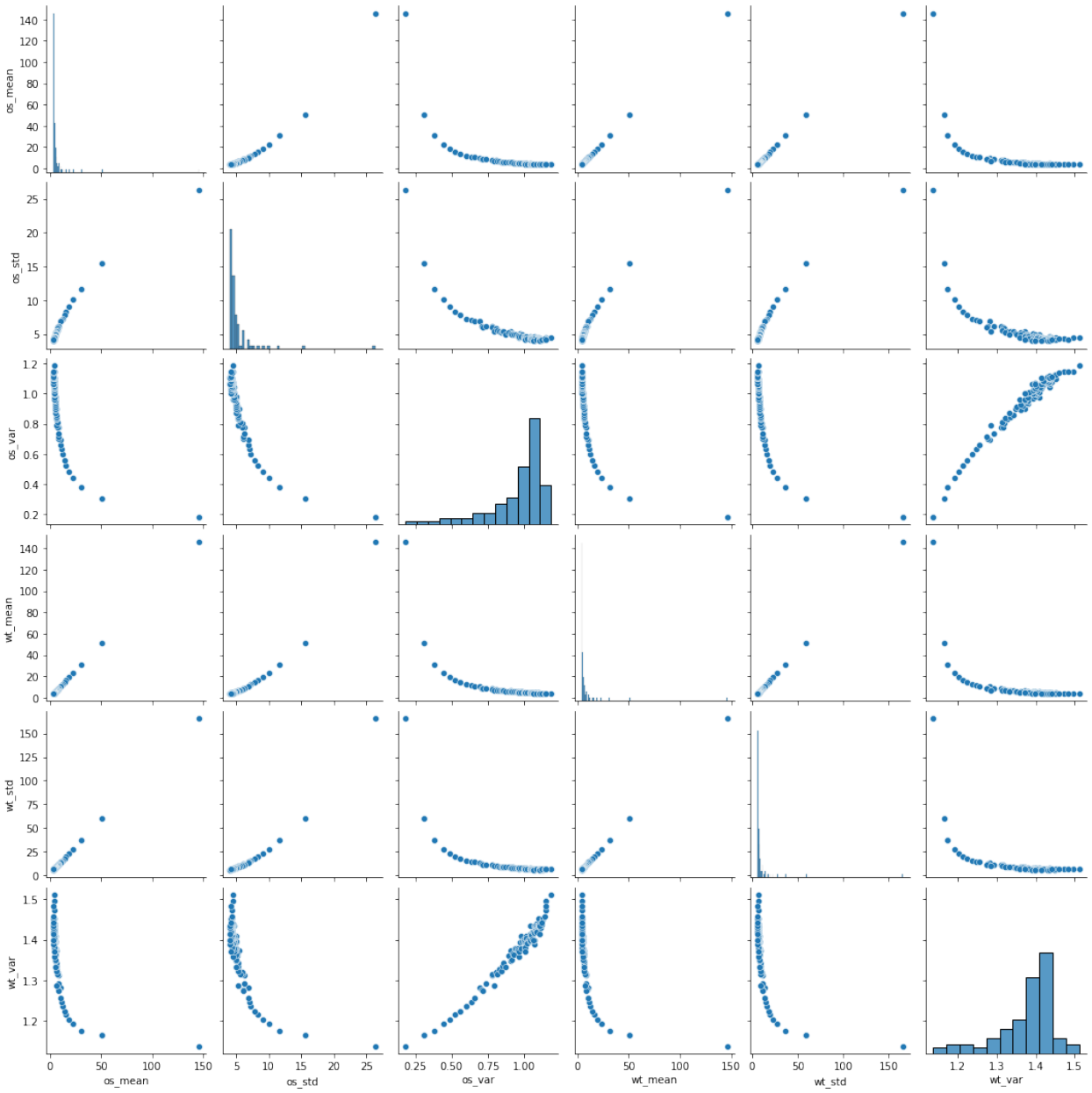

Рисунок 18 — График рассеяния при фиксированном σ

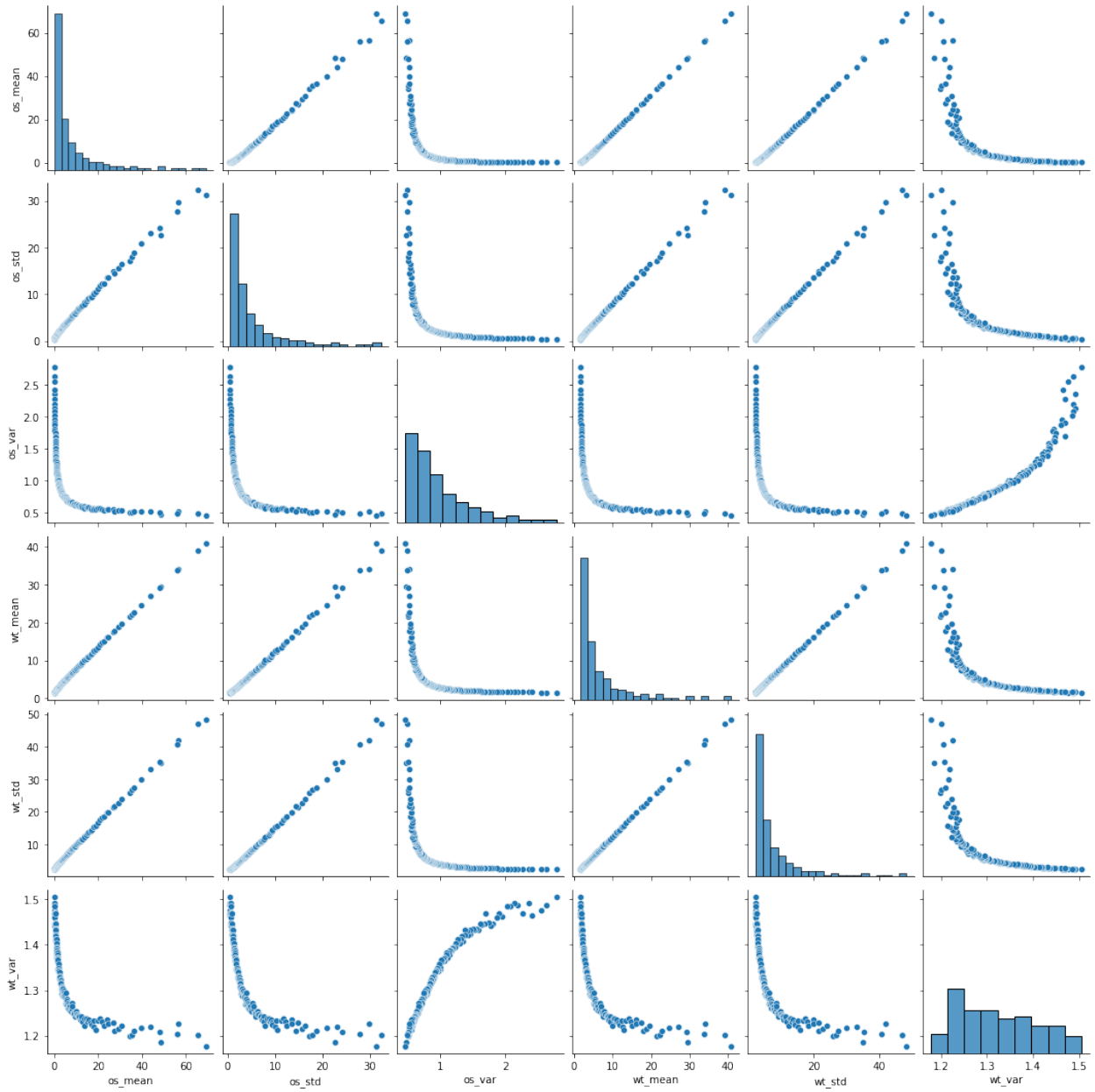

Рисунок 19 — График рассеяния при фиксированном λ

Как видно из полученных наблюдений, исследуемые характеристики явно коррелируют; это можно заключить основываясь на паре os_mean — wt_mean и os_std — wt_std, где наблюдается линейная зависимость между величинами.

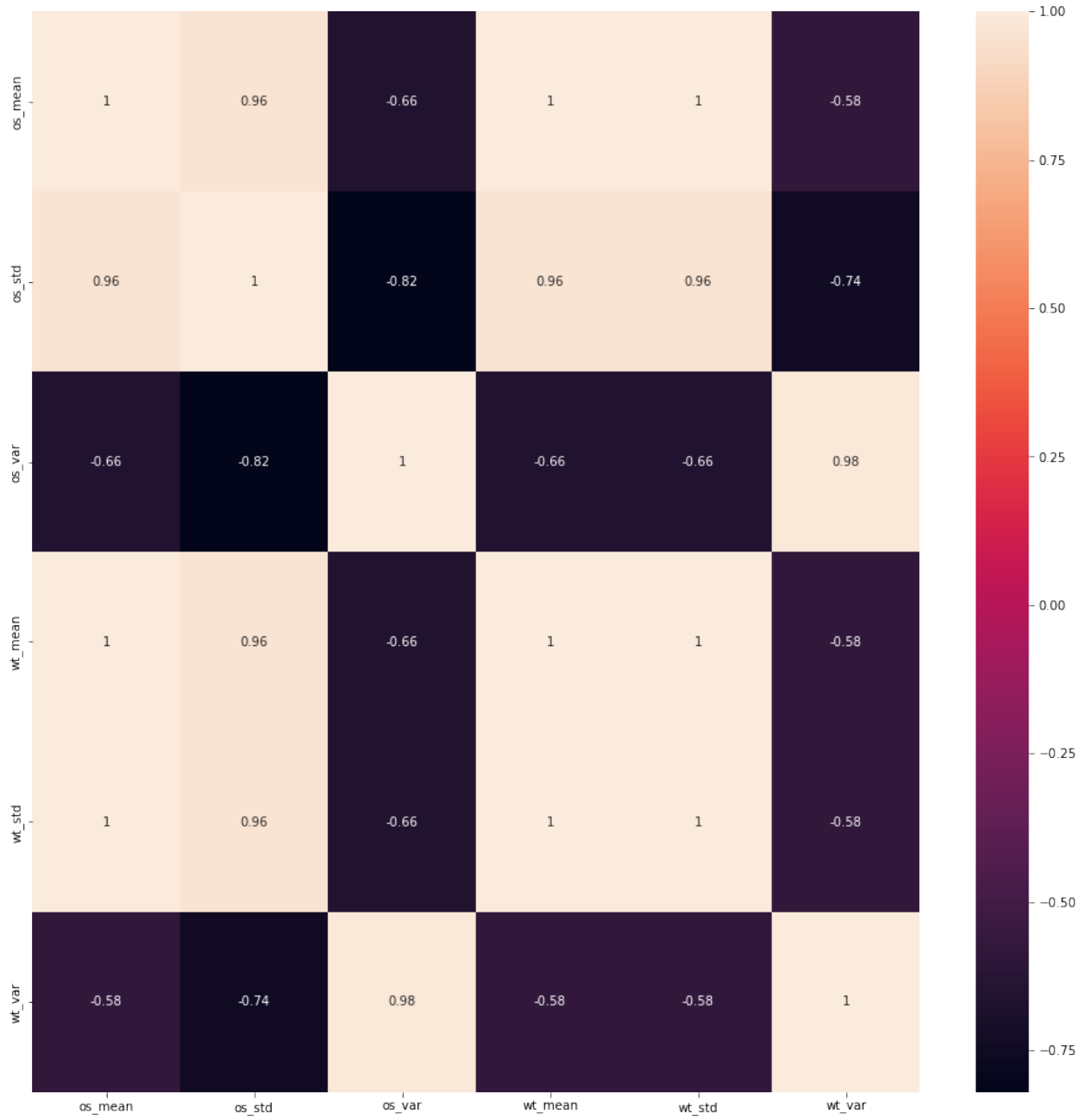

Рисунок 20 — Корреляционная карта числа метрик числа заявок на орбите и времени их ожидания до обслуживания

Корреляционная карта 20 подтверждает произведенные ранее наблюдения на графике рассеяния — между вариацией числа заявок на орбите (os_var) и математическим ожиданием времени их ожидания наблюдается отрицательная зависимость — -0.66. Аналогичные значения вычислены и для других метрик методом Пирсона.

На данном этапе, имея полученные данные о запусках 1760-и моделей с различными параметрами, можно заключить, что между распределением числа заявок на орбите и временем ожидания заявки до обслуживания имеется кор-

реляционная связь.

4.3 Апробация машинного обучения

В данной главе описывается апробирование подхода к анализу систем массового обслуживания при помощи машинного обучения, а именно — предсказывание коэффициента вариации длин интервалов между моментами покидания заявок системы. Данный метод широко используется во многих других областях науки и является универсальным: подходит для задач регрессии, временных рядов, а также классификации и кластеризации [56, 57, 58]. В теории массового обслуживания также есть примеры исследования, в которых успешно используется машинное обучение [59, 60, 61].

В первую очередь, в данном разделе показано, что разработанный программный пакет позволяет проводить крупномасштабные численные эксперименты, результаты которых достаточны, чтобы создать обучающую выборку и на ее основе обучить ряд моделей машинного обучения, способных предсказывать численные характеристики работы модели системы. Благодаря этому для ряда решаемых в теории массового обслуживания задач становятся доступны новые методы исследования.

4.3.1 Разведочный анализ

В данном случае машинное обучение будет использоваться для решения задачи регрессии. Для этой цели, при помощи ранее описанных инструментов, было проведено около 286 тысяч запусков имитационной модели при различных параметрах входящего потока, орбиты и обслуживающего прибора для создания выборки, на основе которой и будет решена задача.

Исходная выборка имеет 9 признаков:

1. elapsed — время работы имитационной модели в секундах. Данный признак был добавлен в утилитарных целях, в частности, для последующей оптимизации работы модели при нестандартных параметрах. Действительное число.

2. mean_input — среднее число обслуженных заявок входящего потока за интервал моделирования. Действительное число.

3. var — коэффициент вариации длин интервалов между моментами наступления событий во входящем потоке. Действительное число.

4. alpha — интенсивность вызова заявок прибором (α). Действительное число.

5. lg — интенсивность входящего потока. Действительное число.

6. mu1 — интенсивность обслуживания входящих заявок (μ_1). Действительное число.

7. mu2 — интенсивность обслуживания вызванных заявок (μ_2). Действительное число.

8. sigma — интенсивность обращений заявок с орбиты (σ). Действительное число.

9. var_input — коэффициент вариации выходящего процесса. Действительное число. Является целевой переменной для предсказаний.

Все признаки имеют 270420 (часть экспериментов выбыли из-за превышения временного лимита на выполнение) ненулевых значений и распределены следующим образом

Таблица 2 — Распределение признаков

	elapsed	mean_input	var_input	alpha	lg	mu1	mu2	sigma	var
mean	7.36	5.52	1.11	1.26	1.90	2.97	2.71	1.41	1.32
std	23.16	5.43	0.20	0.91	0.68	0.66	0.81	0.86	0.31
min	0.01	0.00	0.71	0.00	1.00	1.20	1.00	0.01	1.00
10%	0.79	0.57	0.93	0.20	1.00	2.00	1.40	0.21	1.05
20%	1.29	1.15	1.00	0.40	1.20	2.40	2.00	0.61	1.09
30%	1.74	1.82	1.01	0.60	1.40	2.60	2.20	0.81	1.14
40%	2.19	2.66	1.03	0.80	1.60	2.80	2.60	1.21	1.18
50%	2.70	3.70	1.06	1.20	1.80	3.00	2.80	1.41	1.24
60%	3.34	5.03	1.09	1.40	2.00	3.20	3.00	1.81	1.30
70%	4.32	6.77	1.14	1.80	2.20	3.40	3.20	2.01	1.37
80%	6.21	9.23	1.20	2.00	2.60	3.60	3.60	2.41	1.48
90%	11.76	13.25	1.32	2.60	2.80	3.80	3.80	2.61	1.67
95%	22.72	16.87	1.46	3.00	3.20	3.80	3.80	2.81	1.87
99%	104.70	24.02	1.86	3.40	3.40	3.80	3.80	2.81	2.45
max	419.53	35.89	5.99	3.60	3.60	3.80	3.80	2.81	8.19

На основе сводки из таблицы 2 можно заметить, что в среднем время моделирования занимало 7.36 секунд, половина экспериментов была запущена с интенсивностью входящего потока 1.8, а максимальный коэффициент вариации выходящего процесса составил 8.19.

На данном этапе мы можем избавиться от признаков, которые не будут участвовать в обучении модели: mean_input и elapsed. Для оставшихся признаков построим графики рассеяния [62]. Данные диаграммы полезны для отображения многомерных данных, как в данном случае. С их помощью можно определить потенциальные взаимосвязи между количественными переменными.

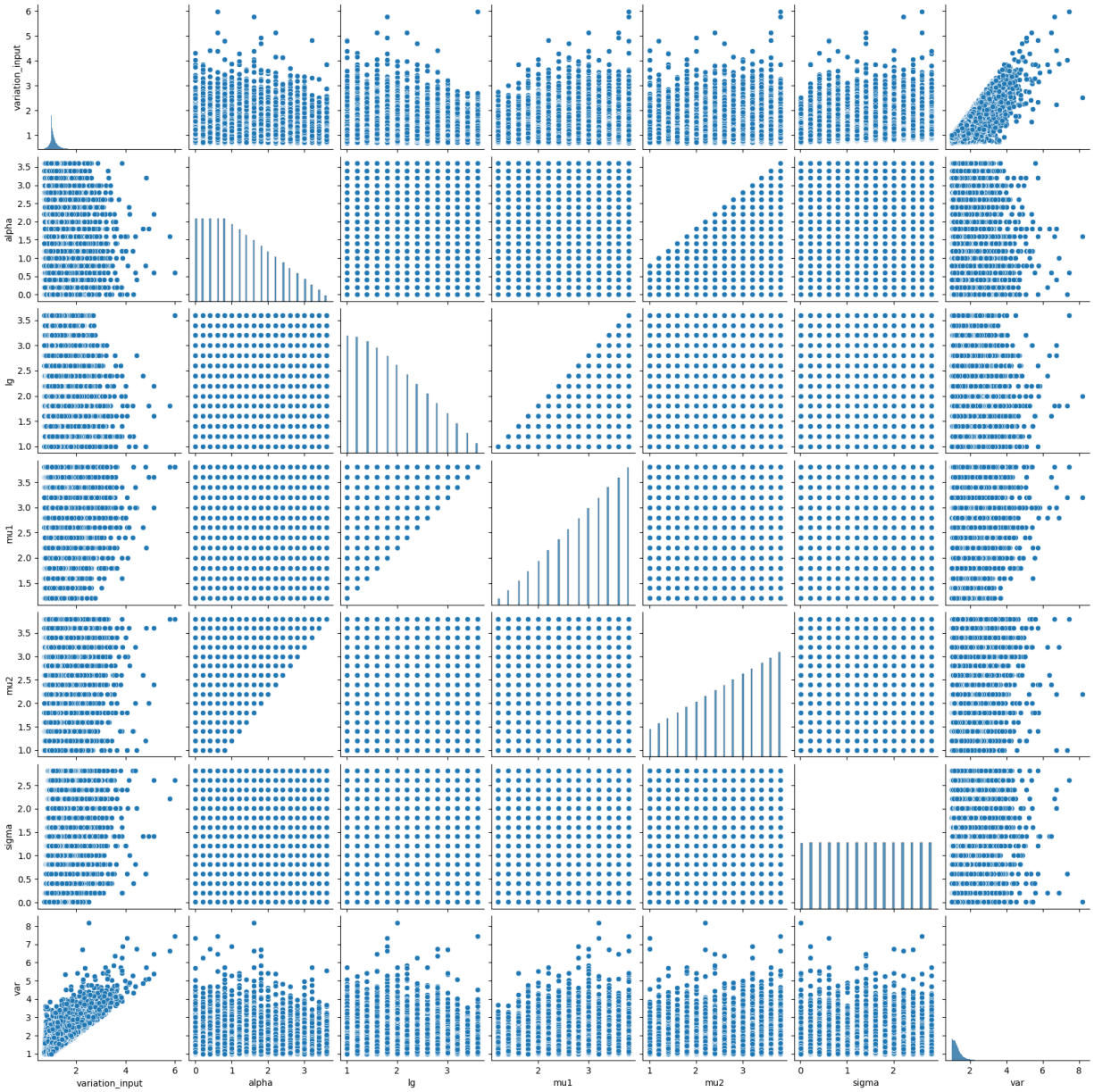

Рисунок 21 — Графики рассеяния признаков

На графике рассеяния 21 видно, что явную связь имеют коэффициент вариации входящего потока (var) и коэффициент вариации выходящего процесса (var_input). Для более точного определения зависимостей построим для признаков тепловую карту коэффициента корреляции Пирсона

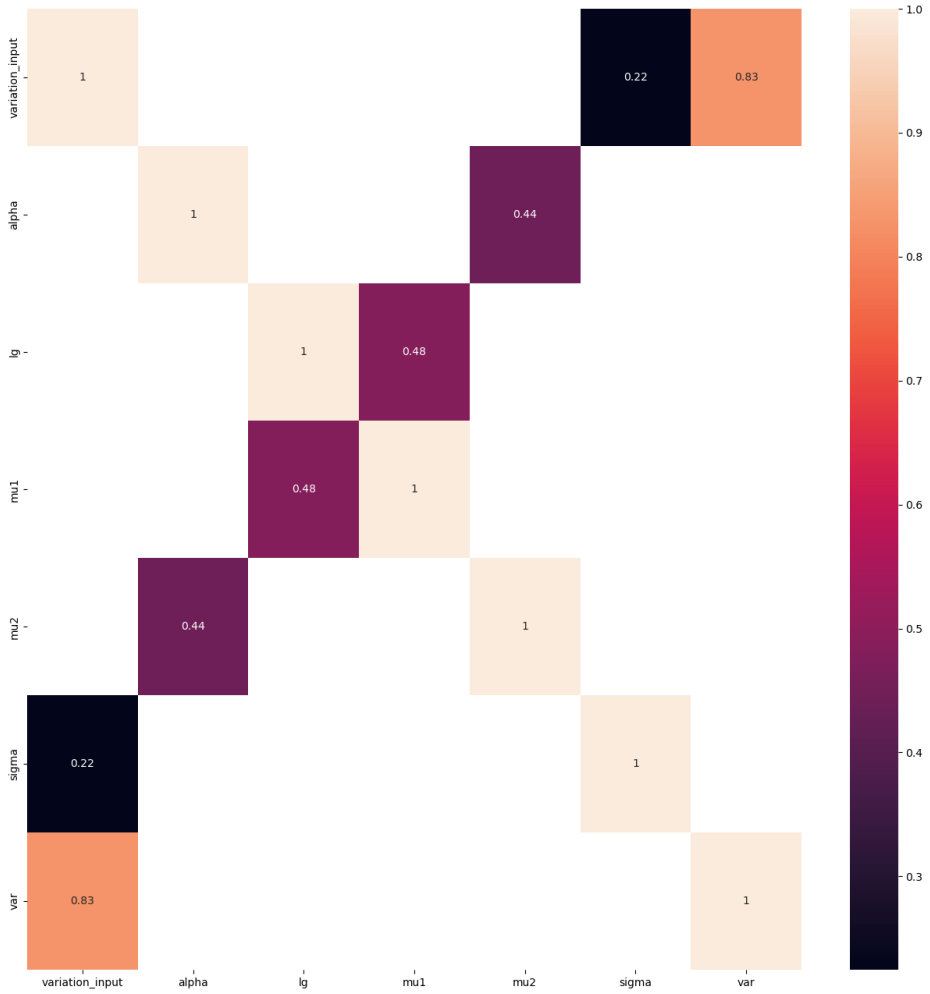

Рисунок 22 — Тепловая карта корреляции признаков

На рисунке 22 можно наблюдать логичную корреляцию таких признаков, как интенсивность обслуживания входящих заявок (mu1) и интенсивность входящего потока (lg), а также интенсивность обслуживания вызванных заявок (mu2) и интенсивность их вызова (alpha). Для целевой переменной же можно наблюдать ранее выявленную корреляцию с коэффициентом вариации входящего потока и интенсивностью обращения заявок с орбиты (sigma), что так же

ожидаемо, поскольку задержка заявок на орбите влияет на пропускную способность прибора.

4.3.2 Построение предсказательных моделей

Для настройки и обучения предсказательных моделей используется язык Python и библиотека для ансамблевого машинного обучения scikit-learn [40]. Ансамблевые методы машинного обучения являются мощным и в то же время простым для понимая инструментом анализа данных. Он представляет собой метод обучения, где несколько моделей обучаются решению одной и той же задачи и объединяются для получения наиболее точных результатов. Ключевое преимущество такого метода заключается в том, что результат работы нескольких моделей будет иметь большую точность, чем результат только одной модели. Будут обучены и протестированы следующие модели:

- LinearRegression — реализация метода наименьших квадратов. Будет использована для проверки правильности выборки и сравнения с другими, более сложными методами.
- RandomForest [63] — алгоритм, основывающийся на ансамбле решающих деревьев с разным ветвлением (зависит от порядка признаков), каждое из которых дает свой ответ на поставленную задачу, сам по себе являющийся неточным, однако при наличии большого количества деревьев, результат имеет хорошую точность.
- GradientBoost [64] — обучение слабых моделей последовательно, таким образом, что каждая последующая исправляет ошибки предыдущих. Как правило, данный алгоритм превосходит в точности лес случайных деревьев.
- CatBoost [65] — оригинальная реализация алгоритма градиентного бустинга компанией Яндекс. Самой компанией используется для улучшения поисковых результатов, ранжирования рекомендаций пользователям и онлайн-аналитики. В общем случае является более точной, чем градиентный бустинг.

Первым шагом в обучении изложенных алгоритмов будет разбиение выборки на тестовую и обучающую, также выделение целевой переменной в от-

дельный вектор y. Обучающая выборка составила 70% от общей. Далее листинги кода будут предложены в формате Python Notebook

In [26]:
```
from sklearn.model_selection import train_test_split
x = df.drop(['var_input'], axis = 1)
y = df['var_input']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
    random_state=9)

x_train = x_train.reset_index(drop = True)
y_train = y_train.reset_index(drop = True)
x_test = x_test.reset_index(drop = True)
y_test = y_test.reset_index(drop = True)
```

In [27]:
```
x_train.shape, y_train.shape
```

```
((189168, 6), (189168,))
```

In [28]:
```
x_test.shape, y_test.shape
```

```
^^I((81072, 6), (81072,))
```

Так, размер обучающей выборки составил 189168 строк (ячейка [27]), размер тестовой выборки — 81072 строк (ячейка [28]).

Поскольку общая выборка не имеет пропущенных значений, этап заполнения пропусков был пропущен.

Поскольку большинство алгоритмов ожидают на вход нормализованных данных, то для корректного их обучения выполним Z–преобразование (ячейка [29]) при помощи объекта scikit-learn StandardScaler. Идея преобразования заключается в нормализации распределения каждого признака таким образом, что математическое ожидание будет равно нулю, а дисперсия — единице. Нормализация производится для каждого признака индивидуально.

In [29]:
```
from sklearn.preprocessing import StandardScaler
scal = StandardScaler()
scal.fit(x_train)
x_train_z = pd.DataFrame(scal.transform(x_train), columns = x_train.columns)
x_test_z = pd.DataFrame(scal.transform(x_test), columns = x_test.columns)
```

После преобразования необходимо выделить каждой из моделей значимые признаки. Это выполняется посредством предварительного обучения и ана-

лиза коэффициентов значимости [40]

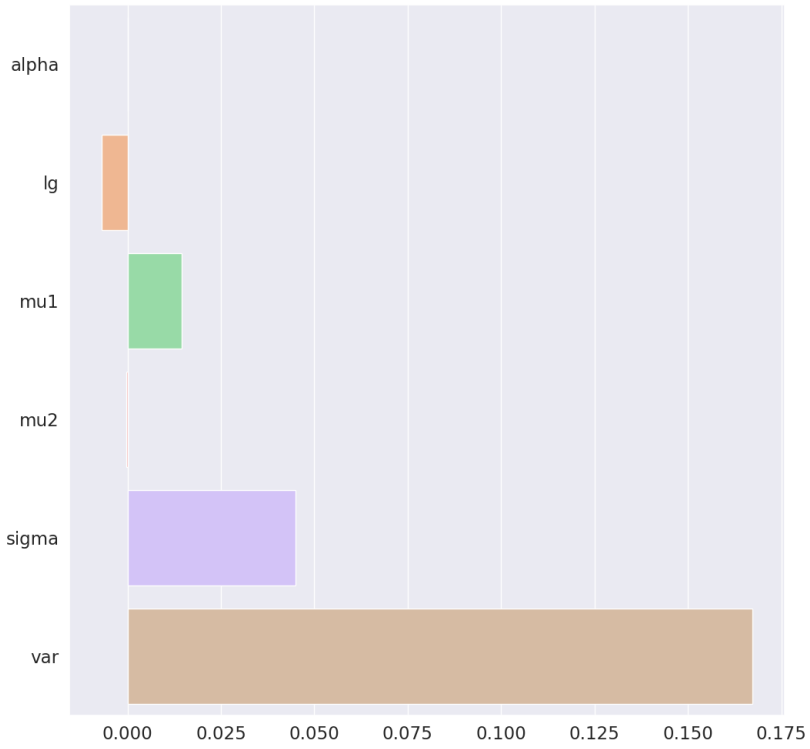

Рисунок 23 — Значимость признаков для LinearRegression

На рисунке 23 видно, что для алгоритма LinearRegression значимыми признаками являются задержка заявок на орбите и коэффициент вариации входящего потока. В меньшей степени является значимой интенсивность обслуживания входящий заявок, и отрицательно сказывается на результате работы наличие признака lg.

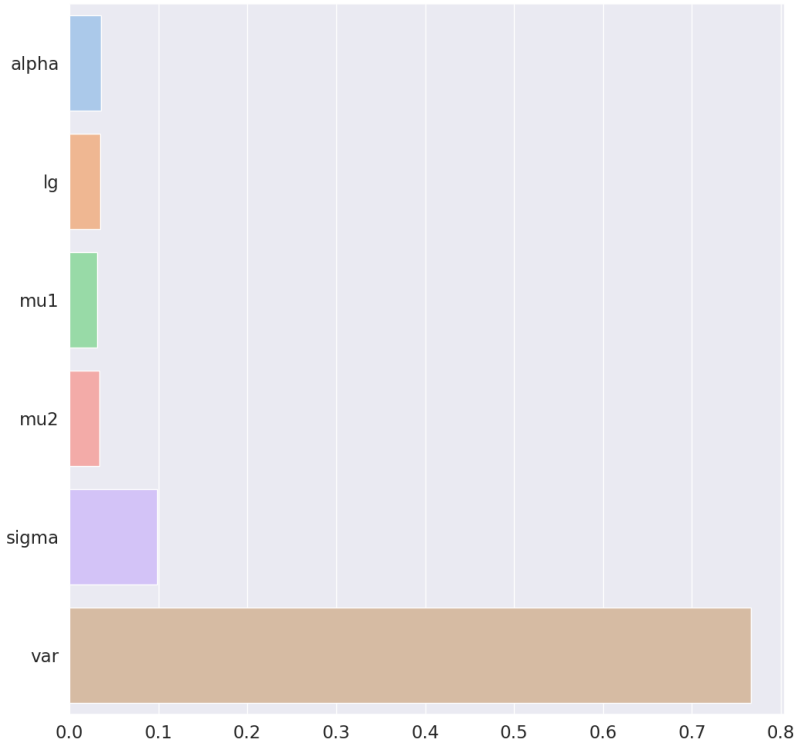

Рисунок 24 — Значимость признаков для RandomForest

На рисунке 24 можно наблюдать, что для алгоритма RandomForest наиболее значимыми оказались также вариация входящего потока и задержка заявок на орбите.

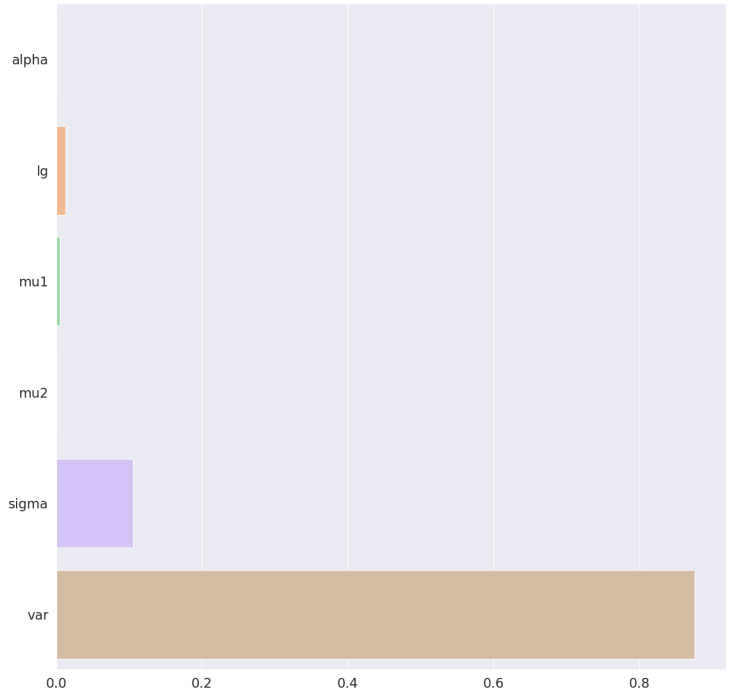

Рисунок 25 — Значимость признаков для GradientBoost

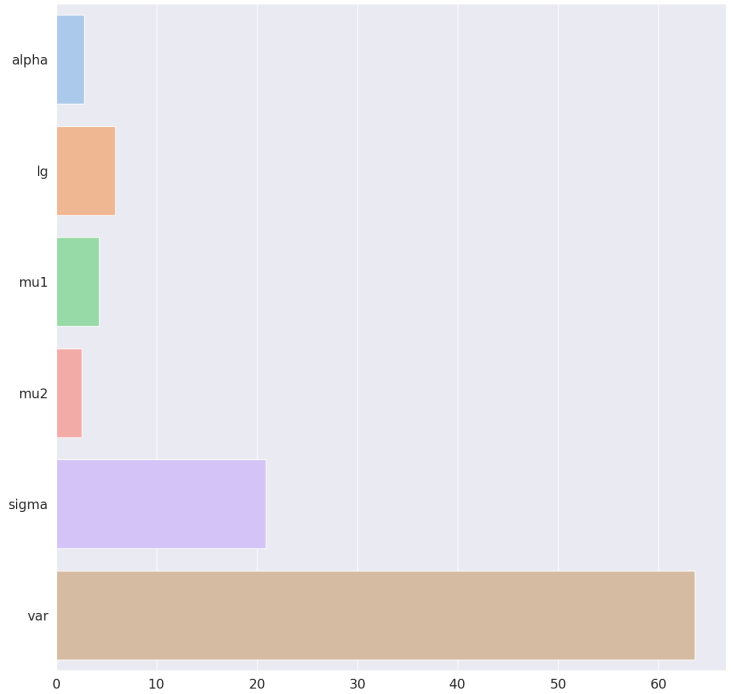

Рисунок 26 — Значимость признаков для CatBoost

Для алгоритмов GradientBoost и CatBoost значимыми признаками также выступают sigma и var. Основываясь на полученных данных, оставим в обучающей выборке для каждой модели только значимые признаки.

Задание гиперпараметров алгоритмов [66] будем производить случайным подбором в 5 итераций при помощи алгоритма scikit-learn RandomizedSearchCV. Важно заметить, что из тестовой выборки не выделялась валидационная, так как RandomizedSearchCV содержит встроенную кросс-валидацию. Он позволяет, начиная с указанных начальных значений, итеративно подобрать оптимальные гиперпараметры для модели. Для метода наименьших квадратов эта процедура не производится, поскольку метод не допускает задания каких-либо параметров.

In [30]:
```
rf = RandomForestRegressor(n_jobs=-1)
parameters_rf = {'n_estimators': range (100,150, 10),
    'criterion': ['squared_error'],
    'max_depth' : range(1,9,1),
    'min_samples_split': range(2,30,5),
    'min_samples_leaf':range(1,30,5)
}
search_rf = RandomizedSearchCV(rf, parameters_rf,n_jobs=-1, n_iter = 5)
search_rf.fit(x_train_rf, y_train)
search_rf.best_params_, search_rf.best_score_
```

```
({'n_estimators': 110,
    'min_samples_split': 2,
    'min_samples_leaf': 26,
    'max_depth': 7,
    'criterion': 'squared_error'},
0.7699992121816944)
```

In [31]:
```
best_model_rf = search_rf.best_estimator_
best_model_rf.fit(x_train_rf, y_train)
y_pred_test_rf = best_model_rf.predict(x_test_rf)

print('MAE:', mean_absolute_error(y_test, y_pred_test_rf))
print('RMSE:', mean_squared_error(y_test, y_pred_test_rf, squared=False))
print('R2_score:', r2_score(y_test, y_pred_test_rf))
```

```
MAE: 0.06538199710376982
RMSE: 0.09781859274247506
R2_score: 0.7695224558769049
```

В ячейке [30] производится задание начальных значений гиперпарамет-

ров для алгоритма случайных деревьев. Среди параметров:

- n_estimators — количество деревьев решений.
- criterion — методы оценки ошибки в процессе обучения. По умолчанию выбрано среднеквадратическое отклонение.
- max_depth — максимальная глубина дерева.
- min_samples_split — минимальное количество сэмплов, требуемое для расщепления вершины дерева.
- min_samples_leaf — минимальное количество сэмплов, требуемое для превращения вершины в лист.

Далее производится подбор параметров алгоритма, дающих максимальную точность, измеряемую коэффициентом детерминации. Коэффициент детерминации является основной метрикой качества для решений задач регрессии [67]. В данном случае, он составил 0.769. В ячейке `[31]` производится обучение модели с лучшими гиперпараметрами и оценка точности, которая проводится при помощи следующих метрик: средняя абсолютная ошибка (MAE), среднеквадратическое отклонение (RMSE) и коэффициент детерминации (R2_score).

Для остальных моделей использовалась та же последовательность операций. В итоге, были получены следующие результаты

Рисунок 27 — Оценка точности моделей

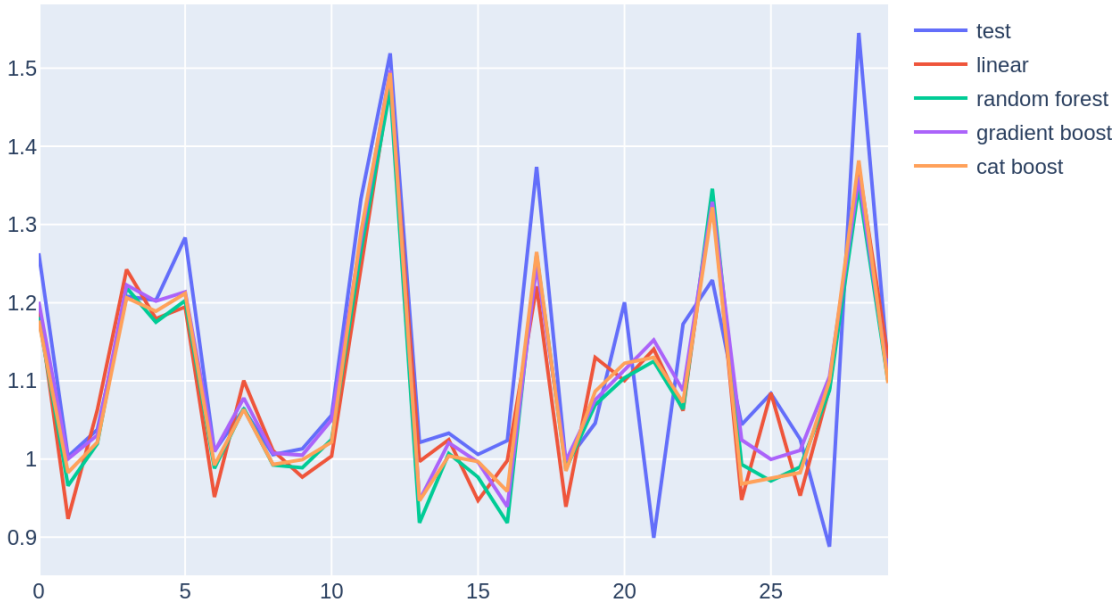

Рисунок 28 — Сравнение предсказаний моделей

Как видно на рисунке 27, CatBoost оказался самым точным среди выбранных алгоритмов — результаты его предсказаний имеют наименьшую среднюю абсолютную и квадратические ошибки и наибольшее значение коэффициента детерминации, равное 0.774. Рассмотрение результатов предсказаний в целом показывает, что общая точность моделей находится в районе 80%. Для апробирования подхода данный результат является в достаточной степени приемлемым. Также, можно предложить, почему точность моделей не оказалась выше: на рисунке 28 представлен отрезок тестового вектора значения коэффициента вариации выходящего процесса с наложением на него предсказаний. Можно заметить, что для значений, которые наиболее приближены к среднему по выборке, предсказания точны, однако для менее частых модели не смогли дать верный ответ. Это обусловлено слабой дисперсией в распределении значимых признаков в выборке (таблица 2), что приводит к переобучению моделей на определенном диапазоне значений признаков, делая предсказания для остальных случаев искаженными. Для улучшения качества предсказаний требуется изменить метод генерации параметров таким образом, чтобы признаки были

распределены более равномерно и охватывали как можно больше возможных значений.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был спроектирован и разработан программный комплекс с набором инструментов для анализа систем теории массового обслуживания, содержащий имитационную модель, алгоритмы вычисления характеристик и вспомогательные утилиты. Согласно указанной цели работы был выполнен ряд задач.

Имитационное моделирование было рассмотрено как численный метод исследования систем массового обслуживания и какие задачи оно позволяет решать. В частности, было отмечено, что численные методы позволяют находить взаимосвязь между наблюдаемые характеристиками работы системы и теми, которые еще не были получены или не могут быть получены аналитическим путем. Также был рассмотрен алгоритм моделирования, основывающийся дискретно–событийном подходе, где в процессе работы модели содержится очередь событий, которые последовательно наступают и меняют состояние системы.

В процессе разработки программного комплекса была определена его архитектура, построена объектная модель предметной области (рисунок 6), описывающая структуру и взаимодействие ее сущностей, среди которых элемент модели (рисунок 3), маршрутизатор (рисунок 5), генератор задержки, заявка и сборщик статистики. Введена система слотов (рисунок 4), которая в совокупности с остальными сущностями позволяет создавать широкий набор различных систем массового обслуживания для моделирования и анализа. Ключевой особенностью выбранной архитектуры является общий интерфейс для всех элементов модели, генераторов задержки и сборщиков статистики, что позволяет легко расширять набор реализаций каждого интерфейса без изменений уже имеющихся реализаций.

Для обеспечения производительности имитационной модели, в качестве языка программирования для реализации был выбран C++, что позволило оптимизировать работу программы для проведения масштабных параллельных вычислений. Для обеспечения интеграции имитационного моделирования в процесс анализа, результат разработки является расширительным пакетом для Python, что позволило совместить эффективность системного языка программирования и простоту скриптового языка. Таким образом, мы получили возможность

конфигурировать и запускать множество моделей параллельно при помощи простого программного интерфейса, а дальнейшая работа с результатами моделирования может происходить непосредственно в среде Python при помощи сторонних инструментов, что значительно сокращает трудоемкость проведения численного анализа и время, требуемое для исследования. Помимо этого, инструмент содержит встроенную документацию или пояснение по работе с каждым классом или функцией, а установка возможна при помощи пакетного менеджера pip, что значительно упрощает начало работы с пакетом.

Помимо этого, в рамках программного комплекса были реализованы алгоритмы вычисления характеристик рассматриваемой модели системы массового обслуживания, включающие также вспомогательные функции для вычисления точности распределения вероятностей при помощи расстояния Колмогорова. Особенностью данного инструмента является наличие функций для подсчета распределения вероятностей при помощи дискретного преобразования Фурье. Их реализация позволяет существенно оптимизировать процесс проведения экспериментов.

На ряде примеров был описан процесс работы с программным пакетом, в частности было показано получение распределения вероятностей числа обслуженных заявок для RQ–системы с повторными вызовами и обратной связью (рисунок 12) и его характеристик, а также проведение сравнения с результатов с асимптотическими. Также на примере исследования зависимости числа обслуженных заявок и времени ожидания заявки до получения обслуживания для той же модели системы был описан подход к проведению параллельных запусков моделирования с различной конфигурацией и последующий анализ полученных данных. Было описано, как можно генерировать необходимое количество наборов параметров, структурировать получаемую информацию и проводить ее анализ.

В конечном итоге, разработанные инструменты позволили провести апробацию методов машинного обучения для расчета одной из характеристик работы системы — вариации длин интервалов между моментами покидания заявками прибора. Было проиллюстрировано, что оптимизация процесса проведения численных экспериментов позволила использовать новые методы исследования систем массового обслуживания — на основе выборки, составленной из

286 тысяч запусков имитационной модели с различными параметрами, были обучены алгоритмы LinearRegression, RandomForest, GradientBoost и CatBoost для решения задачи регрессии. Были выявлены зависимости целевой переменной от ряда признаков, в частности, задержки заявок на орбите и коэффициента вариации входящего потока. Обученные модели позволяют получать результаты с достоверностью около 80%. В дальнейшем исследовании стоит цель увеличить точность предсказаний за счет изменения процедуры генерации наборов параметров таким образом, чтобы распределение каждого признака было более равномерным и охватывало как можно большее количество возможных значений во избежание искажения результатов.

Часть результатов проведенного исследования были представлены в качестве докладов на двух международных конференциях:

- XXIV International Conference on Distributed Computer and Communication Networks (DCCN) (сентябрь 2021, Томск) [23];
- XX Международная конференции по информационным технологиям и математическому моделированию имени А.Ф. Терпугова (ITMM) (декабрь 2021, Томск);

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Назаров А. А., Терпугов А. Ф. Теория массового обслуживания:[учебное пособие по специальностям 010200 (010501) «Прикладная математика и информатика» и 061800 (080116) «Математические методы в экономике». — Томск: Изд-во НТЛ, 2010.

[2] Erlang A. K. The theory of probabilities and telephone conversations // Nyt. Tidsskr. Mat. Ser. B. — 1909. — Vol. 20. — P. 33–39.

[3] Phung-Duc T. Retrial queueing models: A survey on theory and applications // arXiv preprint arXiv:1906.09560. — 2019.

[4] Artalejo Jesus R. Accessible bibliography on retrial queues: progress in 2000–2009 // Mathematical and computer modelling. — 2010. — Vol. 51, no. 9-10. — P. 1071–1081.

[5] Bellovin S. M., Leech M., Taylor T. ICMP traceback messages. — 2003.

[6] Bjornstad J. H. Traffic characteristics and queueing theory: implications and applications to web server systems : Master's thesis / J. H. Bjornstad. — 2006.

[7] Kritzinger P. A performance model of the OSI communication architecture // IEEE transactions on communications. — 1986. — Vol. 34, no. 6. — P. 554–563.

[8] Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы. — Питер, 2010.

[9] Baiocchi A., Blefari-Melazzi N. Steady-state analysis of the MMPP/G/1/K queue // IEEE transactions on communications. — 1993. — Vol. 41, no. 4. — P. 531–534.

[10] Lapatin I. L., Nazarov A. A. Asymptotic Analysis of the Output Process in Retrial Queue with Markov-Modulated Poisson Input Under Low Rate of Retrials Condition // International Conference on Distributed Computer and Communication Networks / Springer. — 2019. — P. 315–324.

[11] Задорожный Владимир Николаевич. Методы аналитико-имитационного моделирования систем с очередями и стохастических сетей : Ph. D. thesis / Владимир Николаевич Задорожный ; СПб., 2011. — 2011.

[12] Горбунов АР, Лычкина НН. Парадигмы имитационного моделирования: новое в решении задач стратегического управления (объединенная логика имитационного моделирования) // Бизнес-информатика. — 2007. — no. 2. — P. 60–66.

[13] Илюхина НА, Комаревцева ОО. Дискретно-событийное моделирование в управлении экономической системы муниципального образования // Современные наукоемкие технологии. — 2015. — no. 7. — P. 77–80.

[14] Григорьева Татьяна Евгеньевна. Дискретно-событийное моделирование в СМ МАРС для курса «Системы массового обслуживания» // Доклады Томского государственного университета систем управления и радиоэлектроники. — 2014. — no. 1 (31). — P. 152–155.

[15] Лебедюк Эдуард Андреевич. Агентное моделирование: состояние и перспективы // Вестник Российского экономического университета им. ГВ Плеханова. — 2017. — no. 6 (96). — P. 155–162.

[16] Glynn Peter W, Iglehart Donald L. Simulation methods for queues: An overview // Queueing systems. — 1988. — Vol. 3, no. 3. — P. 221–255.

[17] Назаров Анатолий Андреевич [et al.]. Асимптотический анализ марковизируемых систем. — 1991.

[18] Исследование RQ-системы M| M| 1 с разнотипными вызываемыми заявками и ненадежным прибором методом асимптотически-диффузионного анализа / Анатолий Андреевич Назаров [et al.] // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. — 2021. — no. 57. — P. 74–83.

[19] Имитационное и статистическое моделирование / Виктор Иванович Лобач [et al.] // Практикум для студентов математических и экономических специальностей. Минск: БГУ. — 2004.

[20] Моисеев Александр Николаевич [et al.]. Исследование математических моделей систем и сетей массового обслуживания с высокоинтенсивными непуассоновскими входящими потоками. — 2016.

[21] Алиев Тауфик Измайлович. Погрешности моделирования высоконагруженных систем в GPSS World // Научно-технический вестник информационных технологий, механики и оптики. — 2013. — no. 1 (83). — P. 70–75.

[22] Fasano Giovanni, Franceschini Alberto. A multidimensional version of the Kolmogorov–Smirnov test // Monthly Notices of the Royal Astronomical Society. — 1987. — Vol. 225, no. 1. — P. 155–170.

[23] Blaginin Alexey, Lapatin Ivan. Approximation of the Two-Dimensional Output Process of a Retrial Queue with MMPP Input // International Conference on Distributed Computer and Communication Networks / Springer. — 2021. — P. 333–345.

[24] The comparison of structured modeling and simulation modeling of queueing systems / Igor Yakimov [et al.] // Information Technologies and Mathematical Modelling. Queueing Theory and Applications: 16th International Conference, ITMM 2017, Named After AF Terpugov, Kazan, Russia, September 29-October 3, 2017, Proceedings 16 / Springer. — 2017. — P. 256–267.

[25] Simulation using building blocks / Edwin Valentin [et al.] // Proceedings conference on AI, Simulation and Planning / Citeseer. — 2002. — P. 65–71.

[26] Claes Peter. Controlling Fluid Simulations with Custom Fields in Houdini Master Thesis. — 2009.

[27] Fowler Martin. Analysis patterns: reusable object models. — Addison-Wesley Professional, 1997.

[28] Pilone Dan, Pitman Neil. UML 2.0 in a Nutshell. — ”O’Reilly Media, Inc. 2005.

[29] Herchi Hatem, Abdessalem Wahiba Ben. From user requirements to UML class diagram // arXiv preprint arXiv:1211.0713. — 2012.

[30] Hallinan Jr Arthur J. A review of the Weibull distribution // Journal of Quality Technology. — 1993. — Vol. 25, no. 2. — P. 85–93.

[31] Schmid Hubert. C++ 11: pure virtual functions with override and final. — 2012.

[32] PyTorch. — https://gcc.gnu.org/. — Accessed: 2023-04-29.

[33] Branco David, Henriques Pedro Rangel. Impact of GCC optimization levels in energy consumption during C/C++ program execution // 2015 IEEE 13th International Scientific Conference on Informatics / IEEE. — 2015. — P. 52–56.

[34] Matsumoto Makoto, Nishimura Takuji. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator // ACM Transactions on Modeling and Computer Simulation (TOMACS). — 1998. — Vol. 8, no. 1. — P. 3–30.

[35] Mersenne Twister Engine. — https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine. — Accessed: 2023-05-02.

[36] The fundamental package for scientific computing with Python. — https://github.com/numpy/numpy. — Accessed: 2023-04-29.

[37] Seamless operability between C++11 and Python. — https://github.com/pybind/pybind11. — Accessed: 2023-04-29.

[38] McKinney Wes. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. — ”O’Reilly Media, Inc. 2012.

[39] McKinney Wes [et al.]. pandas: a foundational Python library for data analysis and statistics // Python for high performance and scientific computing. — 2011. — Vol. 14, no. 9. — P. 1–9.

[40] Hackeling Gavin. Mastering Machine Learning with scikit-learn. — Packt Publishing Ltd, 2017.

[41] SciPy 1.0: fundamental algorithms for scientific computing in Python / Pauli Virtanen [et al.] // Nature methods. — 2020. — Vol. 17, no. 3. — P. 261–272.

[42] PyTorch. — https://pytorch.org/. — Accessed: 2023-04-29.

[43] The PyPA recommended tool for installing Python packages. — https://pypi.org/project/pip/. — Accessed: 2023-05-18.

[44] A built-package format for Python. — https://pypi.org/project/wheel/. — Accessed: 2023-05-18.

[45] Easily download, build, install, upgrade, and uninstall Python packages. — https://pypi.org/project/setuptools/. — Accessed: 2023-05-18.

[46] Ястребов И.П. Дискретизация непрерывных сигналов во времени. Теорема Котельникова // Нижний Новгород: Нижегородский государственный университет им. НИ Лобачевского. — 2012.

[47] Кузнецов Н.А., Козякин В.С. Теорема Котельникова–основа цифрового оценивания и моделирования непрерывных процессов // Радиотехника. — 2008. — □. 8.

[48] Гнеденко Борис Владимирович. Курс теории вероятностей. — URSS, 2010.

[49] Калинина Е.С., Кудрявцева М.В. О применении аппарата производящих функций в теории вероятностей // Современные тенденции развития науки и технологий. — 2016. — ¹ 3-1. — □. 18–22.

[50] Ряды Фурье / В.П. Долгополов [□ □□.]. — 2011.

[51] Тимошенко Л.И. Дискретное преобразование Фурье и его быстрые алгоритмы // Современные наукоемкие технологии. — 2014. — ¹ 12-2. — □. 188–193.

[52] Bronson R. Matrix methods: An introduction. — Gulf Professional Publishing, 1991.

[53] multiprocessing — Process-based parallelism. — https://docs.python.org/3/library/multiprocessing.html. — Accessed: 2023-05-02.

[54] GlobalInterpreterLock. — https://wiki.python.org/moin/GlobalInterpreterLock. — Accessed: 2023-05-14.

[55] Cross-platform lib for process and system monitoring in Python. — https://pypi.org/project/psutil/. — Accessed: 2023-05-14.

[56] Libbrecht Maxwell W, Noble William Stafford. Machine learning applications in genetics and genomics // Nature Reviews Genetics. — 2015. — Vol. 16, no. 6. — P. 321–332.

[57] Shinde Pramila P, Shah Seema. A review of machine learning and deep learning applications // 2018 Fourth international conference on computing communication control and automation (ICCUBEA) / IEEE. — 2018. — P. 1–6.

[58] Soofi Aized Amin, Awan Arshad. Classification techniques in machine learning: applications and issues // Journal of Basic and Applied Sciences. — 2017. — Vol. 13. — P. 459–465.

[59] Learning Deep Generative Models for Queuing Systems / Cesar Ojeda [et al.] // Proceedings of the AAAI Conference on Artificial Intelligence. — Vol. 35. — 2021. — P. 9214–9222.

[60] Scheduling data analytics work with performance guarantees: queuing and machine learning models in synergy / Ji Xue [et al.] // Cluster Computing. — 2016. — Vol. 19, no. 2. — P. 849–864.

[61] Balla Husamelddin AMN, Sheng Chen Guang, Weipeng Jing. Reliability enhancement in cloud computing via optimized job scheduling implementing reinforcement learning algorithm and queuing theory // 2018 1st international conference on data intelligence and security (ICDIS) / IEEE. — 2018. — P. 127–130.

[62] Cox Nicholas. PAIRPLOT: Stata module for plots of paired observations. — 2007.

[63] Rigatti Steven J. Random forest // Journal of Insurance Medicine. — 2017. — Vol. 47, no. 1. — P. 31–39.

[64] Natekin Alexey, Knoll Alois. Gradient boosting machines, a tutorial // Frontiers in neurorobotics. — 2013. — Vol. 7. — P. 21.

[65] Hancock John T, Khoshgoftaar Taghi M. CatBoost for big data: an interdisciplinary review // Journal of big data. — 2020. — Vol. 7, no. 1. — P. 1–45.

[66] Feurer Matthias, Hutter Frank. Hyperparameter optimization // Automated machine learning. — Springer, Cham, 2019. — P. 3–33.

[67] Validation of the Word Accentuation Test (TAP) as a means of estimating premorbid IQ in Spanish speakers / Jesús J Gomar [et al.] // Schizophrenia Research. — 2011. — Vol. 128, no. 1-3. — P. 175–176.

[68] Dynamic performance optimization for cloud computing using M/M/m queueing system / Lizheng Guo [et al.] // Journal of applied mathematics. — 2014. — Vol. 2014.

[69] Bahad Pritika, Saxena Preeti. Study of adaboost and gradient boosting algorithms for predictive analytics // International Conference on Intelligent Computing and Smart Communication 2019 / Springer. — 2020. — P. 235–244.

[70] Artalejo Jesus R., Gomez-Corral A. Retrial Queueing Systems: A Computational Approach. — Springer-Verlag Berlin Heidelberg, 2008.

[71] Falin G., Templeton J. Retrial queues. — CRC Press, 1997. — Vol. 75.

[72] Егоров И. М. Программирование: Учебное методическое пособие. — Томск: Томский межвузовский центр дистанционного образования, 2006. — P. 79.

[73] Falin G. Model of coupled switching in presence of recurrent calls // Engineering Cybernetics. — 1979. — Vol. 17, no. 1. — P. 53–59.

[74] Artalejo Jesus R., Resing JAC. Mean value analysis of single server retrial queues // Asia-Pacific Journal of Operational Research. — 2010. — Vol. 27, no. 03. — P. 335–345.

[75] Fischer W., Meier-Hellstern K. The Markov-modulated Poisson process (MMPP) cookbook // Performance evaluation. — 1993. — Vol. 18, no. 2. — P. 149–171.

[76] Meiert K. S. A Statistical procedure for fitting Markov-modulated Poisson process (applied probability). — 1986.

[77] Meier-Hellstern K. A fitting algorithm for Markov-modulated Poisson processes having two arrival rates // European Journal of Operational Research. — 1987. — Vol. 29, no. 3. — P. 370–377.

[78] IEEE 802.3 az: the road to energy efficient ethernet / K. Christensen [et al.] // IEEE Communications Magazine. — 2010. — Vol. 48, no. 11. — P. 50–56.

[79] Blaginin A. L., Lapatin I. L. The Two-Dimensional Output Process of Retrial Queue with Two-Way Communication // Information Technologies and Mathematical Modelling. Queueing Theory and Applications: 19th International Conference, ITMM 2020, Named after AF Terpugov, Tomsk, Russia, December 2–5, 2020, Revised Selected Papers / Springer Nature. — P. 279–290.

[80] Lapatin I. L., Nazarov A. A., Blaginin A. L. Asymptotic analysis of the output process in retrial queue with two-way communication and MAP input. — 2020.

[81] Kendall D. G. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain // The Annals of Mathematical Statistics. — 1953. — P. 338–354.

[82] Kulkarni V. G. On queueing systems with retrials // Journal of Applied Probability. — 1983. — P. 380–389.

[83] Gharbi N., Dutheillet C. An algorithmic approach for analysis of finite-source retrial systems with unreliable servers // Computers & Mathematics with Applications. — 2011. — Vol. 62, no. 6. — P. 2535–2546.

[84] Nazarov A. A., Paul S., Gudkova I. Asymptotic analysis of Markovian retrial queue with two-way communication under low rate of retrials condition. — 2017.

[85] Burke P. J. The output process of a stationary M/M/s queueing system // The Annals of Mathematical Statistics. — 1968. — Vol. 39, no. 4. — P. 1144–1152.

[86] Paul S., Phung-Duc T. Retrial Queueing Model with Two-Way Communication, Unreliable Server and Resume of Interrupted Call for Cognitive Radio Networks // Information Technologies and Mathematical Modelling. Queueing Theory and Applications. — Springer, 2018. — P. 213–224.

[87] Mirasol N. M. The output of an M/G/∞ queuing system is poisson // Operations Research. — 1963. — Vol. 11, no. 2. — P. 282–284.

[88] Daley D. J. Queueing output processes // Advances in Applied Probability. — 1976. — Vol. 8, no. 2. — P. 395–415.

[89] Wieland J. R., Pasupathy R., Schmeiser B. W. Queueing network simulation analysis: queueing-network stability: simulation-based checking // Proceedings of the 35th conference on Winter simulation: driving innovation. — 2003. — P. 520–527.

[90] ПрЭВМ №2020664030. — `https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2020664929&TypeFile=html`. — Accessed: 2021-06-02.

[91] Благинин А., Лапатин И., Назаров А. Исследование двумерного выходящего потока марковской модели узла обработки запросов с повторными обращениями и вызываемыми заявками // Информационные технологии и математическое моделирование (ИТММ-2020) : материалы XIX Международной конференции имени А. Ф. Терпугова. — Томск: Изд-во НТЛ, 2021. — P. 159–165.

[92] Lapatin I. L., Nazarov A. A., Blaginin A. L. Asymptotic Analysis of the Output Process in Retrial Queue with Two-Way Communication and MAP Input // Пятая Международная конференция по стохастическим методам (МКСМ-5) : материалы Международной научной конференции. — Изд-во РУДН, 2020. — P. 340–344.

[93] Лапатин И. Л., Назаров А. А., Благинин А. Л. Исследование выходящего потока марковской модели узла обработки запросов с повторными обращениями и вызываемыми заявками // Математика, ее приложения и ма-

тематическое образование (МПМО'20) : материалы VII Международной конференции. — Улан-Удэ: Изд-во ВСГУТУ, 2020. — P. 157–159.

[94] Lauder Anthony, Kent Stuart. Precise visual specification of design patterns // European Conference on Object-Oriented Programming / Springer. — 1998. — P. 114–134.

[95] Yang Gang, Yao Luo-gen, Ouyang Zi-sheng. The MAP/PH/N retrial queue in a random environment // Acta Mathematicae Applicatae Sinica, English Series. — 2013. — Vol. 29, no. 4. — P. 725–738.

[96] O'dwyer Arthur. Mastering the C++ 17 STL: Make full use of the standard library components in C++ 17. — Packt Publishing Ltd, 2017.

[97] JSON: data model, query languages and schema specification / Pierre Bourhis [et al.] // Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems. — 2017. — P. 123–135.

[98] Taylor Dave. Wicked Cool Shell Scripts: 101 Scripts for Linux, Mac OS X, and Unix Systems. — No Starch Press, 2004.

[99] Hunt John. Multiprocessing // Advanced Guide to Python 3 Programming. — Springer, 2019. — P. 363–376.

[100] Nussbaumer Henri J. The fast Fourier transform // Fast Fourier Transform and Convolution Algorithms. — Springer, 1981. — P. 80–111.

[101] Bergland Glenn D. A guided tour of the fast Fourier transform // IEEE spectrum. — 1969. — Vol. 6, no. 7. — P. 41–52.

[102] Array programming with NumPy / Charles R Harris [et al.] // Nature. — 2020. — Vol. 585, no. 7825. — P. 357–362.

[103] Вишневский ВМ, Дудин АН, Клименок ВИ. Стохастические системы с коррелированными потоками. Теория и применение в телекоммуникационных сетях // М.: Техносфера. — 2018.