**Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный университет технологий и управления имени К.Г. Разумовского**

**(Первый казачий университет)»**

|  |
| --- |
| Университетский колледж информационных технологий |

**Отчет**

**О прохождении практики**

**УП.02.01 Учебная практика**

**по профессиональному модулю**

**ПМ.02 Осуществление интеграции программных модулей**

*(вид, тип практики)*

|  |
| --- |
| **Обучающимся Мусатовым Даниилом Романовичем** |
| Направления подготовки/специальности  **09.02.07 Информационные системы и программирование** |
| Профиль подготовки (специализация/квалификация) программист |
| Форма обучения очная |
| Курс 3 |
| Группа 090207–9о–пр–21/1 |

практикант, ФГБОУ ВО «МГУТУ им. К.Г. Разумовского (ПКУ)» УКИТ

*(должность в которой проходил практику, наименование организации/предприятия)*

с «09» июня 2025 г. по «28» июня 2025 г.

Подпись обучающегося

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_Д.Р. Мусатов\_\_\_\_\_

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

Подпись руководителя

практики от Университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ⠀⠀ И.А. Озеркова⠀\_\_\_\_\_

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

Отчет принял

Заместитель директора по учебно–методической работе

Университетского колледжа информационных технологий

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ⠀⠀⠀⠀⠀ Е.В. Вернер⠀⠀⠀⠀\_⠀⠀

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc201835005)

[ОСНОВНАЯ ЧАСТЬ 7](#_Toc201835006)

[1. Анализ предметной области 7](#_Toc201835007)

[1.1 Общая постановка задач 7](#_Toc201835008)

[1.2 Уточнение постановки задачи 7](#_Toc201835009)

[1.3 Математическая модель 10](#_Toc201835010)

[2. Обоснование выбора средств методов и средств 11](#_Toc201835011)

[2.1 Описание метода решения 11](#_Toc201835012)

[2.2 Анализ средств разработки 11](#_Toc201835013)

[3. Проектирование приложения 13](#_Toc201835014)

[3.1 UML-диаграммы 13](#_Toc201835015)

[3.2 Wireframe-эскизы приложения 16](#_Toc201835016)

[3.3 Описание выбранных форматов данных 21](#_Toc201835017)

[4. Разработка алгоритмов 24](#_Toc201835018)

[5. Разработка приложения 25](#_Toc201835019)

[6. Тестирование приложения 26](#_Toc201835020)

[7. Анализ качества моделирования 28](#_Toc201835021)

[ЗАКЛЮЧЕНИЕ 30](#_Toc201835022)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc201835023)

[ПРИЛОЖЕНИЕ А Техническое задание 34](#_Toc201835024)

[Приложение Б код приложения 44](#_Toc201835025)

[ПРИЛОЖЕНИЕ В окна Программа 76](#_Toc201835026)

# ВВЕДЕНИЕ

Учебная практика УП02.01 проходила в лаборатории колледжа «Университетский колледж информационных технологий им. К.Г. Разумовского».

разработка автоматизированной системы (AC), включающей модули для моделирования и аналитической обработки. Данная система предназначается для наглядной демонстрации методов анализа и визуализации процесса вычисления площади крупного сегмента, ограниченного окружностью радиуса R (с центром в точке с координатами (x₀, y₀)) и прямой, параллельной вертикальной координатной оси и находящейся на расстоянии C от неё. Расчёты производились с использованием как математических формул, так и метода Монте–Карло.

В рамках практики были поставлены следующие задачи:

* формирование требований, необходимых для решения поставленной преподавателем задачи;
* составление технического задания на основе исходной формулировки;
* изучение теоретической базы, касающейся вычисления площади сегмента заданной фигуры;
* проектирование и реализация пользовательского интерфейса, обеспечивающего интуитивную и удобную работу с программой;
* написание программного кода, использующего алгоритм Монте–Карло для получения площади заданной области;
* создание различных окон приложения, включая стартовую заставку, основной модуль с вводом числа точек, интерфейс численного и графического моделирования, формы для отображения результатов, модуль анализа, а также окно с информацией о приложении;
* добавление встроенной справочной системы для помощи пользователям в освоении программы.

В ходе практики предусматривалось развитие следующих профессиональных компетенций:

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Также были продемонстрированы следующие общие компетенции:

ОК 01 Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам

ОК 02 Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности

ОК 03 Планировать и реализовывать собственное профессиональное и личностное развитие

ОК 04 Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством,

ОК 05 Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста

ОК 06 Проявлять гражданскопатриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, применять стандарты антикоррупционного поведения

ОК 07 Содействовать сохранению окружающей среды, умения: соблюдать нормы экологической безопасности; определять направления ресурсосбережения в рамках профессиональной деятельности по специальности, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях

ОК 08 Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

ОК 09 Использовать информационные технологии в профессиональной деятельности

ОК 10 Пользоваться профессиональной документацией на государственном и иностранном языках

ОК 11 Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере

# ОСНОВНАЯ ЧАСТЬ

## **Анализ предметной области**

## **Общая постановка задач**

Разработка направлена на создание программы, которая сможет вычислять площадь большей части круга, отсекаемой горизонтальной прямой, используя два подхода:

1. Аналитический – точный математический расчёт;
2. Стохастический – приближённый расчёт методом Монте-Карло.

Приложение должно содержать в себе следующие формы:

* заставка с указанием автора и варианта работы;
* основная форма с вводом данных, численным и графическим моделированием и сохранением данных моделирования;
* форма анализа моделирования, включающая в себя графический анализ всех выполненных попыток моделирования;
* форма «О программе»;
* встроенная справка по работе с программой.

Дальше речь пойдет об уточнении поставленной задачи.

## **Уточнение постановки задачи**

IDEF0 наглядно демонстрирует логические связи между операциями. С помощью двух диаграмм IDEF0 (общей контекстной и декомпозиции) можно проследить общую логическую структуру всех объектов.

Общая контекстная диаграмма (A0) на рисунке 1 показывает ключевое действие системы - "Вычислить площадь методом Монте-Карло" (A0). На вход этому блоку поступают данные: "Количество точек" и "Параметры окружности и линии". На выходе формируются "Результаты анализа вычислений", "Графическое представление математического моделирования" и "Сохранённые результаты вычислений". В качестве рабочей среды (механизма) используется "Программа Monte-Karlo", а управление процессом обеспечивают "Методы статистического анализа", "Техническое задание" и "Вычислительные методы". Для хранения результатов задействована "База данных SQLite".

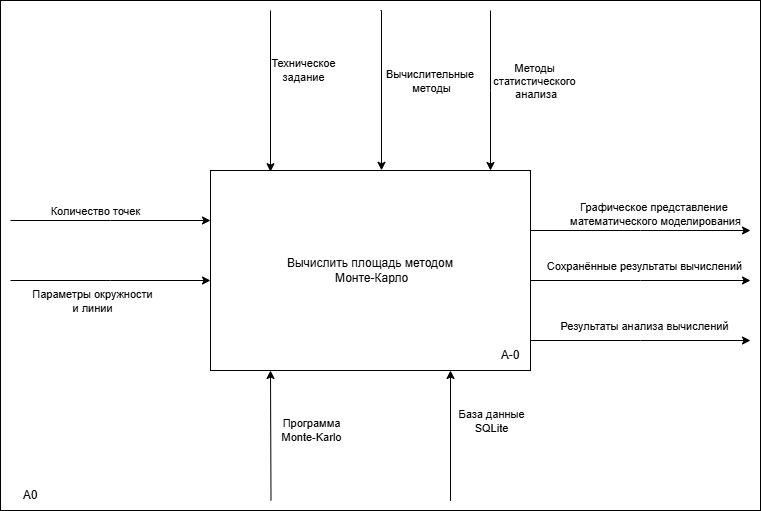


Рисунок 1 – диаграмма IDEF0

Диаграмма декомпозиции (A1-A5) на рисунке 2 раскрывает внутреннюю структуру процесса A0, детализируя его на основные подпроцессы:

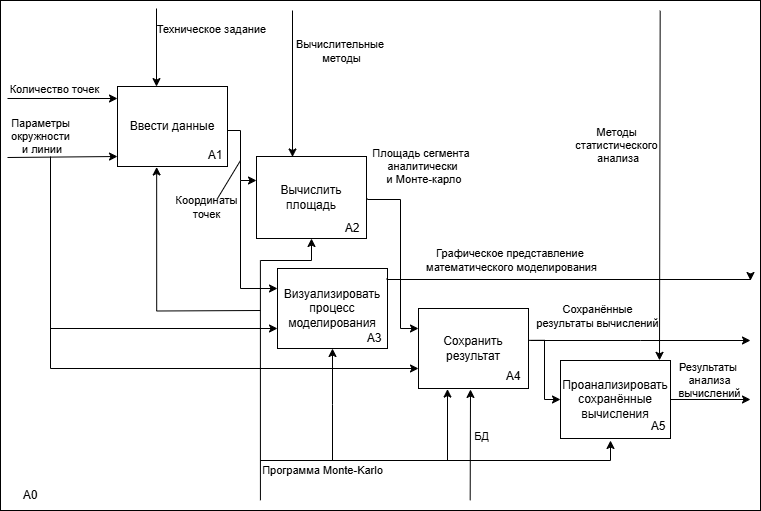


Рисунок 2 – диаграмма IDEF0 с декомпозицией

* A1 "Ввести данные". Принимает на вход, "Количество точек" и "Параметры окружности и линии". "Техническое задание" определяет ограничения на вводимые значения и значения по умолчанию. Результатом работы являются "Координаты точек". Механизм - "Программа Monte-Karlo";
* A2 "Вычислить площадь". Использует "Координаты точек" (выход A1) для выполнения используется механизм "Программа Monte-Karlo", применяющий "Вычислительные методы". Вычисляет "Площадь сегмента аналитически и Монте-карло";
* A3 "Визуализировать процесс моделирования". Принимает результаты вычислений и "Параметры окружности и линии". Основная задача - создать "Графическое представление математического моделирования". Механизм - "Программа Monte-Karlo";
* A4 "Сохранение результаты вычислений". Принимает результаты вычислений. Сохраняет их в "БД" (База данных SQLite). Механизмы - "Программа Monte-Karlo" и "БД";
* A5 "Проанализировать сохранённые вычисления": Принимает сохраненные результаты (выход A4) и "Площадь сегмента аналитически и Монте-карло" (выход A2). Выполняет "Анализ вычислений" с использованием "Методов статистического анализа". Формирует итоговые "Результаты анализа вычислений". Механизм - "Программа Monte-Karlo".

Ниже приводится описание математической модели.

## **Математическая модель**

Математическое моделирование проводится с помощью графического калькулятора Desmos и получается следующий результат, показанный на рисунке 3 при: x0 = 3, y0 = 1, R = 2, C = 2 и горизонтально направленной линии.

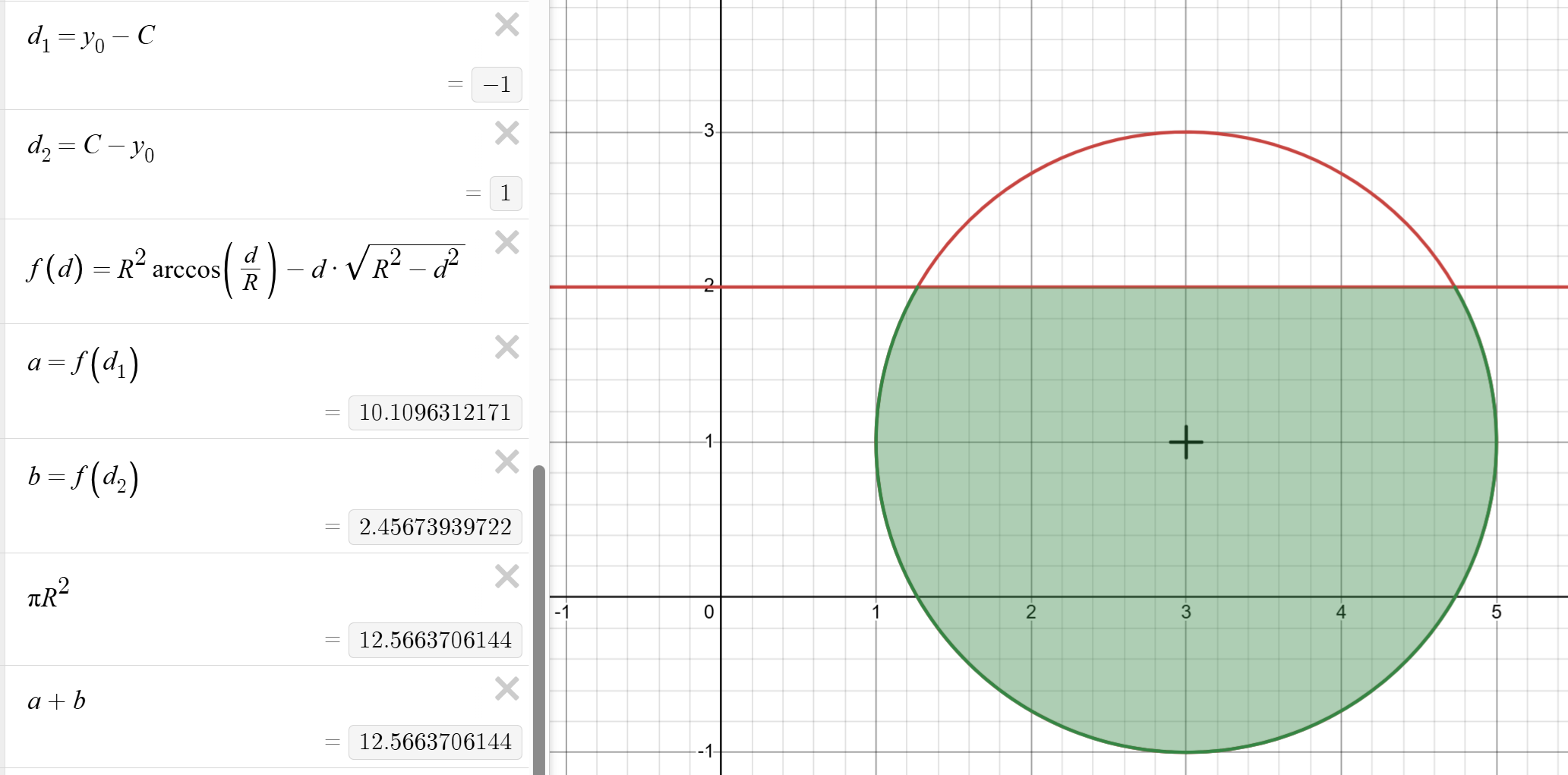


Рисунок 3 – математическое моделирование с помощью Desmos

В результате этого математического моделирования также показывается формула для вычисления площади сегмента окружности, и что сумма этих сегментов равна площади окружности.

## **Обоснование выбора средств методов и средств**

## **Описание метода решения**

Для определения площади большего сегмента будут использоваться 2 подхода:

* аналитический. Даёт точный результат площади сегмента. Формула площади сегмента: S = R²arccos(d/R) - d√(R²-d²), где d = |C - x0| (или y0);
* стохастический. Будет использоваться метод Монте-Карло, в котором будут случайно генерироваться точки в квадрате, описывающем окружность и проверяться, находятся ли сгенерированные точки в большем сегменте окружности. Формула для метода Монте-Карло: S ≈ (Nточек\_внутри/Nточек\_всего) \* Sквадрата.

Важно отметить, что так как метод Монте-Карло стохастический, то он даёт приближённый результат, который тем точнее, чем больше точек проверяется в одном эксперименте, и чем больше проведено экспериментов.

## **Анализ средств разработки**

При разработке программного обеспечения используется язык программирования C#, среда разработки Microsoft Visual Studio и система хранения данных на основе SQLite с применением Entity Framework Core.

Основным преимуществом C# является его интеграция с платформой .NET, что обеспечивает высокую производительность, безопасность и широкие возможности для разработки приложений. C# поддерживает современные парадигмы программирования, включая ООП и асинхронные операции, а также обладает богатой стандартной библиотекой. Кроме того, язык активно развивается и имеет большое сообщество разработчиков.

Недостатком можно считать ограниченную кроссплатформенность (хотя .NET Core и .NET 5+ улучшили эту ситуацию), а также зависимость от экосистемы Microsoft. Однако для Windows-приложений, таких как Windows Forms, C# является одним из лучших выборов.

Среда разработки выбрано Microsoft Visual Studio, так как она предоставляет разнообразные инструменты для написания, отладки и тестирования кода. Среди ключевых преимуществ:

* встроенный визуальный дизайнер форм для Windows Forms, ускоряющий разработку интерфейсов;
* поддержка Entity Framework Core и других ORM, упрощающих работу с базами данных;
* интегрированные средства профилирования и диагностики производительности;
* поддержка Git и систем управления проектами.

Недостатки включают высокие требования к ресурсам (особенно в версии с полным функционалом) и ограниченную поддержку других ОС (хотя есть Visual Studio Code как альтернатива).

Система хранения данных: SQLite + Entity Framework Core. Использование SQLite в связке с EF Core обеспечивает удобное и надежное хранение структурированных данных. Преимущества такого подхода:

* легковесность. SQLite не требует отдельного сервера, данные хранятся в одном файле;
* производительность. Эффективная работа даже с большими объемами данных;
* гибкость. EF Core позволяет легко манипулировать данными через LINQ;
* масштабируемость. Возможность создания сложных связей между таблицами.

Из недостатков можно отметить ограничения SQLite при работе в многопользовательских сценариях, но для локального приложения это не критично.

Как итог, выбранный стек технологий (C#, Windows Forms, .NET 9.0, SQLite + EF Core) оптимален для разработки десктопного приложения с хранением структурированных данных экспериментов. C# и Visual Studio обеспечивают высокую скорость разработки, а SQLite и EF Core – удобное управление данными. Альтернативы (например, MySQL или PostgreSQL) могли бы быть избыточны для локального хранения, а Excel и JSON медленнее и менее надежны для сложных структур.

## **Проектирование приложения**

## **3.1 UML-диаграммы**

UML диаграммы для приложения помогают наглядно отображать структуру, поведение и взаимодействие компонентов, упрощая понимание сложных систем.

**3.1.1 Диаграмма прецедентов**

UML–диаграмма прецедентов показывает, как пользователь взаимодействует с системой. Диаграмма на рисунке 4 показывает, какие действия может выполнить пользователь и в какой последовательности может их выполнять.

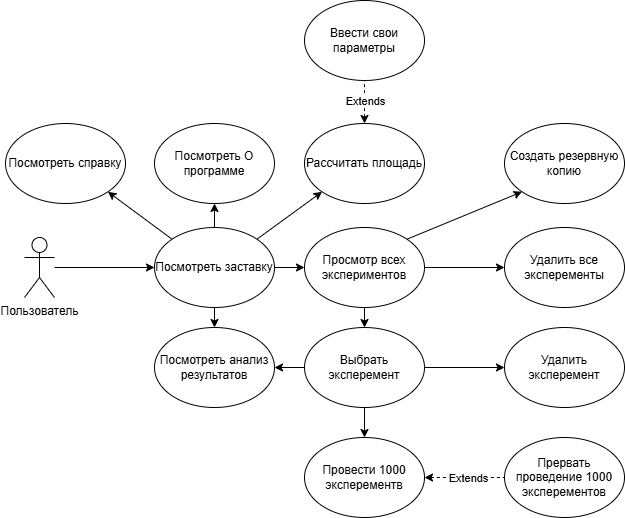


Рисунок 4 – диаграмма прецедентов

С помощью диаграммы прецедентов также можно запланировать какие функции к какой форме будут относиться.

**3.1.2 Диаграмма деятельности**

UML-диаграмма деятельности демонстрирует последовательность действий в системе, описывая её логику и конкретные шаги. На такой диаграмме представлены действия, а также показаны их взаимосвязи и порядок выполнения. Общая диаграмма деятельности для моей АС на рисунке 5.

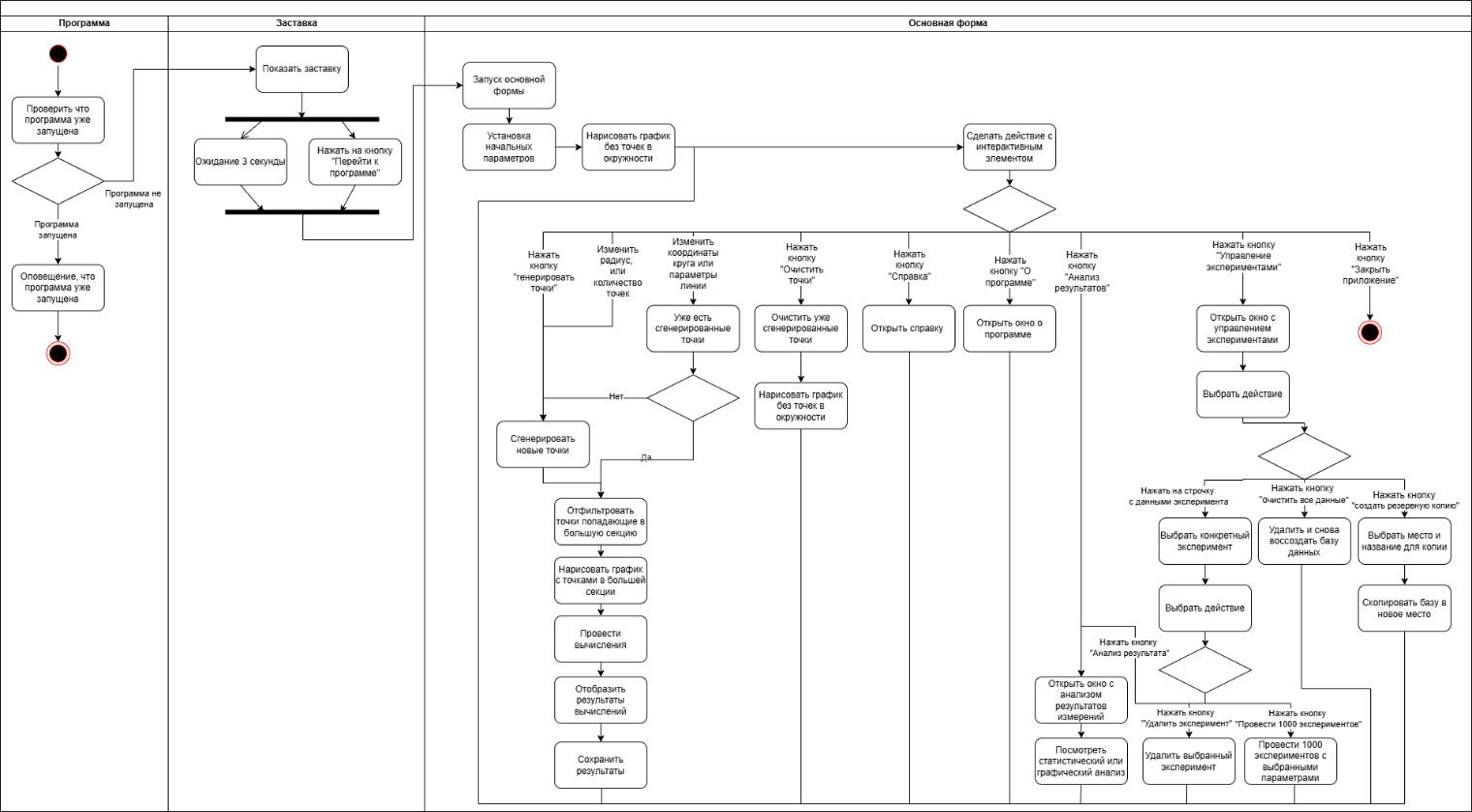


Рисунок 5 – диаграмма деятельности

С помощью этой диаграммы можно запланировать какие интерактивные элементы будут в программе и как будут вызываться функции в программе.

**3.1.3 Диаграмма последовательности**

UML-диаграмма последовательностей представляет собой диаграмму, отображающую взаимодействие объектов в системе на основе последовательности событий во времени. Является важной частью моделирования системы, позволяя представить последовательность действий и коммуникаций между объектами. Так как в АС достаточно много объектов, взаимодействующих между собой, было принято решение на диаграмме, созданной с помощью Plant UML, на рисунке 6 изобразить самое сложное взаимодействие, а именно инициализация объектов и возможная генерация точек с отображением результата.

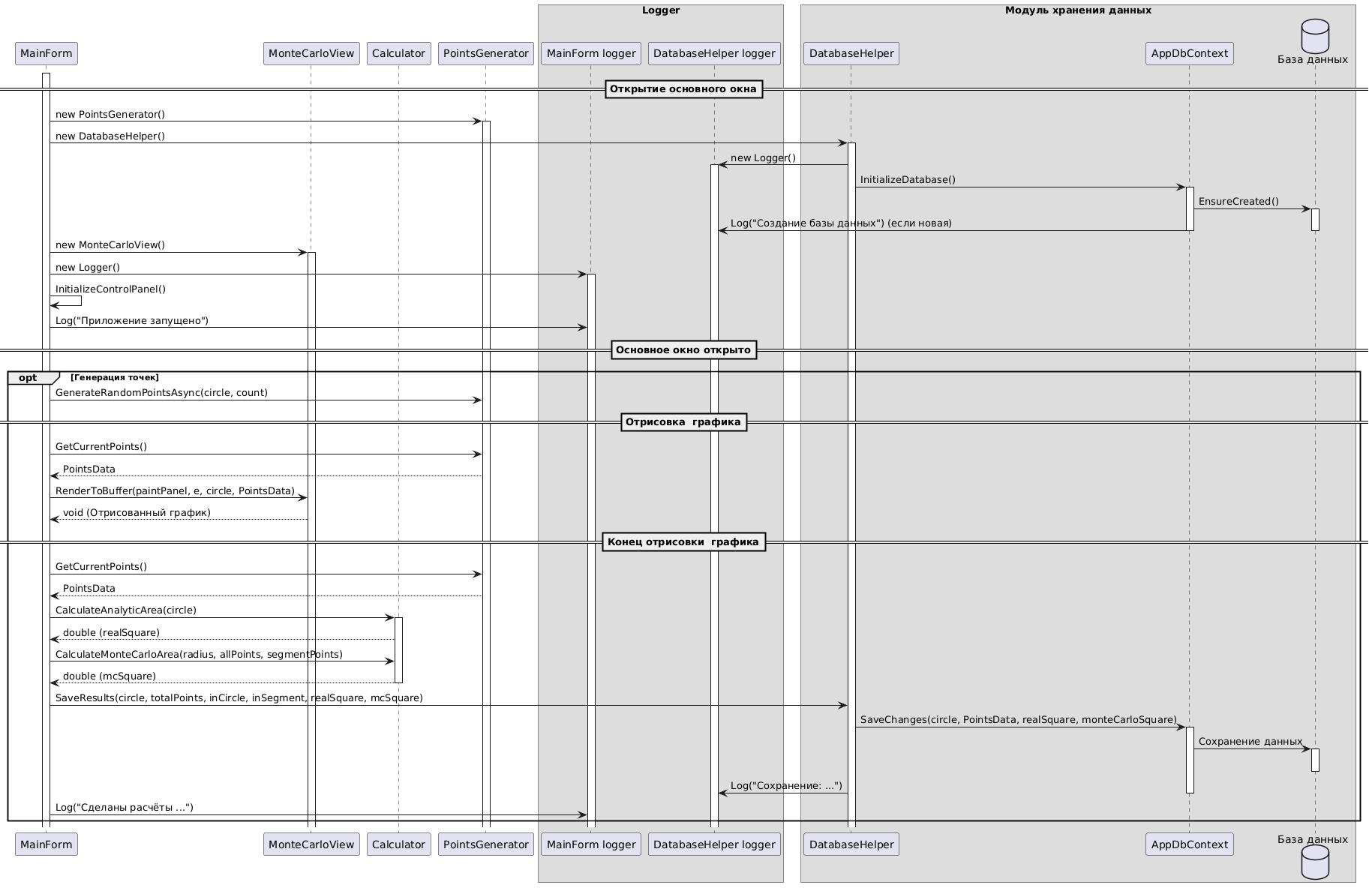


Рисунок 6 – диаграмма последовательности для инициализации объектов и возможной генерации точек

С помощью диаграммы можно понять сколько будут существовать объекты в программе и как часто они используются.

## **3.2 Wireframe-эскизы приложения**

Разработка программы подразумевает многооконное приложение с несколькими формами. Каждый Wireframe-эскиз показывает, как будет выглядеть каждая из форм в приложении ещё до начала разработка АС и при разработке поможет упростить процесс.

Заставка, на рисунке 7, является первым что видит пользователь. Как можно увидеть на эскизе, она будет показывать значения для расчётов по умолчанию, автора программы и картинку программы, как на рисунке 8, показывающую как будет выглядеть график с точками.

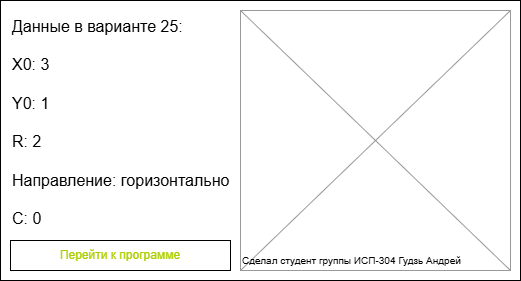


Рисунок 7 – эскиз заставки приложения

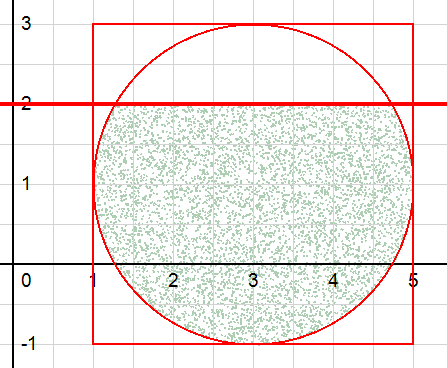


Рисунок 8 – картинка на заставке приложения

От заставки будет переход к основному окну, эскиз которого показан на рисунке 9. На основном окне будет меню, позволяющее переходить между формами, панель управления для математического моделирования и большое пространство для рисования графика справа от панели управления. На графике планируется нарисовать оси координат с подписями, квадрат, в области которого будут случайно размещаться точки, окружность и линия, отсекающая сегмент окружности. Отображать планируется только точки, попадающие в больший сегмент окружности.

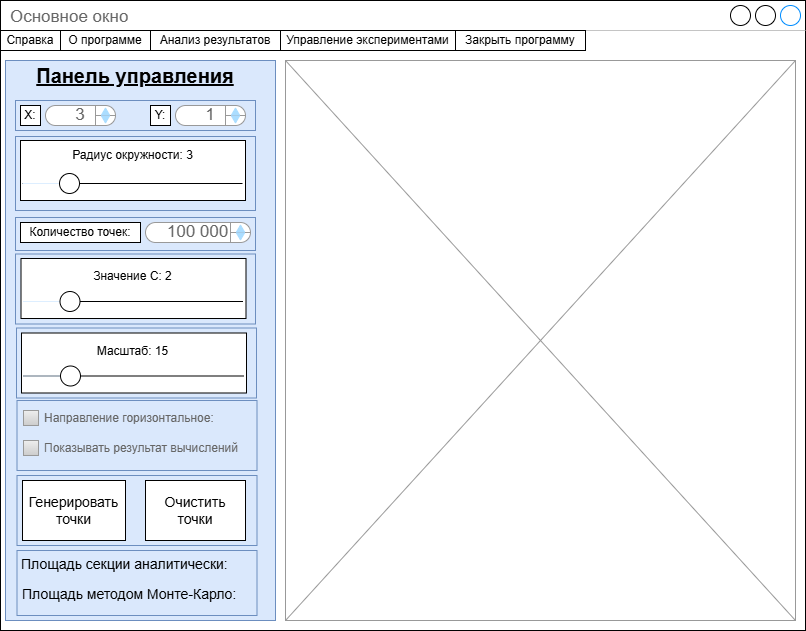


Рисунок 9 – эскиз основного окна

Справка планируется открываться в браузере с дизайном по умолчанию от приложения для создания справки Dr.Explain.

В окно «О программе» планируется разместить название программы, автора, версию приложения, ссылка на GitHub автора и иконка приложения в большом размере. Из интерактивных элементов планируется только большая кнопка «Закрыть». Внешний вид эскиза и иконки показаны на рисунке 10 и рисунке 11.

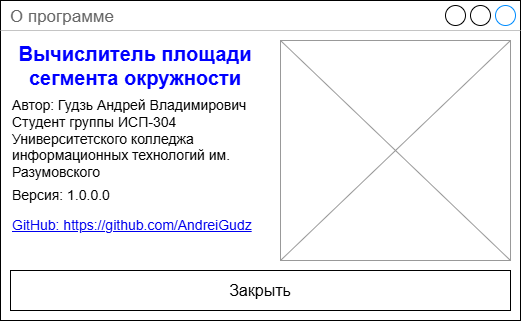


Рисунок 10 – эскиз окно «О программе»



Рисунок 11 – иконка приложения

В окне «Анализ результатов» планируется разместить 2 переключающиеся вкладки, чьи эскизы показаны на рисунке 12. На первой планируется числовой статистический анализ измерений и таблица с результатами эксперимента. На вкладке «Графический анализ» планируется разместить точечную диаграмму, на которой точками будут располагаться результаты вычисления площади методом Монте-Карло, а линиями результат статистического анализа. В правом верхнем углу будет легенда, объясняющая, что означает каждая цветная линия.

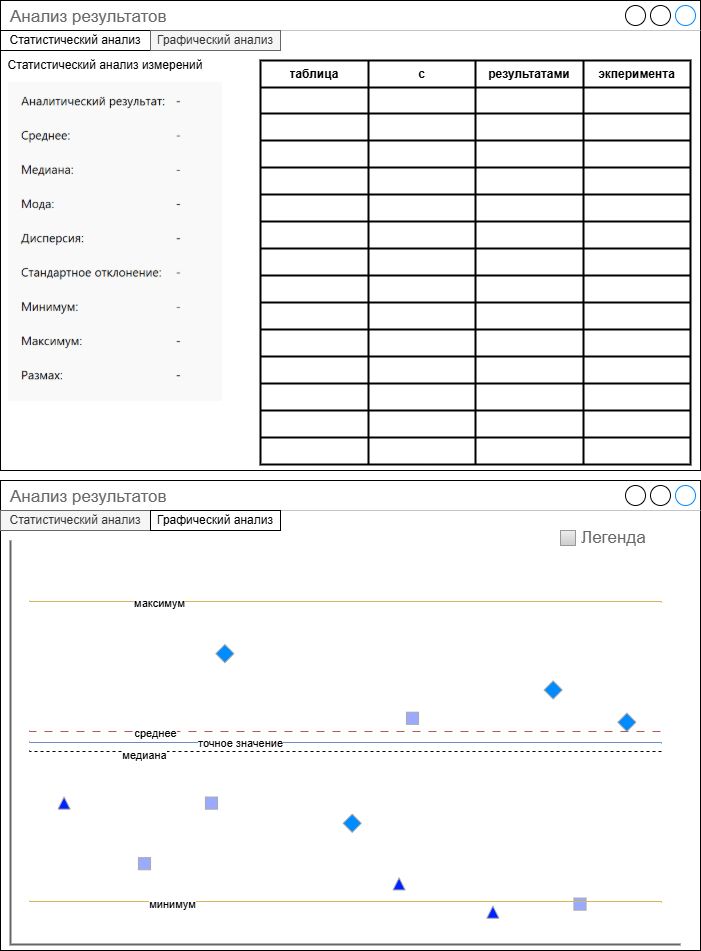


Рисунок 12 – эскиз вкладок в окне с анализом результатов

Окно для управления экспериментами будет содержать таблицу, в которой можно будет выбрать эксперимент, кнопки, влияющие на все эксперименты (такие как «Создать резервную копию» и «Очистить все данные») и кнопки, влияющие только на выбранный. Внизу будет отображаться стока со статусом всех экспериментов или текущей операции. Эскиз показан на рисунке 13.



Рисунок 13 – эскиз окна для управления экспериментами

В целом дизайн приложения планируется сделать простым. Основные цвета: белый, синий и серый.

## **3.3 Описание выбранных форматов данных**

Для реализации системы хранения результатов экспериментов метода Монте-Карло была разработана реляционная модель базы данных, состоящая из двух взаимосвязанных сущностей. Формат данных реализован с использованием SQLite через Entity Framework Core, что обеспечивает строгую типизацию и целостность информации. Логическая модель базы данных на рисунке 14.

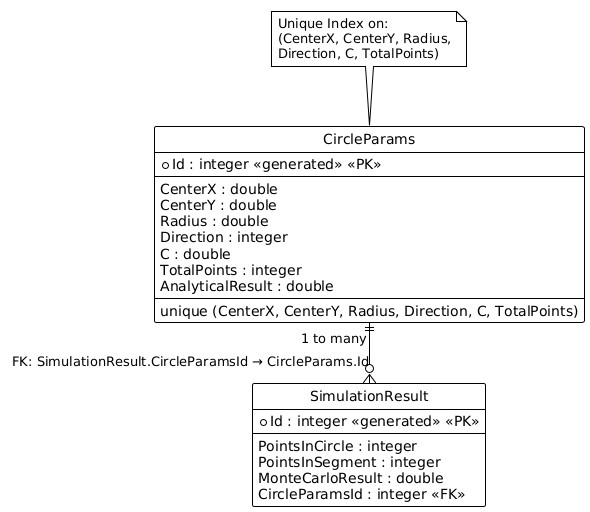


Рисунок 14 – логическая модель базы данных

Таблица CircleParams (Параметры окружности) содержит геометрические параметры фигур для экспериментов:

* Id (int, PK) – уникальный идентификатор записи;
* CenterX, CenterY (double) – координаты центра окружности;
* Radius (double) – радиус окружности;
* Direction (int) – направление сектора (перечисление);
* C (double) – константа для линейного уравнения;
* TotalPoints (int) – общее количество точек в эксперименте;
* AnalyticalResult (double) – аналитически рассчитанная площадь.

Таблица SimulationResult (Результаты симуляций) фиксирует статистику экспериментов:

* Id (int, PK) – уникальный идентификатор;
* PointsInCircle (int) – точки внутри окружности;
* PointsInSegment (int) – точки в целевом сегменте;
* MonteCarloResult (double) – вычисленная площадь методом Монте-Карло;
* CircleParamsId (int, FK) – связь с параметрами эксперимента.

Между таблицами установлена связь один-ко-многим: один набор параметров окружности может соответствовать нескольким результатам экспериментов. Для обеспечения уникальности параметров создан составной уникальный индекс по полям (CenterX, CenterY, Radius, Direction, C, TotalPoints).

Использование реляционной модели в SQLite обеспечивает критически важные для научных расчетов преимущества:

* целостность данных – внешние ключи гарантируют отсутствие "сиротствующих" результатов экспериментов без соответствующих параметров;
* эффективность запросов – уникальный индекс ускоряет поиск дубликатов параметров и снижает избыточность хранения;
* точность вычислений – тип double гарантирует сохранение дробных значений без потери точности;
* масштабируемость – нормализованная структура позволяет добавлять новые параметры экспериментов без изменения схемы;
* атомарность операций – транзакционная модель SQLite защищает данные от частичных обновлений при сбоях.

Такой подход особенно эффективен для научных задач, где требуется сопоставление параметров экспериментов с их результатами, статистический анализ серий вычислений и верификация данных. Реляционная модель обеспечивает однозначную интерпретацию связей между геометрическими параметрами и результатами симуляций, что принципиально важно для воспроизводимости экспериментов.

## **Разработка алгоритмов**

Для разработки алгоритмов была нарисована общая блок схема приложения. На рисунке 15 показано в общих чертах как реализован алгоритм на основной форме программы.

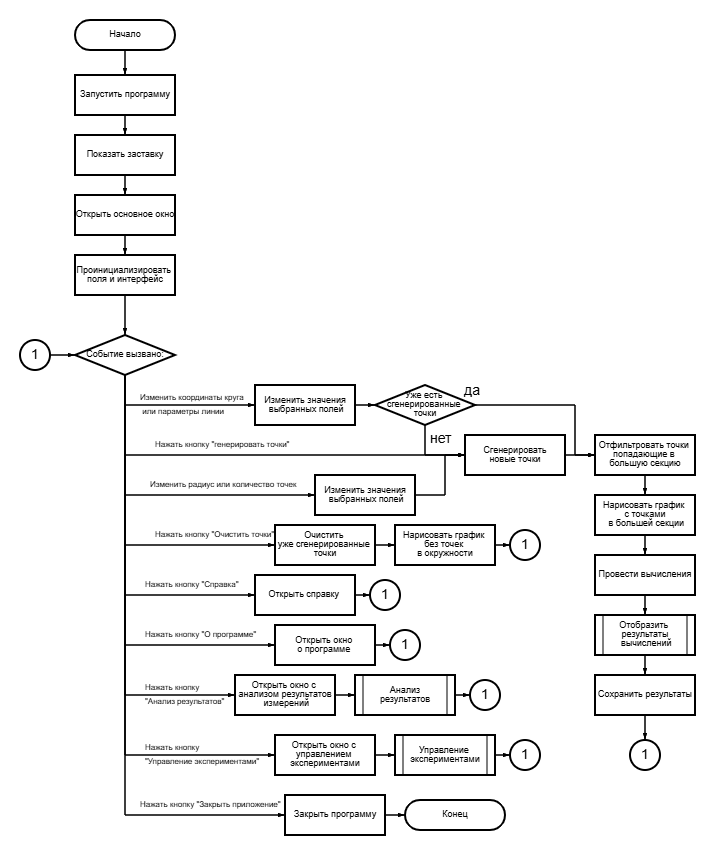


Рисунок 15 – общая блок схем приложения

Эта диаграмма поможет на этапе разработки приложения.

## **Разработка приложения**

Разработка приложения происходит в соответствии с её проектированием

При запуске исполняемого файла открывается заставка, показанная на рисунке В1 в приложении. Если программа уже была запущена, появится оповещение, что программа уже запущена, и не запустит дубликат программы.

Заставка показывается 3 секунды или пока не будет нажата кнопка «Перейти к программе». В любом из этих случаев произойдёт переход к основному окну.

В основном окне, показанном на рисунке В2 в приложении, происходит математическое моделирование методом Монте-Карло. Параметры окружности и линии связаны между собой и не дадут линии выйти за пределы окружности. В числовые поля, доступные для ввода с клавиатуры не получится ввести некорректные значения, такие как буквы или символы, а количество точек не получится задать меньше 1 и больше 106.

От основного окна с помощью верхнего меню происходит переход между другими окнами. Другие окна открываются как модальные, то есть пока открыто модальное окно, взаимодействовать с другими окнами приложения не получится. Кнопка «Закрыть приложение» просто закрывает приложение.

Нажав на справку, произойдёт запуск страницы с руководством пользователя в браузере по умолчанию. Пример этого показан на рисунке В3 в приложении.

В окне «О программе», показанном на рисунке В4 в приложении, находится информация об авторе и программе. Если нажать на ссылку на GitHub откроется репозиторий автора.

Окно с анализом, чьи вкладки показаны на рисунках В5 и В6 в приложении, отображают анализ экспериментов в разных форматах.

В окне с управлением экспериментами, показанном на рисунке В7 в приложении, происходит глобальное управление экспериментами. В нём можно создать резервную копию базы данных, удалить все результаты. Выбрав эксперимент, нажатием на строчку с экспериментом, можно удалить его, посмотреть анализ или провести сразу 1000 экспериментов. При проведении 1000 экспериментов, их можно прервать, нажав на кнопку «Прервать» как показано в приложении на рисунке В8, при этом уже сгенерированные эксперименты сохраняются.

Все операции с базой данных, кроме получения данных, записываются в файлы, в папке Logs.

## **Тестирование приложения**

Тестирование приложения проводилось комплексно с использованием юнит-тестов и интеграционных тестов, обеспечивающих проверку корректности работы всех компонентов системы. Всего было разработано 49 юнит-тестов и 1 интеграционный тест, что подтверждает высокую степень покрытия кода тестами. Результаты успешного прохождения тестов представлены на рисунке 16.

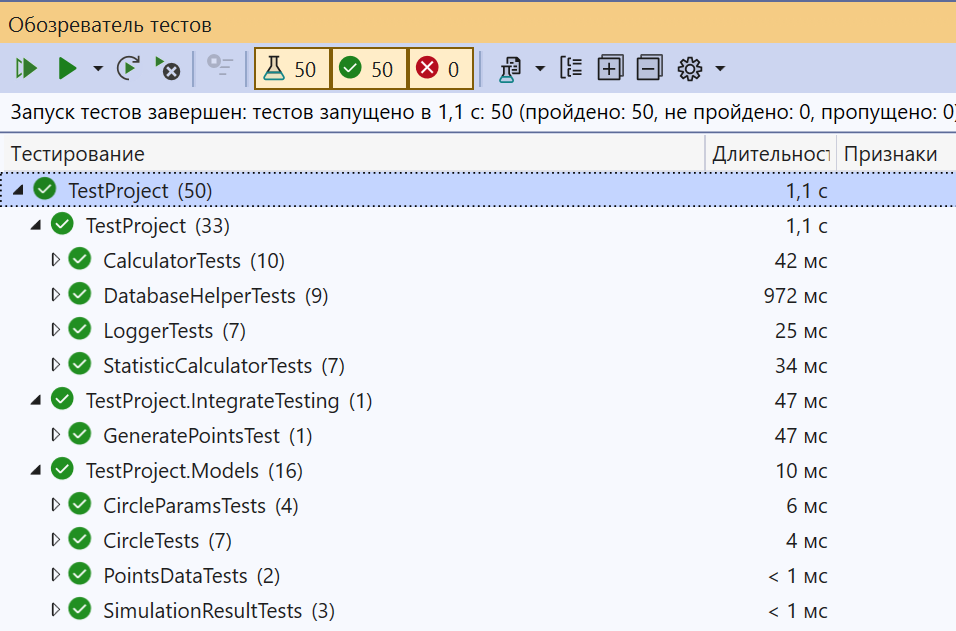


Рисунок 16 – результат прохождения авто тестов

Основные направления тестирования включают:

* тестирование математических вычислений. Были проверены все математические модули приложения и такие аспекты как: проверка корректности вычисления площади сегмента на основе соотношения точек, аналитические расчёты, тестирование граничных условий (нулевой радиус, секущая через центр) и обработка ошибок;
* тестирование моделей данных. Протестированы все сущности предметной области. Все сущности корректно инициализируются и работают, им нельзя присвоить некорректные параметры, и они правильно переводятся в строковую форму;
* тестирование работы с базой данных, а именно модуля DatabaseHelper. В нём проверялось создание и пересоздание базы, CRUD-операции, согласованность данных при добавлении связанных сущностей и механизм создания бэкапа;
* тестирование системы логирования. Автоматическое создание директорий и файлов, корректность добавления временных меток к логам, тестирование записи стек трейсов ошибок и самих ошибок и проверка работы в условиях отсутствия прав на запись;
* тестирование статистических вычислений в модуле StatisticCalculator. В него входило проверка правильности вычисления моды, медианы, размаха, дисперсии, стандартного отклонения;
* интеграционное тестирование. Проведено тестирование комплексного сценария: создание 100 000 случайных точек, проверка попадания точек в сегмент и расхождение между аналитическим и вероятностным методом ≤ 5%.

Также проходило тестирование интерфейса, проверяющее, что все запланированные элементы корректно отображаются и что переход между вкладками происходит корректно.

## **Анализ качества моделирования**

Основным результатом моделирования в рамках данной работы является оценка площади большого сегмента, образованного окружностью и горизонтальной прямой, с использованием метода Монте-Карло (МК). Качество полученных оценок анализировалось по следующим ключевым аспектам:

* зависимость точности от объема выборки (N). Как и ожидалось теоретически, метод Монте-Карло демонстрирует приблизительный характер результатов. Главным фактором, определяющим точность оценки площади, является количество случайных точек (N), сгенерированных внутри ограничивающего прямоугольника. Наблюдается четкая закономерность: погрешность оценки уменьшается по мере увеличения N. Результаты, полученные при малом N (например, 100-1000 точек), характеризуются значительным разбросом и высокой относительной погрешностью относительно аналитического значения. При увеличении N до 10⁴, 10⁵ и более точек, результаты становятся существенно стабильнее и ближе к истинному значению площади. Эта зависимость напрямую связана со статистической природой метода Монте-Карло: ошибка оценки убывает пропорционально 1/√N (согласно закону больших чисел и центральной предельной теореме);
* стабилизация оценки при повторных экспериментах. Важным подтверждением надежности метода и качества моделирования стало наблюдение за поведением оценки при многократном повторении расчета для одного и того же набора параметров (R, x0, y0, C) и фиксированного, но относительно малого N. Было установлено, что хотя результаты отдельных экспериментов с малым N могут сильно отклоняться от аналитического значения и друг от друга, их среднее арифметическое и медиана по серии таких экспериментов (например, 100 или 1000 запусков) устойчиво приближаются к точному аналитическому значению площади. Этот эффект является прямым следствием Закона Больших Чисел: среднее результатов независимых испытаний сходится к математическому ожиданию (которое в данном случае равно искомой площади) при увеличении числа испытаний. Сходимость медианы также подтверждает устойчивость оценки;
* источники погрешности. Основным источником погрешности в данной реализации является статистическая погрешность, присущая самому методу Монте-Карло. Она обусловлена случайным характером генерации точек и конечностью выборки N. Величина этой погрешности может быть оценена (например, через стандартную ошибку среднего или с использованием доверительных интервалов), и она управляема через параметр N. Потенциальные систематические погрешности (например, связанные с некорректной генерацией точек или проверкой попадания) в корректно реализованной модели должны быть пренебрежимо малы по сравнению со статистической погрешностью при достаточно больших N.

Как итог, можно сказать, что разработанная модель на основе метода Монте-Карло эффективна для решения поставленной задачи оценки площади сегмента. Модель демонстрирует предсказуемое поведение, соответствующее теоретическим основам метода: точность результата прямо и управляемо зависит от количества сгенерированных случайных точек (N). Наблюдаемая сходимость среднего и медианы результатов множества экспериментов с малым N к аналитическому значению является сильным эмпирическим подтверждением корректности реализации модели и статистической обоснованности получаемых с ее помощью оценок. Для получения оценки с заданной точностью необходимо выбирать N, соответствующее требуемому уровню статистической погрешности. Альтернативно, при ограниченных вычислительных ресурсах (малое N), повышения надежности оценки можно достичь путем усреднения результатов большого количества независимых прогонов модели.

# ЗАКЛЮЧЕНИЕ

В ходе прохождения учебной практики по профессиональному модулю ПМ.02 Осуществление интеграции программных модулей в период с 09.06.2025 по 28.06.2025 в Университетском колледже информационных технологий ФГБОУ ВО «МГУТУ им. К.Г. Разумовского (ПКУ)» в Лаборатории программного обеспечения и сопровождения компьютерных систем, освоены необходимые практические умения и закреплены навыки работы по данному модулю.

Целью практики являлось развитие практических навыков, умений и опыта, а также формирование компетенций в процессе выполнения определенных практических задач, связанных с будущей профессиональной деятельностью.

Выполнены задачи учебной практики: получен опыт выполнения работ по интеграции модулей программного обеспечения для компьютерных систем, освоены общие компетенции и профессиональные компетенции.

В ходе работы было выполнено:

1. Анализ и формулировка требований для решения задачи расчёта площади большого сегмента окружности;
2. Разработка технического задания на основе условий, предоставленных преподавателем;
3. Исследование математических основ расчёта площади сегмента с использованием аналитического метода и метода Монте-Карло;
4. Проектирование и реализация удобного графического интерфейса, включающего: заставку, основной модуль для ввода параметров (радиус окружности, координаты центра, расстояние до прямой), модуль численного и графического моделирования, сохранение и анализ результатов, справочную систему;
5. Программная реализация метода Монте-Карло для вычисления площади большего сегмента с визуализацией процесса;
6. Интеграция встроенной справочной системы для поддержки пользователей;
7. Тестирование и отладка приложения, проверка корректности расчётов.

Таким образом, в ходе практики были успешно выполнены все поставленные задачи: разработано программное обеспечение для вычисления площади большого сегмента аналитическим методом и методом Монте-Карло, создан удобный интерфейс с поддержкой пользователя. Приобретены практические навыки проектирования, программирования и интеграции программных модулей, что способствует формированию профессиональных компетенций в области разработки программного обеспечения. Полученный опыт будет полезен в дальнейшей учебной и профессиональной деятельности.

# СПИСОК ЛИТЕРАТУРЫ

1. Система вопросов и ответов о программировании: [Электронный ресурс]. URL: <https://stackoverflow.com/questions/> (дата обращения: 21.06.2024)
2. Система вопросов и ответов о программировании на русском языке: [Электронный ресурс]. URL: <https://ru.stackoverflow.com/questions/> (дата обращения: 22.06.2024)
3. Как оформить список литературы по госту: [Электронный ресурс]. URL: https://journal.tinkoff.ru/how-to-make-bibliography/ (дата обращения: 22.06.2024)
4. Документация по MySQL Workbench: [Электронный ресурс]. URL: <https://dev.mysql.com/doc/workbench/en/> (дата обращения: 15.06.2024)
5. MySQL 8.4 Reference Manual: [Электронный ресурс]. URL: <https://dev.mysql.com/doc/refman/8.4/en/> (дата обращения: 15.06.2024)
6. Исчерпывающее руководство по Юникоду и кодировке символов в Python: [Электронный ресурс]. URL: <https://tproger.ru/translations/unicode-and-encoding-python-guide> (дата обращения: 18.06.2024)
7. How to support full Unicode in MySQL databases: [Электронный ресурс]. URL: <https://mathiasbynens.be/notes/mysql-utf8mb4#character-sets> (дата обращения: 18.06.2024)
8. Справочник по Transact-SQL (ядро СУБД): [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/sql/t-sql/language-reference?view=sql-server-ver16> (дата обращения: 24.06.2024)
9. SQL Tutorial: [Электронный ресурс]. URL: <https://www.w3schools.com/sql/default.asp> (дата обращения: 25.06.2024)
10. Транзакции в MySQL: [Электронный ресурс]. URL: <https://www.garb.ru/blog/transaction.html> (дата обращения: 22.06.2024)
11. Сброс AUTO\_INCREMENT в MySQL: как принудительно начать с 1: [Электронный ресурс]. URL: <https://sky.pro/wiki/sql/sbros-auto-increment-v-my-sql-kak-prinuditelno-nachat-s-1/> (дата обращения: 21.06.2024)
12. Типы данных MySQL: [Электронный ресурс]. URL: <https://metanit.com/sql/mysql/2.3.php> (дата обращения: 18.06.2024)
13. Сообщение об ошибке «входная строка имела неверный формат» при отсутствии параметра decimal во время функции службы Web в Microsoft Dynamics NAV 2009: [Электронный ресурс]. URL: [https://support.microsoft.com/ru-ru/topic/сообщение-об-ошибке-входная-строка-имела-неверный-формат-при-отсутствии-параметра-decimal-во-время-функции-службы-web-в-microsoft-dynamics-nav-2009-8286ef77-cd0a-ae9a-d912-136c2fa2e5d1](https://support.microsoft.com/ru-ru/topic/%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BE%D0%B1-%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B5-%D0%B2%D1%85%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F-%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0-%D0%B8%D0%BC%D0%B5%D0%BB%D0%B0-%D0%BD%D0%B5%D0%B2%D0%B5%D1%80%D0%BD%D1%8B%D0%B9-%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82-%D0%BF%D1%80%D0%B8-%D0%BE%D1%82%D1%81%D1%83%D1%82%D1%81%D1%82%D0%B2%D0%B8%D0%B8-%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B0-decimal-%D0%B2%D0%BE-%D0%B2%D1%80%D0%B5%D0%BC%D1%8F-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D1%8B-web-%D0%B2-microsoft-dynamics-nav-2009-8286ef77-cd0a-ae9a-d912-136c2fa2e5d1) (дата обращения: 24.06.2024)
14. Grant a user permission to only view a MySQL view and nothing else: [Электронный ресурс]. URL: https://stackoverflow.com/questions/3108656/grant-a-user-permission-to-only-view-a-mysql-view-and-nothing-else (дата обращения: 21.06.2024)
15. Использование триггеров базы данных MySQL: [Электронный ресурс]. URL: <https://www.digitalocean.com/community/tutorials/how-to-manage-and-use-mysql-database-triggers-on-ubuntu-18-04-ru> (дата обращения: 21.06.2024)

# ПРИЛОЖЕНИЕ А Техническое задание

**Техническое задание**

1. **Общие сведения**

Наименование: Вычислитель площади сегмента окружности.

Условное обозначение: Monte-Karlo v1.0.

Наименование организации: Университетский колледж информационных технологий им. Разумовского.

Разработчик: Гудзь Андрей Владимирович

Перечь документов, на основании которых создается АС утверждено руководителем практики 09.06.2025.

Плановые сроки начала и окончания работ по созданию АС

Начало работ: 09.06.2025

Окончание работ: 28.06.2025

Работа выполняется в рамках учебной практики. Финансирование не предусмотрено.

1. **Цели создание АС:**

* разработать программный продукт для численного моделирования методом Монте-Карло;
* автоматизировать вычисления площади сегмента окружности;
* визуализировать процесс моделирования;
* визуализировать процесс анализа;
* освоить практический навык программирования.

Критерии оценки достижения целей создания АС

* корректность численных расчетов (погрешность < 5% при количестве точек > 10 000);
* время расчета: ≤1 сек для N ≤ 10000;
* отображать график, соответствующий результатам моделирования;
* удобство пользовательского интерфейса.

**Назначение АС**

Программа предназначена для поддержки учебного процесса, демонстрации метода Монте-Карло и автоматизации расчетов в учебных и исследовательских целях.

1. **Характеристика объектов автоматизации**

Объект автоматизации – процесс численного моделирования площади сегментов окружности.

**Характеристика окружающей среды**

Эксплуатация в стандартных офисных и учебных помещениях.

**4. Требования к автоматизированной системе**

**4.1. Требования к структуре АС в целом**

4.1.1 Требования к структуре АС в целом

Структура программы:

* заставка. Показывается при запуске АС на краткое время;
* основная форма. На ней происходит математическое моделирование;
* форма анализа моделирования. Содержит графический анализ;
* форма «О программе». Содержит информацию о авторе и программе;
* форма управления экспериментами;
* встроенная справка.

4.1.2 Компоненты взаимодействуют через общий модуль данных.

4.1.3 Требования к характеристикам взаимосвязей создаваемой АС со смежными АС не предусмотрено.

4.1.4Требование к режимам функционирования АС:

* интерактивный режим с визуализацией.

4.1.5. Требования по диагностированию АС

Обработка исключительных ситуаций:

* все исключения должны быть перехвачены. Отображение стеков трассировки пользователю недопустимо, следует показывать дружелюбные сообщения;
* каждое критическое событие должно фиксироваться в отдельном журнале («error.log») с указанием причины и времени возникновения.

4.1.6 Перспективы развития, модернизации АС:

* добавление поддержки других геометрических фигур;
* возможность задания произвольной прямой;
* многоязычная поддержка интерфейса;
* добавление динамического округления в зависимости от количества точек;
* добавление других способов анализ данных.

**4.2 Требования к функциям (задачам), выполняемым АС**

4.2.1 Перечень функций:

* прием и проверка исходных данных;
* аналитический расчет площади;
* численное моделирование методом Монте-Карло;
* графическое отображение процесса моделирования;
* сохранение результатов;
* графическое отображение анализа;
* проведение серии моделирований для анализа погрешности;
* управление результатами моделирований;
* логирование операций удаления и записи в базу данных и генерации данных;
* отображение справки.

4.2.2 Результаты выполнения функций:

* вывод значения площади сегмента;
* визуализация результатов;
* сохраненные файлы результатов;
* файлы логирования.

**4.3 Требования к видам обеспечения АС**

4.3.1 Требования к математическому обеспечению АС

Формулы расчета площади сегмента:

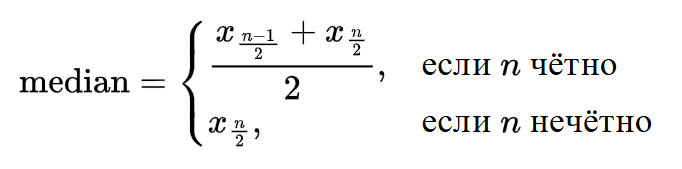
* формула площади круга: S = πR2
* формула площади квадрата, описывающего круг Sквадрата = 4R2
* формула площади сегмента: S = R²arccos(d/R) - d√(R²-d²), где d = |C - x0| (или y0)
* формула для метода Монте-Карло: S ≈ (Nвнутри/Nвсего) \* Sквадрата
* алгоритм генерации случайных точек: координата точки по оси X или Y это случайное число с плавающей точкой от 0 до 1, умноженное на удвоенный радиус минус радиус. Получается координата случайной точки в области [-R, R).

Методы оценки точности:

* абсолютная погрешность ∆X = Xд – X, где Xд – действительное значение, а X – измеренное значение;
* относительная погрешность δ = (|∆X| / Xд) \* 100%, где Xд – действительное значение, а ∆X – абсолютная погрешность измерений.

Методы статистического анализа:

* **медиана**, значение, разделяющее упорядоченную выборку на две равные части.



Где x0≤x1≤…≤ xn-1 - упорядоченная выборка, n - размер выборки.

* **мода** наиболее часто встречающееся значение в выборке.
* **дисперсия** среднее арифметическое квадратов отклонений от среднего значения.



Где x0≤x1≤…≤ xn-1 - упорядоченная выборка, n - размер выборки.

* **стандартное отклонение** корень квадратный из дисперсии.
* **размах (Range)** разность между максимальным и минимальным значениями выборки. Range = max(values) − min(values), где values – выборка.

4.3.2 Требования к информационному обеспечению:

* ввод исходных данных пользователем;
* хранение и сохранение результатов в базе данных SQLite.

4.3.3 Требования к лингвистическому обеспечению

Интерфейс – русский язык.

4.3.4 Требования к программному обеспечению

* язык программирования – C# 13;
* среда выполнения – .NET 9.0;
* операционная система – Windows 7/10/11;
* библиотеки – Math и Microsoft.EntityFrameworkCore.SQLite.

4.3.5 Требования к техническому обеспечению:

* ПК с процессором не ниже Intel Core i3;
* ОЗУ не менее 512 МБ;
* клавиатура;
* мышь;
* цветной монитор 13 дюймов;
* место на диске 1 ГБ.

4.3.6 Требования к метрологическому обеспечению

* оценка погрешности < 5%, при количестве точек ≥ 10 000;
* округление абсолютной и относительной погрешности до двух значащих цифр.

4.3.7 Требования к организационному обеспечению не требуются.

4.3.8 Использовать методические материалы по практике дисциплины:

* ГОСТ 34.602-2020;
* ГОСТ 19.701-90;
* ГОСТ Р 7.0.80-2023;
* ГОСТ 7.32-2017;
* ГОСТ Р 7.05.-2008.

**4.4 Общие технические требования АС**

4.4.1 Требования к численности и квалификации персонала и пользователя АС

Пользователь – студент или преподаватель. Навыки: базовые математической теории вероятности и умение пользоваться операционной системой Windows.

4.4.2 Требования к показателям назначения:

* автоматизация расчета
* демонстрация метода Монте-Карло.

4.4.3 Требования к надежности:

* устойчивость к ошибкам;
* обработка недопустимых значений.

4.4.4 Требования по безопасности не требуют специальных мер.

4.4.5 Требования к эргономике и технической эстетике

* простой и понятный интерфейс;
* предупреждение для эпилептиков если есть моргающие элементы;
* графическая визуализация результатов;
* подписи всех элементов управления;
* цветовая дифференциация элементов на графике.

4.4.6 Требования к транспортабельности для подвижных АСне применимо.

4.4.7 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов АС не требуются.

4.4.9 Требования по сохранности информации при авариях:

Создание резервной копии пользователем и возможность восстановиться с неё.

4.4.10 Требования к защите от влияния внешних воздействий не требуются.

4.4.11 Требования к патентной чистоте и патентоспособности не требуются.

4.4.12 Требования по стандартизации и унификации

Использование стандартных и сторонних библиотек C#.

4.4.13 Дополнительные требования не предусмотрены.

5. **Состав и содержание работ по созданию автоматизированной системы**

* анализ требований к разработке ТЗ (09.06.2025);
* разработка ТЗ к разработке (10.06.2025);
* разработка вычислений методом Монте-Карло (11.06.2025 – 12.06.2025);
* разработка пользовательского интерфейса (13.06.2025 – 14.06.2025);
* разработка формы анализа (15.06.2025);
* подготовка автоматических тестов (16.06.2025);
* подготовка ручных тестов (17.06.2025);
* проведение тестирования (18.06.2025 – 20.06.2025);
* подготовка пояснительной записки (21.06.2025 – 26.06.2025);
* подготовка к приему разработанной АС (27.06.2025);
* прием разработанной АС (28.06.2025).

6. **Порядок разработки автоматизированной системы**

Разработка АС осуществляется по следующим этапам с учётом выбранной программно-технической платформы:

6.1 Формулировка цели и задач разработки

* определяются цели автоматизации, состав и назначение задач, решаемых системой;
* анализируется предметная область;
* оценивается эффективность внедрения.

6.2. Выбор среды и средств разработки

В качестве основной платформы выбрана:

* ОС: Windows 11;
* язык программирования: C# 13;
* среда выполнения: .NET 9.0 – windows;
* фреймворк: Windows Forms;
* математическая библиотека: Math.NET;
* работа с базой данных через Microsoft.EntityFrameworkCore;
* среда разработки: Visual Studio 2022 или новее.

6.3. Разработка математического и алгоритмического обеспечения

* построение математических моделей процессов;
* использование стандартной библиотеки Math для реализации алгоритмов, включая:
* статистические расчёты;
* метод Монте-Карло;

6.4. Разработка программного обеспечения и пользовательского интерфейса

* реализация графического интерфейса с использованием Windows Forms;
* реализация функций сохранения данных в базу данных SQLite.

6.5. Проведение испытаний программного обеспечения

* модульное тестирование классов и алгоритмов с использованием MS Test;
* интеграционные тесты взаимодействия между модулями с использованием MS Test;
* испытания соответствуют этапам: предварительные, приёмочные, опытная эксплуатация.

6.6. Оформление документации и подготовка к вводу в эксплуатацию:

* подготовка комплекта документации;
* создание установщика;
* подготовка системы к сдаче: финальное тестирование, резервное копирование, выдача документации заказчику.

**7. Порядок контроля и приемки автоматизированной системы**

* проверка ручным тестированием пользовательский интерфейс;
* автоматизированное тестирование математических расчетов;
* проверка корректности реагирование программы на ошибки;
* проверка графического отображения результатов;
* демонстрация работы программы на нескольких наборах исходных данных.

**8. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу автоматизированной системы в действие**

* OC Windows 7/10/11;
* установка .NET v9;
* установка приложения;
* проверка корректной работы программы на целевом оборудовании.

Установка приложения происходит следующим образом:

1. Подготовка системы. Установите .NET 9 Runtime (требует прав администратора) и скачайте архив Monte-Karlo.rar с официального источника;
2. Распаковка. Скопируйте архив в целевую директорию (например, C:\Program Files\Monte-Karlo\) и распакуйте архив с сохранением структуры папок (используйте WinRAR/7-Zip);
3. Запуск программы. Откройте распакованную папку и запустите Monte-Karlo.exe. При первом запуске автоматически создадутся DataBase.db, хранилище экспериментов, и папка Logs\ с журналами ошибок в формате YYYY-MM-DD.log.

**9. Требования к документированию**

* инструкция пользователя;
* встроенная справка по работе с программой;
* форма «О программе» с указанием автора и версии;
* код программы.

**10. Источники разработки**

* задание на учебную практику;
* ГОСТ 34.602-2020;

# Приложение Б код приложения

Файл Program.cs

// Класс, с точкой запуска приложения

namespace Monte\_Karlo

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

// Защита от дублирования запуска приложения

if (System.Diagnostics.Process.GetProcessesByName(System.Diagnostics.Process.GetCurrentProcess().ProcessName).Length > 1)

{

MessageBox.Show(

"Программа уже запущена.",

"Оповещение",

MessageBoxButtons.OK,

MessageBoxIcon.Information

);

return;

}

Application.Run(new SplashScreenForm());

}

}

}

Файл SplashScreenForm.cs

// Модуль SplashScreenForm реализует форму заставки (splash screen) приложения.

// Заставка автоматически закрывается через заданное время или по нажатию кнопки,

// после чего открывает главную форму приложения.

namespace Monte\_Karlo

{

public partial class SplashScreenForm : Form

{

private int \_time = 0;

private int \_timeout = 3; // время до автоматического закрытия заставки

// Инициализирует форму заставки и настраивает прозрачность метки автора

public SplashScreenForm()

{

InitializeComponent();

SetTransparentColor(authorLabel, pictureBox1);

}

// Устанавливает прозрачный фон для элемента относительно родителя

public void SetTransparentColor(Control control, Control parent)

{

if (parent == control)

return;

control.Parent = parent;

control.BackColor = Color.Transparent;

}

// Обрабатывает нажатие кнопки, открывая главную форму и скрывая заставку

private void startButton\_Click(object sender, EventArgs e)

{

var form = new MainForm();

form.WindowState = FormWindowState.Normal;

form.Show();

this.Hide();

}

// Запускает таймер при загрузке формы заставки

private void Screensaver\_Load(object sender, EventArgs e)

{

timer1.Start();

}

// Обрабатывает тик таймера, автоматически закрывая заставку по истечении времени

private void timer1\_Tick(object sender, EventArgs e)

{

if (++\_time >= \_timeout)

{

timer1.Stop();

startButton.PerformClick();

}

}

}

}

Файл MainForm.cs

// Основная форма программы для ввода параметров расчёта площади сегмента круга

// методом Монте-Карло и визуализации результатов

using Monte\_Karlo.DataBase;

using Monte\_Karlo.Forms;

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

using Monte\_Karlo.Utilites.View;

using Monte\_Karlo.Utilites;

using System.Diagnostics;

using System.Reflection;

namespace Monte\_Karlo

{

public partial class MainForm : Form

{

private float cofficient = 2, divisionScale = 0.5f;

private Circle circle = new Circle();

private int pointsCount = 100\_000;

private CancellationTokenSource \_generationCts;

private PointsGenerator \_pointsGenerator;

private MonteCarloView \_view;

private DatabaseHelper \_databaseHelper;

private Logger \_logger;

// Инициализирует главную форму и компоненты

public MainForm()

{

InitializeComponent();

\_pointsGenerator = new PointsGenerator();

\_databaseHelper = new DatabaseHelper();

\_view = new MonteCarloView();

\_logger = new Logger();

DoubleBuffered = true;

typeof(Panel).InvokeMember("DoubleBuffered",

BindingFlags.SetProperty | BindingFlags.Instance | BindingFlags.NonPublic,

null, paintPanel, new object[] { true });

InitializeControlPanel();

\_databaseHelper.InitializeDatabase();

\_logger.Log("Приложение запущено");

}

// Настраивает начальные значения элементов управления

private void InitializeControlPanel()

{

xNumericUpDown.Value = circle.circleCenter.X;

yNumericUpDown.Value = circle.circleCenter.Y;

radiusTrackBar.Value = (int)circle.radius;

radiusLabel.Text = $"Радиус круга: {radiusTrackBar.Value}";

SetCTrackBarBorders();

scaleTrackBar.Value = \_view.GridStep;

scaleLabel.Text = $"Масштаб: {scaleTrackBar.Value}";

cTrackBar.Value = Convert.ToInt32(circle.C \* cofficient);

cLabel.Text = $"Значение C: {circle.C}";

pointsCountUpdown.Value = pointsCount;

}

// Выполняет расчёт методом Монте-Карло

private async Task MonteCarloCalculate(bool generateNewPoints)

{

if (this.Visible != true)

return;

\_generationCts?.Cancel();

\_generationCts = new CancellationTokenSource();

try

{

var token = \_generationCts.Token;

// Генерировать новые точки или переобработать старые

if (generateNewPoints)

{

await \_pointsGenerator.GenerateRandomPointsAsync(circle, pointsCount, token);

}

else

{

await \_pointsGenerator.CalculateCuttedPointsAsync(circle, pointsCount, token);

}

if (token.IsCancellationRequested)

return;

// Переотрисовка

paintPanel.Invalidate();

// Расчёты

double realSquare = Calculator.CalculateAnalyticArea(circle);

var roundedRealSquare = Math.Round(realSquare, 4);

var currentPoints = \_pointsGenerator.GetCurrentPoints();

double monteCarloSquare = Calculator.CalculateMonteCarloArea(

circle.radius,

currentPoints.Points.Count,

currentPoints.CuttedPoints.Count);

var roundedMonteCarloSquare = Math.Round(monteCarloSquare, 4);

// Вывод результатов

ShowAnswereMessage(realSquare, monteCarloSquare, currentPoints);

\_logger.Log($"Сделаны расчёты с параметрами: {circle.ToString()} и количеством точек {pointsCount}");

WriteResultOnLabels(roundedRealSquare, roundedMonteCarloSquare);

// Сохранение в БД

\_databaseHelper.SaveResults(

circle,

currentPoints,

realSquare,

monteCarloSquare);

}

catch (OperationCanceledException)

{

// Игнорируем отмену

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Выводит результаты расчётов на метки формы

private void WriteResultOnLabels(double realSquare, double monteCarloSquare)

{

realSquareLabel.Text = $"Площадь секции аналитически: {realSquare:F4}";

monteCarloSquareLabel.Text = $"Площадь методом Монте-Карло: {monteCarloSquare:F4}";

}

// Отображает подробные результаты расчётов в сообщении

private void ShowAnswereMessage(double realSquare, double monteCarloSquare, PointsData currentPoints)

{

if (!showMessageCheckBox.Checked)

return;

double absoluteError = Calculator.CalculateAbsoluteError(realSquare, monteCarloSquare);

double relativeError = Calculator.CalculateRelativeError(realSquare, monteCarloSquare);

double roundAbsoluteError = Math.Round(absoluteError, 4);

double roundRelativeError = Math.Round(relativeError, 4);

double maxAccuracy = 1 / (double)pointsCount;

string message = $"""

Площадь круга: {Calculator.CircleSuare(circle.radius):F4}

Всего точек: {currentPoints.Points.Count}

Количество точек попавших в круг {currentPoints.IncludedPoints.Count}

Количество точек в большей секции: {currentPoints.CuttedPoints.Count}

Площадь секции аналитически: {realSquare:F4}

Площадь секции методом Монте-Карло: {monteCarloSquare:F4}

Абсолютаня погрешность вычислений: {roundAbsoluteError}

Относительная погрешность вычислений: {roundRelativeError}%

Максимальная точность при заданном количестве точек: {maxAccuracy}

""";

MessageBox.Show(message, "Результат вычислений");

}

// Отображает информацию об исключении

private void ShowException(Exception ex, string message = "")

{

MessageBox.Show($"{message} {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

\_logger.LogException(ex, message);

}

// Устанавливает границы для трекбара параметра C

private void SetCTrackBarBorders()

{

int border = Convert.ToInt32(circle.radius \* cofficient);

int min, max;

if (circle.direction == Direction.vertical)

{

min = -border + circle.circleCenter.X \* (int)cofficient;

max = border + circle.circleCenter.X \* (int)cofficient;

}

else

{

min = -border + circle.circleCenter.Y \* (int)cofficient;

max = border + circle.circleCenter.Y \* (int)cofficient;

}

circle.C = Math.Clamp(circle.C \* cofficient, min, max) \* divisionScale;

cTrackBar.Minimum = min;

cTrackBar.Maximum = max;

cLabel.Text = $"Значение C: {circle.C}";

}

#region Код ответственный за интерактивное взаимодействие на пенели управления

// Обрабатывает изменение координаты X центра круга

private async void xNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

circle.circleCenter.X = (int)Math.Floor(xNumericUpDown.Value);

SetCTrackBarBorders();

await MonteCarloCalculate(false);

}

// Обрабатывает изменение координаты Y центра круга

private async void yNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

circle.circleCenter.Y = (int)Math.Floor(yNumericUpDown.Value);

SetCTrackBarBorders();

await MonteCarloCalculate(false);

}

// Обрабатывает изменение радиуса круга

private async void radiusSlider\_Scroll(object sender, EventArgs e)

{

radiusLabel.Text = $"Радиус круга: {radiusTrackBar.Value}";

circle.radius = (float)radiusTrackBar.Value;

SetCTrackBarBorders();

await MonteCarloCalculate(true);

}

// Запускает генерацию точек и расчёт по методу Монте-Карло

private async void btnGeneratePoints\_Click(object sender, EventArgs e)

{

await MonteCarloCalculate(true);

}

// Очищает сгенерированные точки

private void btnClear\_Click(object sender, EventArgs e)

{

try

{

\_pointsGenerator.ClearPoints();

paintPanel.Invalidate();

WriteResultOnLabels(0, 0);

\_logger.Log("Очищение точек");

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Обрабатывает изменение количества точек для расчёта

private async void pointsCountUpdown\_ValueChanged(object sender, EventArgs e)

{

pointsCount = (int)pointsCountUpdown.Value;

await MonteCarloCalculate(true);

}

// Обрабатывает изменение параметра C

private async void cTrackbar\_ValueChanged(object sender, EventArgs e)

{

circle.C = cTrackBar.Value \* divisionScale;

cLabel.Text = $"Значение C: {circle.C}";

await MonteCarloCalculate(false);

}

// Обрабатывает изменение масштаба отображения

private void scaleTrackbar\_Scroll(object sender, EventArgs e)

{

scaleLabel.Text = $"Масштаб: {scaleTrackBar.Value}";

\_view.GridStep = scaleTrackBar.Value;

paintPanel.Invalidate();

}

// Обрабатывает изменение направления сегмента (горизонтальное/вертикальное)

private async void horizontalCheckBox\_CheckedChanged(object sender, EventArgs e)

{

if (circle.direction == Direction.horizontal)

circle.direction = Direction.vertical;

else

circle.direction = Direction.horizontal;

SetCTrackBarBorders();

await MonteCarloCalculate(false);

}

#endregion

#region Код ответственный за верхнее меню

// Открывает файл справки

private void programHelpToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

string helpFile = Path.Combine(Application.StartupPath, "Help", "index.htm");

// Открываем справку в браузере по умолчанию

Process.Start(new ProcessStartInfo

{

FileName = helpFile,

UseShellExecute = true

});

}

catch (Exception ex)

{

ShowException(ex, $"Не удалось открыть справку");

}

}

// Открывает форму "О программе"

private void aboutProgramToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var form = new AboutProgramForm();

form.ShowDialog();

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Открывает форму анализа результатов для текущих параметров

private void analysisOfResultsToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var circleParam = \_databaseHelper.GetData(circle, pointsCount);

var form = new AnalysisForm(circleParam);

form.ShowDialog();

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Открывает форму управления данными экспериментов

private void dataManagementToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var form = new DataManagementForm();

form.ShowDialog();

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Закрывает приложение

private void closeProgramToolStripMenuItem\_Click(object sender, EventArgs e) => this.Close();

#endregion

// Перерисовывает панель при изменении размера

private void paintPanel\_Resize(object sender, EventArgs e)

{

paintPanel.Invalidate();

}

// Обрабатывает закрытие формы

private void MainForm\_FormClosed(object sender, FormClosedEventArgs e)

{

\_generationCts?.Cancel();

\_logger.Log("Приложение закрыто");

Application.Exit();

}

// Обрабатывает событие отрисовки панели с графикой

private void paintPanel\_Paint(object sender, PaintEventArgs e)

{

try

{

\_view.RenderToBuffer(

paintPanel,

e,

circle,

\_pointsGenerator.GetCurrentPoints()

);

}

catch (Exception ex)

{

MessageBox.Show("Произошла ошибка из-за частой переотрисовки графика.\n" +

"Пожалуйста, дайте время на переотрисовку графика",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

Thread.Sleep(100);

\_view.RenderToBuffer(

paintPanel,

e,

circle,

\_pointsGenerator.GetCurrentPoints()

);

}

base.OnPaint(e);

}

}

}

Файл DataManagementForm.cs

// Форма для управления данными экспериментов: просмотр, резервное копирование,

// удаление данных и генерация дополнительных экспериментов

using Monte\_Karlo.DataBase;

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

using Monte\_Karlo.Utilites;

using System.ComponentModel;

using Timer = System.Windows.Forms.Timer;

namespace Monte\_Karlo.Forms

{

public partial class DataManagementForm : Form

{

private DatabaseHelper databaseHelper = new DatabaseHelper();

private Logger logger = new Logger();

private CancellationTokenSource \_cts;

// Инициализирует форму управления данными

public DataManagementForm()

{

InitializeComponent();

}

// Загружает список экспериментов при открытии формы

private void DataManagementForm\_Load(object sender, EventArgs e)

{

LoadExperiments();

}

// Загружает список экспериментов из базы данных в DataGridView

private void LoadExperiments()

{

try

{

dgvExperiments.Columns.Clear();

var data = databaseHelper.GetAllData();

var experiments = data

.Select(cp => new

{

ID = cp.Id,

Центр\_X = cp.CenterX,

Центр\_Y = cp.CenterY,

Радиус = cp.Radius,

Направление = cp.Direction == Direction.horizontal ?

"Горизонтально" :

"Вертикально",

Параметр\_C = cp.C,

Всего\_точек = cp.TotalPoints,

Аналитический\_результат = cp.AnalyticalResult,

Количество\_экспериментов = cp.Results.Count

})

.ToList();

var bindingSource = new BindingSource() { DataSource = experiments };

dgvExperiments.DataSource = bindingSource;

dgvExperiments.Columns["ID"].Visible = false; // Скрываем ID

lblStatus.Text = $"Загружено экспериментов: {experiments.Count}";

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

logger.LogException(ex, "Ошибка загрузки данных");

}

}

// Создает резервную копию базы данных

private void btnBackup\_Click(object sender, EventArgs e)

{

using (var saveDialog = new SaveFileDialog())

{

saveDialog.Filter = "SQLite база данных|\*.db";

saveDialog.Title = "Создание резервной копии";

saveDialog.FileName = $"MonteCarlo\_Backup\_{DateTime.Now:yyyyMMdd\_HHmmss}.db";

if (saveDialog.ShowDialog() == DialogResult.OK)

{

try

{

databaseHelper.CreateBackup(saveDialog.FileName);

lblStatus.Text = $"Резервная копия создана: {Path.GetFileName(saveDialog.FileName)}";

MessageBox.Show("Резервное копирование выполнено успешно!", "Успех",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка резервного копирования: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

logger.LogException(ex, "Ошибка резервного копирования");

}

}

}

}

// Удаляет все эксперименты из базы данных

private void btnClearAll\_Click(object sender, EventArgs e)

{

if (MessageBox.Show("Вы уверены, что хотите удалить ВСЕ данные экспериментов? Это действие нельзя отменить.",

"Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)

{

try

{

databaseHelper.ClearDatabase();

LoadExperiments();

lblStatus.Text = "Все данные экспериментов удалены";

MessageBox.Show("Все данные успешно удалены.", "Успех",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка удаления данных: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

logger.LogException(ex, "Ошибка удаления данных");

}

}

}

// Удаляет выбранный эксперимент из базы данных

private void btnClearSelected\_Click(object sender, EventArgs e)

{

if (dgvExperiments.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите эксперимент для удаления", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

var selectedId = (int)dgvExperiments.SelectedRows[0].Cells["ID"].Value;

var centerX = dgvExperiments.SelectedRows[0].Cells["Центр\_X"].Value;

var centerY = dgvExperiments.SelectedRows[0].Cells["Центр\_Y"].Value;

if (MessageBox.Show($"Вы уверены, что хотите удалить все данные для эксперимента с центром ({centerX}, {centerY})?",

"Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)

{

try

{

databaseHelper.RemoveCircleParamsById(selectedId);

LoadExperiments();

lblStatus.Text = "Выбранный эксперимент удалён";

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка удаления эксперимента: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

logger.LogException(ex, "Ошибка удаления эксперимента");

}

}

}

// Открывает форму анализа для выбранного эксперимента

private void btnanalysisOfResults\_Click(object sender, EventArgs e)

{

if (dgvExperiments.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите эксперимент для анализа", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

var selectedId = (int)dgvExperiments.SelectedRows[0].Cells["ID"].Value;

var circleParam = databaseHelper.GetDataById(selectedId);

var form = new AnalysisForm(circleParam);

form.ShowDialog();

}

// Сортирует данные по клику на заголовок колонки

private void dgvExperiments\_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)

{

if (dgvExperiments.RowCount == 0)

return;

DataGridViewColumn column = dgvExperiments.Columns[e.ColumnIndex];

// Определяем направление сортировки

ListSortDirection direction = column.HeaderCell.SortGlyphDirection == SortOrder.Ascending ?

ListSortDirection.Descending :

ListSortDirection.Ascending;

// Сортируем данные

SortData(column.Name, direction);

// Обновляем иконку сортировки

dgvExperiments.Columns.Cast<DataGridViewColumn>()

.ToList()

.ForEach(c => c.HeaderCell.SortGlyphDirection = SortOrder.None);

dgvExperiments.Columns[e.ColumnIndex].HeaderCell.SortGlyphDirection = direction == ListSortDirection.Ascending ?

SortOrder.Ascending :

SortOrder.Descending;

}

// Выполняет сортировку данных по указанной колонке

private void SortData(string columnName, ListSortDirection direction)

{

if (dgvExperiments.DataSource is BindingSource bindingSource)

{

var data = bindingSource.List.Cast<dynamic>().ToList();

switch (columnName)

{

case "ID":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.ID).ToList() :

data.OrderByDescending(x => x.ID).ToList();

break;

case "Центр\_X":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Центр\_X).ToList() :

data.OrderByDescending(x => x.Центр\_X).ToList();

break;

case "Центр\_Y":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Центр\_Y).ToList() :

data.OrderByDescending(x => x.Центр\_Y).ToList();

break;

case "Радиус":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Радиус).ToList() :

data.OrderByDescending(x => x.Радиус).ToList();

break;

case "Направление":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Направление).ToList() :

data.OrderByDescending(x => x.Направление).ToList();

break;

case "Параметр\_C":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Параметр\_C).ToList() :

data.OrderByDescending(x => x.Параметр\_C).ToList();

break;

case "Всего\_точек":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Всего\_точек).ToList() :

data.OrderByDescending(x => x.Всего\_точек).ToList();

break;

case "Аналитический\_результат":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Аналитический\_результат).ToList() :

data.OrderByDescending(x => x.Аналитический\_результат).ToList();

break;

case "Количество\_экспериментов":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Количество\_экспериментов).ToList() :

data.OrderByDescending(x => x.Количество\_экспериментов).ToList();

break;

default:

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.ID).ToList() :

data.OrderByDescending(x => x.ID).ToList();

break;

}

}

}

// Генерирует 1000 экспериментов для выбранной конфигурации

private async void btn1000Experiments\_Click(object sender, EventArgs e)

{

// Прерывание операции генерации экспериментов

if (btn1000Experiments.Text == "Прервать")

{

\_cts?.Cancel();

return;

}

if (dgvExperiments.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите эксперимент для генерации", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

string originalButtonText = btn1000Experiments.Text;

btn1000Experiments.Text = "Прервать";

try

{

\_cts = new CancellationTokenSource();

var token = \_cts.Token;

var selectedId = (int)dgvExperiments.SelectedRows[0].Cells["ID"].Value;

var circleParam = databaseHelper.GetDataById(selectedId);

var point = new Point((int)circleParam.CenterX, (int)circleParam.CenterY);

var circle = new Circle(point, (float)circleParam.Radius, circleParam.Direction, (float)circleParam.C);

int pointsCount = circleParam.TotalPoints;

var pointsGenerator = new PointsGenerator();

// Отображение прогресса генерации экспериментов

IProgress<string> progress = new Progress<string>(s => lblStatus.Text = s);

using Timer timer = new Timer();

timer.Interval = 50;

int i = 0;

timer.Tick += (object? sender, EventArgs e) =>

{

progress.Report($"Генерация эксперимента {i + 1}/1000...");

};

timer.Start();

// Цикл с генерацией и сохранением результата

for (i = 0; i < 1000; i++)

{

token.ThrowIfCancellationRequested();

await pointsGenerator.GenerateRandomPointsAsync(circle, pointsCount, token);

var currentPoints = pointsGenerator.GetCurrentPoints();

var realSquare = Calculator.CalculateAnalyticArea(circle);

var monteCarloSquare = Calculator.CalculateMonteCarloArea(

circle.radius,

currentPoints.Points.Count,

currentPoints.CuttedPoints.Count);

await Task.Run(() =>

databaseHelper.SaveResults(

circle,

currentPoints,

realSquare,

monteCarloSquare));

}

progress.Report($"Генерация эксперимента {i}/1000...");

timer.Stop();

MessageBox.Show("1000 экспериментов успешно сгенерированы!", "Готово");

}

catch (OperationCanceledException)

{

MessageBox.Show("Операция генерации прервана", "Оповещение");

}

finally

{

btn1000Experiments.Text = originalButtonText;

lblStatus.Text = "Готово";

LoadExperiments();

\_cts?.Dispose();

\_cts = null;

}

}

// Обеспечивает корректное завершение операций при закрытии формы

private void DataManagementForm\_FormClosed(object sender, FormClosedEventArgs e)

{

\_cts?.Cancel();

\_cts?.Dispose();

\_cts = null;

}

}

}

Файл AnalysisForm.cs

// Форма для отображения графического и статистического анализа результатов

// метода Монте-Карло для вычисления площади сегмента круга

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

using Monte\_Karlo.Utilites.View;

using System.ComponentModel;

namespace Monte\_Karlo

{

public partial class AnalysisForm : Form

{

private List<SimulationResult> \_results = new List<SimulationResult>();

private CircleParams \_currentParams;

private AnalysisView \_view;

// Инициализирует форму анализа с пустыми параметрами

public AnalysisForm()

{

InitializeComponent();

\_view = new AnalysisView();

}

// Инициализирует форму анализа с заданными параметрами круга

public AnalysisForm(CircleParams circleParams) : this()

{

if (circleParams is not null)

{

\_currentParams = circleParams;

\_results = circleParams.Results;

}

}

// Загружает данные и рассчитывает статистику при открытии формы

private void AnalysisForm\_Load(object sender, EventArgs e)

{

SetupDataGridView();

CalculateStatistics();

}

// Настраивает DataGridView для отображения результатов

private void SetupDataGridView()

{

dataGridViewResults.Columns.Clear();

AddColumn("Id", "№", "Id", true, "D2", null, DataGridViewAutoSizeColumnMode.DisplayedCells);

AddColumn("PointsInCircle", "Точек в окружности", "PointsInCircle", true, "N0");

AddColumn("PointsInSegment", "Точек в сегменте", "PointsInSegment", true, "N0");

AddColumn("AnalyticalResult", "Аналитический резльтат", "AnalyticalResult", true, "F4",

\_currentParams?.AnalyticalResult.ToString("F4") ?? "N/A");

AddColumn("MonteCarloResult", "Результат Монте-Карло", "MonteCarloResult", true, "F4");

AddColumn("AbsoluteError", "Абсолютная ошибка", "AbsoluteError", true, "F2");

AddColumn("RelativeError", "Ошибка (%)", "RelativeError", true, "F2");

if (\_currentParams != null && \_currentParams.Results.Any())

{

int id = 0;

var displayResults = \_currentParams.Results

.OrderByDescending(r => r.Id)

.Select(r => new

{

Id = ++id,

r.PointsInCircle,

r.PointsInSegment,

AnalyticalResult = \_currentParams.AnalyticalResult,

r.MonteCarloResult,

AbsoluteError = Calculator.CalculateAbsoluteError(\_currentParams.AnalyticalResult, r.MonteCarloResult).ToString(),

RelativeError = Calculator.CalculateRelativeError(\_currentParams.AnalyticalResult, r.MonteCarloResult).ToString()

})

.ToList();

var bindingSource = new BindingSource() { DataSource = displayResults };

dataGridViewResults.DataSource = bindingSource;

}

else

{

dataGridViewResults.DataSource = null;

}

}

// Применяет цветовое форматирование для ячеек с относительной ошибкой

private void DataGridViewResults\_CellFormatting(object? sender, DataGridViewCellFormattingEventArgs e)

{

{

if (e.ColumnIndex == dataGridViewResults.Columns["RelativeError"].Index && e.Value != null)

{

double error = Convert.ToDouble(e.Value);

e.CellStyle.ForeColor = error switch

{

> 10 => Color.Red,

> 5 => Color.Orange,

\_ => Color.Green

};

}

};

}

// Добавляет колонку в DataGridView с заданными параметрами

private void AddColumn(string name, string header, string dataPropertyName, bool isReadOnly, string format,

object defaultValue = null, DataGridViewAutoSizeColumnMode AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells)

{

var col = new DataGridViewTextBoxColumn

{

Name = name,

HeaderText = header,

DataPropertyName = dataPropertyName,

ReadOnly = isReadOnly,

DefaultCellStyle = new DataGridViewCellStyle

{

Format = format,

Alignment = DataGridViewContentAlignment.MiddleCenter,

},

SortMode = DataGridViewColumnSortMode.Programmatic,

AutoSizeMode = AutoSizeMode

};

if (defaultValue != null)

{

col.DefaultCellStyle.NullValue = defaultValue;

}

dataGridViewResults.Columns.Add(col);

}

// Обрабатывает клик по заголовку колонки для сортировки данных

private void DataGridViewResults\_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)

{

DataGridViewColumn column = dataGridViewResults.Columns[e.ColumnIndex];

// Определяем направление сортировки

ListSortDirection direction = column.HeaderCell.SortGlyphDirection == SortOrder.Ascending ?

ListSortDirection.Descending :

ListSortDirection.Ascending;

// Сортируем данные

SortData(column.Name, direction);

// Обновляем иконку сортировки

dataGridViewResults.Columns.Cast<DataGridViewColumn>()

.ToList()

.ForEach(c => c.HeaderCell.SortGlyphDirection = SortOrder.None);

dataGridViewResults.Columns[e.ColumnIndex].HeaderCell.SortGlyphDirection = direction == ListSortDirection.Ascending ?

SortOrder.Ascending :

SortOrder.Descending;

}

// Сортирует данные в DataGridView по указанной колонке

private void SortData(string columnName, ListSortDirection direction)

{

if (dataGridViewResults.DataSource is BindingSource bindingSource)

{

var data = bindingSource.List.Cast<dynamic>().ToList();

switch (columnName)

{

case "PointsInCircle":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.PointsInCircle).ToList() :

data.OrderByDescending(x => x.PointsInCircle).ToList();

break;

case "PointsInSegment":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.PointsInSegment).ToList() :

data.OrderByDescending(x => x.PointsInSegment).ToList();

break;

case "MonteCarloResult":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.MonteCarloResult).ToList() :

data.OrderByDescending(x => x.MonteCarloResult).ToList();

break;

case "AnalyticalResult":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.AnalyticalResult).ToList() :

data.OrderByDescending(x => x.AnalyticalResult).ToList();

break;

case "AbsoluteError":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => Convert.ToDouble(x.AbsoluteError)).ToList() :

data.OrderByDescending(x => Convert.ToDouble(x.AbsoluteError)).ToList();

break;

case "RelativeError":

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => Convert.ToDouble(x.RelativeError)).ToList() :

data.OrderByDescending(x => Convert.ToDouble(x.RelativeError)).ToList();

break;

default:

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

data.OrderBy(x => x.Id).ToList() :

data.OrderByDescending(x => x.Id).ToList();

break;

}

}

}

// Рассчитывает и отображает статистические показатели

private void CalculateStatistics()

{

if (\_results == null || \_results.Count == 0)

return;

var mcResults = \_results.Select(r => r.MonteCarloResult).ToList();

double variance = StatisticCalculator.CalculateVariance(mcResults);

variance = Calculator.RoundToTwoSignificantDigits(variance, 2);

double stdDev = StatisticCalculator.CalculateStandardDeviation(variance);

stdDev = Calculator.RoundToTwoSignificantDigits(stdDev, 2);

lblAnalisicResult.Text = \_currentParams.AnalyticalResult.ToString("F4");

lblMean.Text = mcResults.Average().ToString("F4");

lblMedian.Text = StatisticCalculator.CalculateMedian(mcResults).ToString("F4");

lblMode.Text = StatisticCalculator.CalculateMode(mcResults).ToString("F4");

lblVariance.Text = variance.ToString();

lblStdDev.Text = stdDev.ToString();

lblMinimum.Text = mcResults.Min().ToString("F4");

lblMaximum.Text = mcResults.Max().ToString("F4");

lblRange.Text = StatisticCalculator.CalculateRange(mcResults).ToString("F4");

}

// Отрисовывает графическое представление анализа

private void paintPanel\_Paint(object sender, PaintEventArgs e)

{

\_view.RenderAnalysis(paintPanel, e, \_currentParams);

base.OnPaint(e);

}

}

}

Файл AboutProgramForm.cs

// Форма "О программе" - отображает информацию о версии приложения

// и предоставляет ссылку на GitHub разработчика

using System.Diagnostics;

using System.Reflection;

namespace Monte\_Karlo

{

public partial class AboutProgramForm : Form

{

// Инициализирует форму и устанавливает текущую версию приложения

public AboutProgramForm()

{

InitializeComponent();

var version = Assembly.GetExecutingAssembly().GetName().Version;

versionLabel.Text = $"Версия: {version.Major}.{version.Minor}.{version.Build}.{version.MinorRevision}";

}

// Закрывает форму при нажатии на кнопку

private void closeButton\_Click(object sender, EventArgs e)

{

this.Close();

}

// Открывает ссылку на GitHub разработчика в браузере по умолчанию

private void githubLinkLabel\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

try

{

// Открываем ссылку в браузере по умолчанию

Process.Start(new ProcessStartInfo

{

FileName = "https://github.com/AndreiGudz",

UseShellExecute = true

});

}

catch (Exception ex)

{

MessageBox.Show($"Не удалось открыть ссылку: {ex.Message}",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

Файл AppDbContext.cs

// Класс, описывающий базу данных, подключение к ней, таблицы и их структуру

using Microsoft.EntityFrameworkCore;

using Monte\_Karlo.Models;

namespace Monte\_Karlo.DataBase

{

public class AppDbContext : DbContext

{

// Таблица с параметрами окружности, для которой проводят эксперементы

public DbSet<CircleParams> CircleParams { get; set; }

// Таблица с результатами эксперементов

public DbSet<SimulationResult> SimulationResults { get; set; }

private string databasePath = "DataBase.db";

// Указание для EF Core, что будет использоваться SQLite

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite($"Data Source={databasePath}");

//optionsBuilder.LogTo(message => System.Diagnostics.Debug.WriteLine(message));

}

// Создание индекса для CircleParams

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<CircleParams>()

.HasIndex(cp => new

{

cp.CenterX,

cp.CenterY,

cp.Radius,

cp.Direction,

cp.C,

cp.TotalPoints

})

.IsUnique();

}

}

}

Файл DatabaseHelper.cs

// Класс, ответственный за взаимодействие остального приложения с базой данных

using Microsoft.EntityFrameworkCore;

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites;

namespace Monte\_Karlo.DataBase

{

public class DatabaseHelper

{

private Logger logger;

// Инициализирует помощник базы данных и проверяет подключение

public DatabaseHelper()

{

logger = new Logger();

InitializeDatabase();

}

// Инициализирует базу данных, если она еще не создана

public void InitializeDatabase()

{

using var context = new AppDbContext();

if (context.Database.CanConnect())

return;

context.Database.EnsureCreated();

logger.Log("Создание базы данных");

}

// Полностью очищает базу данных (удаляет и создает заново)

public void ClearDatabase()

{

using (var context = new AppDbContext())

{

context.Database.EnsureDeleted();

context.Database.EnsureCreated();

}

logger.Log("Очистка базы данных");

}

// Сохраняет результаты расчета в базу данных

public void SaveResults(Circle circle, PointsData pointsData, double analyticalResult, double monteCarloResult)

{

using var context = new AppDbContext();

int totalPoints = pointsData.Points.Count;

int pointsInCircle = pointsData.IncludedPoints.Count;

int pointsInSegment = pointsData.CuttedPoints.Count;

// Поиск существующей записи с такими параметрами

var circleParams = context.CircleParams

.Include(cp => cp.Results)

.FirstOrDefault(cp =>

cp.CenterX == circle.circleCenter.X &&

cp.CenterY == circle.circleCenter.Y &&

cp.Radius == circle.radius &&

cp.Direction == circle.direction &&

cp.C == circle.C &&

cp.TotalPoints == totalPoints);

// Создание новой записи, если не найдена существующая

if (circleParams == null)

{

circleParams = new CircleParams

{

CenterX = circle.circleCenter.X,

CenterY = circle.circleCenter.Y,

Radius = circle.radius,

Direction = circle.direction,

C = circle.C,

TotalPoints = totalPoints,

AnalyticalResult = analyticalResult

};

context.CircleParams.Add(circleParams);

}

// Создание записи с результатами расчета

var result = new SimulationResult

{

CircleParams = circleParams,

PointsInCircle = pointsInCircle,

PointsInSegment = pointsInSegment,

MonteCarloResult = monteCarloResult

};

context.SimulationResults.Add(result);

context.SaveChanges();

logger.Log($"Сохранение:\n{circleParams}\n{result}");

}

// Получает параметры круга и результаты по заданным параметрам

public CircleParams GetData(Circle circle, int totalPoints)

{

using var context = new AppDbContext();

var query = context.CircleParams

.Include(cp => cp.Results)

.Where(cp =>

cp.CenterX == circle.circleCenter.X &&

cp.CenterY == circle.circleCenter.Y &&

cp.Radius == circle.radius &&

cp.Direction == circle.direction &&

cp.C == circle.C &&

cp.TotalPoints == totalPoints);

return query.FirstOrDefault();

}

// Получает параметры круга и результаты по ID

public CircleParams GetDataById(int selectedId)

{

using var context = new AppDbContext();

var result = context.CircleParams

.Include(cp => cp.Results)

.FirstOrDefault(cp => cp.Id == selectedId);

return result;

}

// Получает все данные экспериментов из базы

public List<CircleParams> GetAllData()

{

using var context = new AppDbContext();

var results = context.CircleParams

.Include(cp => cp.Results)

.ToList();

return results;

}

// Удаляет эксперимент по его ID

public void RemoveCircleParamsById(int selectedId)

{

using var context = new AppDbContext();

var experiment = context.CircleParams

.Include(cp => cp.Results)

.FirstOrDefault(cp => cp.Id == selectedId);

if (experiment != null)

{

context.CircleParams.Remove(experiment);

context.SaveChanges();

}

logger.Log($"Удаление:\n{experiment}");

}

// Создает резервную копию базы данных

public string CreateBackup(string fileName)

{

var currentDirectory = Directory.GetCurrentDirectory();

var sourcePath = Path.Combine(currentDirectory, "DataBase.db");

File.Copy(sourcePath, fileName, true);

string message = $"Резервная копия создана: {Path.GetFileName(fileName)}";

logger.Log(message);

return Path.GetFileName(fileName);

}

}

}

Файл PointsData.cs

// Класс для хранения сгенерированных и отфильтрованных точек

namespace Monte\_Karlo.Models

{

public class PointsData

{

public List<PointF> Points { get; set; } = new();

public List<PointF> IncludedPoints { get; set; } = new();

public List<PointF> CuttedPoints { get; set; } = new();

}

}

Файл Direction.cs

// Перечисление с возможными ориентациями линии

namespace Monte\_Karlo.Models

{

public enum Direction

{

horizontal,

vertical

}

}

Файл CircleParams.cs

// Таблица с параметрами окружности, для которой проводят эксперементы

using System.ComponentModel.DataAnnotations;

namespace Monte\_Karlo.Models

{

public class CircleParams

{

[Key]

public int Id { get; set; }

public double CenterX { get; set; }

public double CenterY { get; set; }

public double Radius { get; set; }

public Direction Direction { get; set; }

public double C { get; set; }

public int TotalPoints { get; set; }

public double AnalyticalResult { get; set; }

// Связь с результатами эксперементов с такими параметрами

public List<SimulationResult> Results { get; set; } = new List<SimulationResult>();

// Вывод всех параметров в стоку

public override string ToString()

{

return $"""

Id: {Id}, CenterX: {CenterX}, CenterY: {CenterY}, Radius: {Radius},

Direction: {Direction}, C: {C}, TotalPoints: {TotalPoints},

AnalyticalResult: {AnalyticalResult}, ResultsCount: {Results.Count}

""";

}

}

}

Файл Circle.cs

// Класс для хранения параметров окружности и линии

namespace Monte\_Karlo.Models

{

public class Circle

{

public Point circleCenter = new Point(3, 1);

public float radius = 2;

public Direction direction = Direction.horizontal;

public float C = 2;

// Конструктор без параметров для создания окружности и линии с значениями по умолчанию

public Circle() { }

// Конструктор для задания всех своих параметров

public Circle(Point circleCenter, float radius, Direction direction, float c)

{

this.circleCenter = circleCenter;

if (radius <= 0)

throw new ArgumentException($"R <= 0: R = {radius}");

this.radius = radius;

this.direction = direction;

this.C = c;

}

// Сравнение на равенство для EF

public override bool Equals(object obj)

{

return obj is Circle other &&

circleCenter.X == other.circleCenter.X &&

circleCenter.Y == other.circleCenter.Y &&

radius == other.radius &&

direction == other.direction &&

C == other.C;

}

// Генерация хэша для EF

public override int GetHashCode()

{

return HashCode.Combine(circleCenter.X, circleCenter.Y, radius, direction, C);

}

// Вывод всех параметров в стоку

public override string ToString()

{

return $"CircleCenter: {circleCenter}, Radius: {radius}, Direction: {direction}, C: {C}";

}

}

}

Файл SimulationResult.cs

// Таблица с результатами эксперементов

using System.ComponentModel.DataAnnotations;

namespace Monte\_Karlo.Models

{

public class SimulationResult

{

[Key]

public int Id { get; set; }

public int PointsInCircle { get; set; }

public int PointsInSegment { get; set; }

public double MonteCarloResult { get; set; }

// Связь по внешнему ключу с таблицей параметров круга и линии, для которой проводился эксперемент

public int CircleParamsId { get; set; }

public CircleParams CircleParams { get; set; }

// Вывод всех параметров в стоку

public override string ToString()

{

return $"""

Id: {Id}, PointsInCircle: {PointsInCircle}, PointsInSegment: {PointsInSegment},

MonteCarloResult: {MonteCarloResult}, CircleParamsId: {CircleParamsId}

""";

}

}

}

Файл MonteCarloView.cs

// Класс для визуализации графика Монте-Карло на основной форме

// Отрисовывает фигуры и точки в большем сегменте на панели

using Monte\_Karlo.Models;

namespace Monte\_Karlo.Utilites.View

{

public class MonteCarloView

{

// Шаг сетки (в пикселях)

public int GridStep

{

get => \_gridStep;

set

{

\_gridStep = value;

\_step = \_gridStep \* 2;

}

}

private int \_gridStep = 40;

private int \_step = 80;

// Цвета и кисти для отрисовки

private static readonly Color \_backgroundColor = Color.White;

private static readonly Pen \_gridPen = new(Color.LightGray, 1);

private static readonly Pen \_axisPen = new(Color.Black, 2);

private static readonly Pen \_cutterPen = new(Color.Red, 4);

private static readonly Pen \_circlePen = new(Color.Red, 2);

private static readonly Pen \_squarePen = new(Color.Red, 2);

private static readonly Pen \_cuttedPointsBrush = new(Color.FromArgb(174, 206, 180), 1);

private static readonly Color \_textColor = Color.Black;

private static readonly Brush \_textBrush = new SolidBrush(\_textColor);

private static readonly Font \_textFont = new("Arial", 8);

// Основной метод рендеринга на буфер панели

public void RenderToBuffer(Panel panel, PaintEventArgs e, Circle circle, PointsData pointsData)

{

e.Graphics.Clear(\_backgroundColor);

OnPaint(panel, e, circle.radius, circle.circleCenter, circle.direction, circle.C, pointsData);

}

// Обрабатывает событие отрисовки, координирует вызов всех методов рисования

private void OnPaint(Panel panel, PaintEventArgs e, float radius, Point circleCenter, Direction direction, float C, PointsData pointsData)

{

var g = e.Graphics;

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighSpeed;

// Вычисляем координаты центра и углов

float centerX = panel.Size.Width / 2;

float centerY = panel.Size.Height / 2;

var centerScreen = new PointF(centerX, centerY);

float squareX = centerX - radius \* \_step;

float squareY = centerY - radius \* \_step;

var squarePoint = new PointF(squareX, squareY);

float originX = centerX - circleCenter.X \* \_step;

float originY = centerY + circleCenter.Y \* \_step;

var origin = new PointF(originX, originY);

// Последовательность отрисовки элементов

DrawPoints(g, centerScreen, \_step, pointsData);

DrawGrid(panel, g, origin);

DrawAxis(panel, g, origin);

DrawCoordinateNumbers(panel, g, origin);

DrawRectangle(g, squarePoint, \_step \* radius \* 2);

DrawEllipse(g, squarePoint, \_step \* radius \* 2);

DrawCutter(panel, g, origin, direction, C);

}

// Рисует координатную сетку

private void DrawGrid(Panel panel, Graphics g, PointF origin)

{

// Вертикальные линии

for (float x = origin.X; x >= 0; x -= \_gridStep)

{

g.DrawLine(\_gridPen, x, 0, x, panel.Height);

}

for (float x = origin.X; x <= panel.Width; x += \_gridStep)

{

g.DrawLine(\_gridPen, x, 0, x, panel.Height);

}

// Горизонтальные линии

for (float y = origin.Y; y >= 0; y -= \_gridStep)

{

g.DrawLine(\_gridPen, 0, y, panel.Width, y);

}

for (float y = origin.Y; y <= panel.Height; y += \_gridStep)

{

g.DrawLine(\_gridPen, 0, y, panel.Width, y);

}

}

// Рисует оси координат

private void DrawAxis(Panel panel, Graphics g, PointF center)

{

g.DrawLine(\_axisPen, 0, center.Y, panel.Width, center.Y);

g.DrawLine(\_axisPen, center.X, 0, center.X, panel.Height);

}

// Рисует числовые подписи на осях координат

private void DrawCoordinateNumbers(Panel panel, Graphics g, PointF origin)

{

// Числа на оси X (влево)

for (float x = origin.X; x >= 0; x -= \_step)

{

int digit = (int)Math.Round((x - origin.X) / \_step);

if (digit == 0)

continue;

string text = digit.ToString();

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = x - textSize.Width / 2;

float textY = origin.Y + 5;

if (TextInPanel(panel, textSize, textX, textY))

{

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

}

// Числа на оси X (вправо)

for (float x = origin.X; x <= panel.Width; x += \_step)

{

int digit = (int)Math.Round((x - origin.X) / \_step);

if (digit == 0)

continue;

string text = digit.ToString();

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = x - textSize.Width / 2;

float textY = origin.Y + 5;

if (TextInPanel(panel, textSize, textX, textY))

{

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

}

// Числа на оси Y (вверх)

for (float y = origin.Y; y >= 0; y -= \_step)

{

int digit = -(int)Math.Round((y - origin.Y) / \_step);

if (digit == 0)

continue;

string text = digit.ToString();

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = origin.X + 5;

float textY = y - textSize.Height / 2;

if (TextInPanel(panel, textSize, textX, textY))

{

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

}

// Числа на оси Y (вниз)

for (float y = origin.Y; y <= panel.Height; y += \_step)

{

int digit = -(int)Math.Round((y - origin.Y) / \_step);

if (digit == 0)

continue;

string text = digit.ToString();

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = origin.X + 5;

float textY = y - textSize.Height / 2;

if (TextInPanel(panel, textSize, textX, textY))

{

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

}

// Рисуем ноль в начале координат

g.DrawString("0", \_textFont, \_textBrush, origin.X + 5, origin.Y + 5);

}

// Проверяет, помещается ли текст в границы панели

private bool TextInPanel(Panel panel, SizeF textSize, float textX = 0, float textY = 0)

{

bool xIn = textX >= 0 && textX + textSize.Width <= panel.Width;

bool yIn = textY >= 0 && textY + textSize.Height <= panel.Height;

return xIn && yIn;

}

// Рисует ограничивающий квадрат

private void DrawRectangle(Graphics g, PointF square, float squareSize)

{

g.DrawRectangle(\_squarePen, square.X, square.Y, squareSize, squareSize);

}

// Рисует окружность

private void DrawEllipse(Graphics g, PointF square, float squareSize)

{

g.DrawEllipse(\_circlePen, square.X, square.Y, squareSize, squareSize);

}

// Рисует линию разреза (сектор)

private void DrawCutter(Panel panel, Graphics g, PointF center, Direction direction, float C)

{

if (direction == Direction.horizontal)

g.DrawLine(\_cutterPen, 0, center.Y + \_step \* -C, panel.Width, center.Y + \_step \* -C);

else

g.DrawLine(\_cutterPen, center.X + \_step \* C, 0, center.X + \_step \* C, panel.Height);

}

// Рисует точки Монте-Карло (с ограничением количества для производительности)

private void DrawPoints(Graphics g, PointF center, float gridStep, PointsData pointsData)

{

if (pointsData.CuttedPoints.Count == 0)

return;

// Ограничение количества отображаемых точек

const int ViewPointsLimit = 100\_000;

int pointsToDraw = Math.Min(pointsData.CuttedPoints.Count, ViewPointsLimit);

var rectangles = new RectangleF[pointsToDraw];

// Преобразуем координаты точек в экранные координаты

for (int i = 0; i < pointsToDraw; i++)

{

var point = pointsData.CuttedPoints[i];

float screenX = center.X + point.X \* gridStep;

float screenY = center.Y - point.Y \* gridStep;

rectangles[i] = new RectangleF(screenX, screenY, 1, 1);

}

// Массовая отрисовка точек

g.DrawRectangles(\_cuttedPointsBrush, rectangles);

}

}

}

Файл AnalysisView.cs

// Класс для визуализации результатов анализа метода Монте-Карло

// Отрисовывает графики, точки и статистические показатели на панели

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

namespace Monte\_Karlo.Utilites.View

{

public class AnalysisView

{

// Цвета элементов

private static readonly Color \_analyticalColor = Color.Blue;

private static readonly Color \_pointsColor = Color.Green;

private static readonly Color \_meanColor = Color.Red;

private static readonly Color \_medianColor = Color.Purple;

private static readonly Color \_minMaxColor = Color.Orange;

private static readonly Color \_backgroundColor = Color.White;

private static readonly Color \_gridColor = Color.LightGray;

private static readonly Padding \_padding = new Padding(50, 20, 70, 40);

private static readonly double \_percentYPadding = 0.1;

private static readonly System.Drawing.Font \_textFont = SystemFonts.DefaultFont;

private static readonly Brush \_textBrush = Brushes.Black;

private static readonly float \_pointRadius = 4;

// Основной метод отрисовки анализа на панели

public void RenderAnalysis(Panel panel, PaintEventArgs e, CircleParams circleParams)

{

var g = e.Graphics;

g.Clear(\_backgroundColor);

if (circleParams == null || circleParams.Results == null || circleParams.Results.Count == 0)

{

MessageBox.Show("Нет результатов для анализа", "Оповещение");

return;

}

OnPaint(panel, g, circleParams);

}

// Обрабатывает событие отрисовки, вычисляет статистику и вызывает методы рисования

private void OnPaint(Panel panel, Graphics g, CircleParams circleParams)

{

List<double> mcResults = circleParams.Results.Select(r => r.MonteCarloResult).ToList();

double analyticalValue = circleParams.AnalyticalResult;

double mean = mcResults.Average();

double median = StatisticCalculator.CalculateMedian(mcResults);

double min = mcResults.Min();

double max = mcResults.Max();

Rectangle plotArea = new Rectangle(\_padding.Left, \_padding.Top, panel.Width - \_padding.Right, panel.Height - \_padding.Bottom);

double yMin = Math.Min(analyticalValue, min);

double yMax = Math.Max(analyticalValue, max);

double yRange = yMax - yMin;

yMin -= yRange \* \_percentYPadding;

yMax += yRange \* \_percentYPadding;

yRange = yMax - yMin;

DrawGrid(g, plotArea, mcResults.Count, yMin, yMax);

DrawAnalyticalLine(g, plotArea, analyticalValue, yMin, yRange);

DrawMonteCarloPoints(g, plotArea, mcResults, yMin, yRange);

DrawMeanLine(g, plotArea, mean, yMin, yRange);

DrawMedianLine(g, plotArea, median, yMin, yRange);

DrawMinMaxLines(g, plotArea, min, max, yMin, yRange);

DrawLegend(g, plotArea, median);

}

// Рисует координатную сетку с подписями осей

private void DrawGrid(Graphics g, Rectangle plotArea, int pointsCount, double yMin, double yMax)

{

Pen girdPen = new Pen(\_gridColor);

g.DrawRectangle(girdPen, plotArea);

// Вертикальные линии (каждые 10% экспериментов)

int step = Math.Max(1, pointsCount / 10);

for (int i = 0; i <= pointsCount; i += step)

{

float x = plotArea.Left + plotArea.Width \* i / pointsCount;

g.DrawLine(girdPen, x, plotArea.Top, x, plotArea.Bottom);

// Подписи номеров экспериментов

string text = i.ToString();

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = x - textSize.Width / 2;

float textY = plotArea.Bottom - textSize.Height;

// особое расположение для 0

if (i == 0)

textX += textSize.Width;

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

// Горизонтальные линии сетки (7 линий)

int linesCount = 7 - 1;

for (int i = 0; i <= linesCount; i++)

{

float y = plotArea.Top + plotArea.Height \* i / linesCount;

g.DrawLine(girdPen, plotArea.Left, y, plotArea.Right, y);

// Подписи значений

double yRange = yMax - yMin;

double value = yMax - yRange \* i / linesCount;

string text = value.ToString("F2");

SizeF textSize = g.MeasureString(text, \_textFont);

float textX = plotArea.Left - textSize.Width;

float textY = y - textSize.Height / 2;

g.DrawString(text, \_textFont, \_textBrush, textX, textY);

}

}

// Рисует линию аналитического решения

private void DrawAnalyticalLine(Graphics g, Rectangle area, double value, double yMin, double yRange)

{

float y = area.Bottom - (float)((value - yMin) / yRange \* area.Height);

g.DrawLine(new Pen(\_analyticalColor, 4), area.Left, y, area.Right, y);

}

// Рисует точки результатов метода Монте-Карло

private void DrawMonteCarloPoints(Graphics g, Rectangle area, List<double> results, double yMin, double yRange)

{

if (results == null || results.Count == 0)

return;

int count = results.Count;

float width = area.Width;

float height = area.Height;

float left = area.Left;

float bottom = area.Bottom;

// Предварительно вычисляем часто используемые значения

float xStep = width / (count - 1);

float yScale = height / (float)yRange;

// Используем один экземпляр кисти для всех точек

using (SolidBrush brush = new SolidBrush(\_pointsColor))

{

float diameter = 2 \* \_pointRadius;

for (int i = 0; i < count; i++)

{

try

{

float x = left + i \* xStep;

float y = bottom - (float)((results[i] - yMin) \* yScale);

g.FillEllipse(brush, x - \_pointRadius, y - \_pointRadius, diameter, diameter);

}

catch (DivideByZeroException ex)

{

MessageBox.Show("Слишком мало данных измерений (минимум 2)", "Оповещение");

return; // Прерываем выполнение после ошибки

}

}

}

}

// Рисует линию среднего значения

private void DrawMeanLine(Graphics g, Rectangle area, double value, double yMin, double yRange)

{

float y = area.Bottom - (float)((value - yMin) / yRange \* area.Height);

g.DrawLine(new Pen(\_meanColor, 3) { DashStyle = System.Drawing.Drawing2D.DashStyle.Dash },

area.Left, y, area.Right, y);

}

// Рисует линию медианного значения

private void DrawMedianLine(Graphics g, Rectangle area, double value, double yMin, double yRange)

{

float y = area.Bottom - (float)((value - yMin) / yRange \* area.Height);

g.DrawLine(new Pen(\_medianColor, 3) { DashStyle = System.Drawing.Drawing2D.DashStyle.Dot },

area.Left, y, area.Right, y);

}

// Рисует линии минимального и максимального значений

private void DrawMinMaxLines(Graphics g, Rectangle area, double min, double max, double yMin, double yRange)

{

float yMinPos = area.Bottom - (float)((min - yMin) / yRange \* area.Height);

float yMaxPos = area.Bottom - (float)((max - yMin) / yRange \* area.Height);

g.DrawLine(new Pen(\_minMaxColor, 3), area.Left, yMinPos, area.Right, yMinPos);

g.DrawLine(new Pen(\_minMaxColor, 3), area.Left, yMaxPos, area.Right, yMaxPos);

}

// Рисует легенду графика

private void DrawLegend(Graphics g, Rectangle area, double mode)

{

SizeF textSize = g.MeasureString("Аналитическое решение", \_textFont);

float boxWidth = 20;

float startX = area.Width - textSize.Width - boxWidth - 5;

float startY = area.Top;

float itemHeight = textSize.Height;

DrawLegendItem(g, "Аналитическое решение", \_analyticalColor, startX, startY, boxWidth, itemHeight);

DrawLegendItem(g, "Точки Монте-Карло", \_pointsColor, startX, startY + itemHeight, boxWidth, itemHeight);

DrawLegendItem(g, "Среднее значение", \_meanColor, startX, startY + itemHeight \* 2, boxWidth, itemHeight);

DrawLegendItem(g, "Медиана", \_medianColor, startX, startY + itemHeight \* 3, boxWidth, itemHeight);

DrawLegendItem(g, "Минимум/Максимум", \_minMaxColor, startX, startY + itemHeight \* 4, boxWidth, itemHeight);

}

// Рисует один элемент легенды (цветной прямоугольник + текст)

private void DrawLegendItem(Graphics g, string text, Color color, float x, float y, float boxWidth, float boxHeight)

{

SizeF textSize = g.MeasureString("Аналитическое решение", \_textFont);

g.FillRectangle(new SolidBrush(color), x, y + 1, boxWidth, boxHeight - 2);

g.DrawRectangle(Pens.Black, x, y + 1, boxWidth, boxHeight - 2);

g.DrawString(text, \_textFont, \_textBrush, x + boxWidth + 5, y);

}

}

}

Файл PointsGenerator.cs

// Класс для генерации случайных точек и вычисления их принадлежности кругу и сегменту

using Monte\_Karlo.Models;

using System.Collections.Concurrent;

namespace Monte\_Karlo.Utilites

{

public class PointsGenerator

{

private readonly Mutex \_mutex = new(); // Мьютекс для потокобезопасного доступа

private PointsData \_currentPoints = new(); // Текущий набор точек

// Генерирует новые случайные точки и вычисляет их принадлежность

public async Task GenerateRandomPointsAsync(Circle circle, int count, CancellationToken token)

{

try

{

\_mutex.WaitOne(); // Блокировка для потокобезопасности

token.ThrowIfCancellationRequested();

var newPoints = new PointsData();

newPoints.Points = new List<PointF>(count);

float radius = circle.radius;

await Task.Run(() =>

{

var parallelOptions = new ParallelOptions

{

CancellationToken = token,

MaxDegreeOfParallelism = Environment.ProcessorCount

};

// Генерация точек

GeneratePoints(newPoints, count, radius, parallelOptions);

token.ThrowIfCancellationRequested();

// Вычисление точек внутри круга

CalculateIncludedPoints(newPoints, radius, parallelOptions);

token.ThrowIfCancellationRequested();

// Вычисление точек в сегменте

CalculateCuttedPoints(newPoints, circle, parallelOptions);

}, token);

\_currentPoints = newPoints; // Сохранение результатов

}

finally

{

\_mutex.ReleaseMutex(); // Освобождение блокировки

}

}

// Пересчитывает точки в сегменте для существующего набора точек

public async Task CalculateCuttedPointsAsync(Circle circle, int count, CancellationToken token)

{

if (\_currentPoints.Points.Count == 0)

{

await GenerateRandomPointsAsync(circle, count, token);

return;

}

try

{

\_mutex.WaitOne();

token.ThrowIfCancellationRequested();

await Task.Run(() =>

{

var parallelOptions = new ParallelOptions

{

CancellationToken = token,

MaxDegreeOfParallelism = Environment.ProcessorCount

};

CalculateCuttedPoints(\_currentPoints, circle, parallelOptions);

}, token);

}

finally

{

\_mutex.ReleaseMutex();

}

}

// Очищает текущий набор точек

public void ClearPoints()

{

\_currentPoints = new PointsData();

}

// Возвращает текущий набор точек

public PointsData GetCurrentPoints()

{

return \_currentPoints;

}

// Генерирует случайные точки в квадрате [-radius, radius] x [-radius, radius]

private static void GeneratePoints(PointsData pointsData, int count, float radius, ParallelOptions parallelOptions)

{

var random = new ThreadLocal<Random>(() => new Random(Guid.NewGuid().GetHashCode()));

var points = new PointF[count];

Parallel.For(0, count, parallelOptions, i =>

{

float x = (float)random.Value.NextDouble() \* radius \* 2 - radius;

float y = (float)random.Value.NextDouble() \* radius \* 2 - radius;

points[i] = new PointF(x, y);

});

pointsData.Points = points.ToList();

}

// Фильтрует точки, попавшие внутрь круга

private static void CalculateIncludedPoints(PointsData pointsData, float radius, ParallelOptions parallelOptions)

{

float radiusSquared = radius \* radius;

var includedPoints = new ConcurrentBag<PointF>();

Parallel.ForEach(pointsData.Points, parallelOptions, point =>

{

float distanceSquared = point.X \* point.X + point.Y \* point.Y;

if (distanceSquared < radiusSquared)

{

includedPoints.Add(point);

}

});

pointsData.IncludedPoints = includedPoints.ToList();

}

// Фильтрует точки, попавшие в заданный сегмент круга

private static void CalculateCuttedPoints(PointsData pointsData, Circle circle, ParallelOptions parallelOptions)

{

if (pointsData.IncludedPoints.Count == 0)

return;

var cuttedPoints = new ConcurrentBag<PointF>();

Point center = circle.circleCenter;

Direction direction = circle.direction;

float C = circle.C;

if (direction == Direction.vertical)

{

bool lefter = C < center.X;

float centerX = center.X;

Parallel.ForEach(pointsData.IncludedPoints, parallelOptions, point =>

{

bool condition = lefter

? point.X + centerX >= C

: point.X + centerX <= C;

if (condition)

{

cuttedPoints.Add(point);

}

});

}

else // horizontal

{

bool downer = C < center.Y;

float centerY = center.Y;

Parallel.ForEach(pointsData.IncludedPoints, parallelOptions, point =>

{

bool condition = downer

? centerY + point.Y >= C

: centerY + point.Y <= C;

if (condition)

{

cuttedPoints.Add(point);

}

});

}

pointsData.CuttedPoints.Clear();

pointsData.CuttedPoints = cuttedPoints.ToList();

}

}

}

Файл Logger.cs

// Класс ответственный за логирование сообщений и ошибок

namespace Monte\_Karlo.Utilites

{

public class Logger

{

private string logDir = Path.Combine(Application.StartupPath, "Logs");

// Метод для логирования сообщений

public void Log(string message)

{

string logFile = Path.Combine(logDir, DateTime.Now.ToString("yyyy-MM-dd") + ".log");

try

{

if (!Directory.Exists(logDir))

Directory.CreateDirectory(logDir);

string timestamp = DateTime.Now.ToString("HH:mm:ss");

string line = $"[{timestamp}] {message}";

File.AppendAllText(logFile, line + Environment.NewLine);

System.Diagnostics.Debug.WriteLine("[LOG] " + line);

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("[LOG ERROR] " + ex.Message);

LogException(ex, message);

}

}

// Метод для логирования ошибок

public void LogException(Exception exception, string message = "")

{

System.Diagnostics.Debug.WriteLine($"[ERROR] {message} {exception.Message}\n{exception.StackTrace}");

string logFile = Path.Combine(logDir, DateTime.Now.ToString("error") + ".log");

try

{

if (!Directory.Exists(logDir))

Directory.CreateDirectory(logDir);

string timestamp = DateTime.Now.ToString("yyyy-MM-dd - HH:mm:ss");

string line = $"[{timestamp}] {message} {exception.Message}\n{exception.StackTrace}";

File.AppendAllText(logFile, line + Environment.NewLine);

System.Diagnostics.Debug.WriteLine("[LOG] " + line);

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("[LOG ERROR] " + ex.Message);

}

}

}

}

Файл StatisticCalculator.cs

// Статический класс для вычисление статистических характеристик выборки

namespace Monte\_Karlo.Utilites.Calculators

{

public static class StatisticCalculator

{

// Медиана

public static double CalculateMedian(List<double> values)

{

var sorted = values.OrderBy(x => x).ToList();

int count = sorted.Count;

if (count % 2 == 0)

return (sorted[count / 2 - 1] + sorted[count / 2]) / 2;

else

return sorted[count / 2];

}

// Мода

public static double CalculateMode(List<double> values)

{

return values.GroupBy(x => x)

.OrderByDescending(g => g.Count())

.First()

.Key;

}

// Дисперсия

public static double CalculateVariance(List<double> values)

{

double mean = values.Average();

return values.Average(Xi => Math.Pow(mean - Xi, 2));

}

// Среднее отклонение

public static double CalculateStandardDeviation(double variance)

{

return Math.Sqrt(variance);

}

// Среднее отклонение через список

public static double CalculateStandardDeviation(List<double> values)

{

return CalculateStandardDeviation(CalculateVariance(values));

}

// Размах выборки

public static double CalculateRange(List<double> values) => values.Max() - values.Min();

}

}

Файл Calculator.cs

// Статистический класс для вычисления площади большего сегмента окружности

// Методы для аналитического вычисления площади и методом Монте-Карло

// Также вспомогательные методы с вычислениями

using Monte\_Karlo.Models;

namespace Monte\_Karlo.Utilites.Calculators

{

public static class Calculator

{

// Вычисление площади большего сектора аналитически

public static double CalculateAnalyticArea(Circle circle)

{

Point center = circle.circleCenter;

double R = circle.radius;

Direction direction = circle.direction;

double C = circle.C;

if (R <= 0)

throw new ArgumentException("R <= 0");

if (direction == Direction.horizontal)

{

double yLine = C;

double d = Math.Abs(center.Y - yLine); // расстояние от центра до хорды

double h = Math.Abs(R - d); // расстояние от хорды до окружности

double CircleArea = Math.PI \* R \* R;

if (d >= R)

return CircleArea;

if (h == R)

return CircleArea / 2;

double segmentArea = GetSegmentArea(R, d);

return CircleArea - segmentArea;

}

else

{

double xLine = C;

double d = Math.Abs(center.X - xLine); // расстояние от центра до хорды

double h = Math.Abs(R - d); // расстояние от хорды до окружности

double CircleArea = Math.PI \* R \* R;

if (d >= R)

return CircleArea;

if (h == R)

return CircleArea / 2;

double segmentArea = GetSegmentArea(R, d);

return CircleArea - segmentArea;

}

}

// Вычисление площади сектора окружности

// ссылка на формулу https://en.wikipedia.org/wiki/Circular\_segment

private static double GetSegmentArea(double R, double d)

{

return R \* R \* Math.Acos(d / R) - d \* Math.Sqrt(R \* R - d \* d);

}

// Площадь круга

public static double CircleSuare(double R) => Math.PI \* R \* R;

// Вычисление площади большего сектора методом Монте-Карло

public static double CalculateMonteCarloArea(float radius, int allPoints, int cuttedPoints)

{

double squareArea = 4 \* radius \* radius;

return cuttedPoints / (double)allPoints \* squareArea;

}

// Вычисление абсолютной погрешности

public static double CalculateAbsoluteError(double expectedResult, double actualResult)

{

var result = expectedResult - actualResult;

result = RoundToTwoSignificantDigits(result, 2);

return result;

}

// Вычисление относительной погрешности

public static double CalculateRelativeError(double expectedResult, double actualResult)

{

if (expectedResult <= 0)

throw new ArgumentException("Ожидаемое значение не может быть <= 0");

if (actualResult < 0)

throw new ArgumentException("Полученное значение не может быть < 0");

var result = Math.Abs(CalculateAbsoluteError(expectedResult, actualResult)) / expectedResult \* 100d;

result = RoundToTwoSignificantDigits(result, 2);

return result;

}

// Округление до определённого количества значащих цифр

public static double RoundToTwoSignificantDigits(double value, int significantDigits)

{

if (value == 0.0)

return 0.0;

int log10 = (int)Math.Floor(Math.Log10(Math.Abs(value)));

double scale = Math.Pow(10, significantDigits - log10 - 1);

double rounded = Math.Round(value \* scale) / scale;

return rounded;

}

}

}

# ПРИЛОЖЕНИЕ В окна Программа



Рисунок В1 – Заставка программы

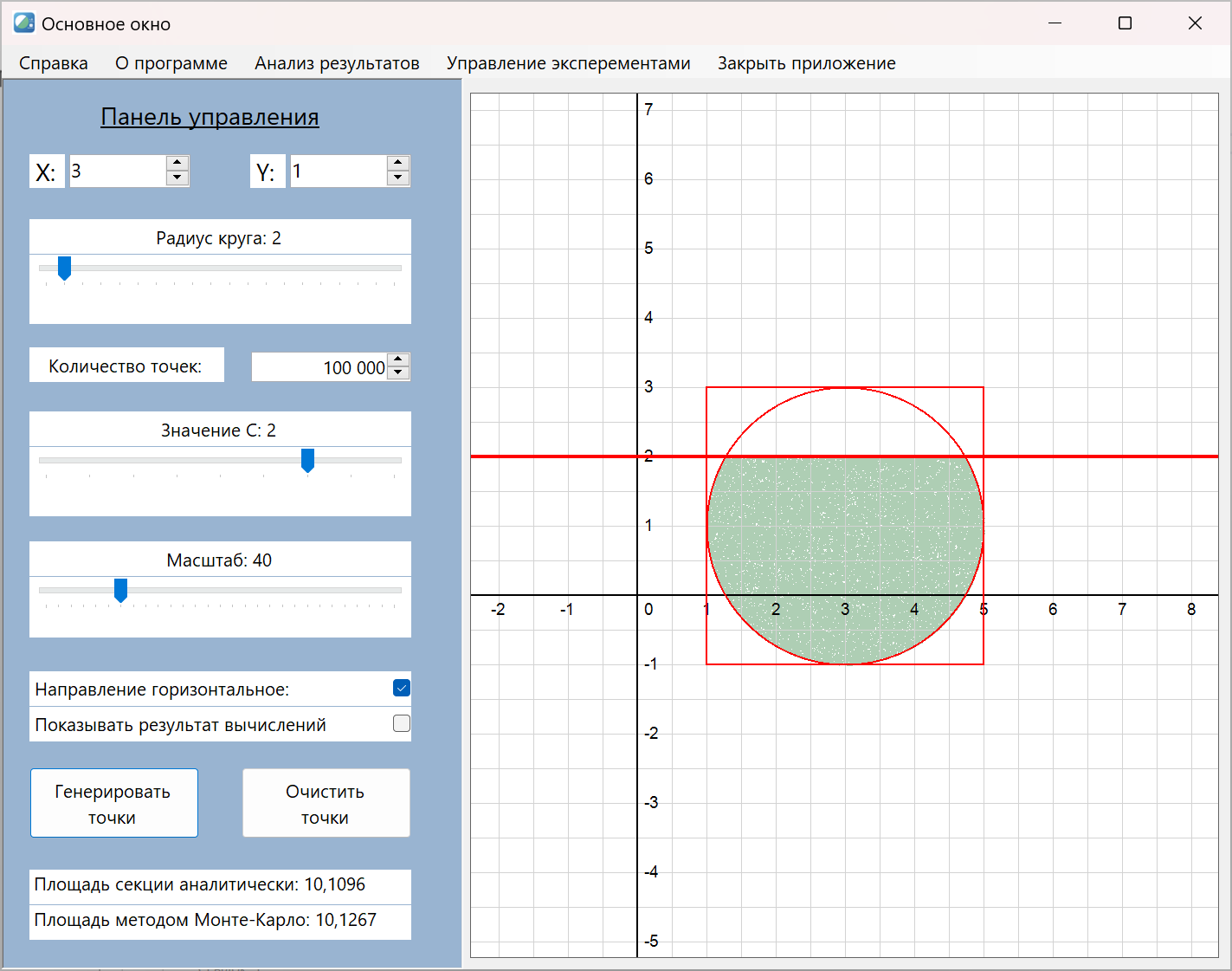


Рисунок В2 – Основное окно программы

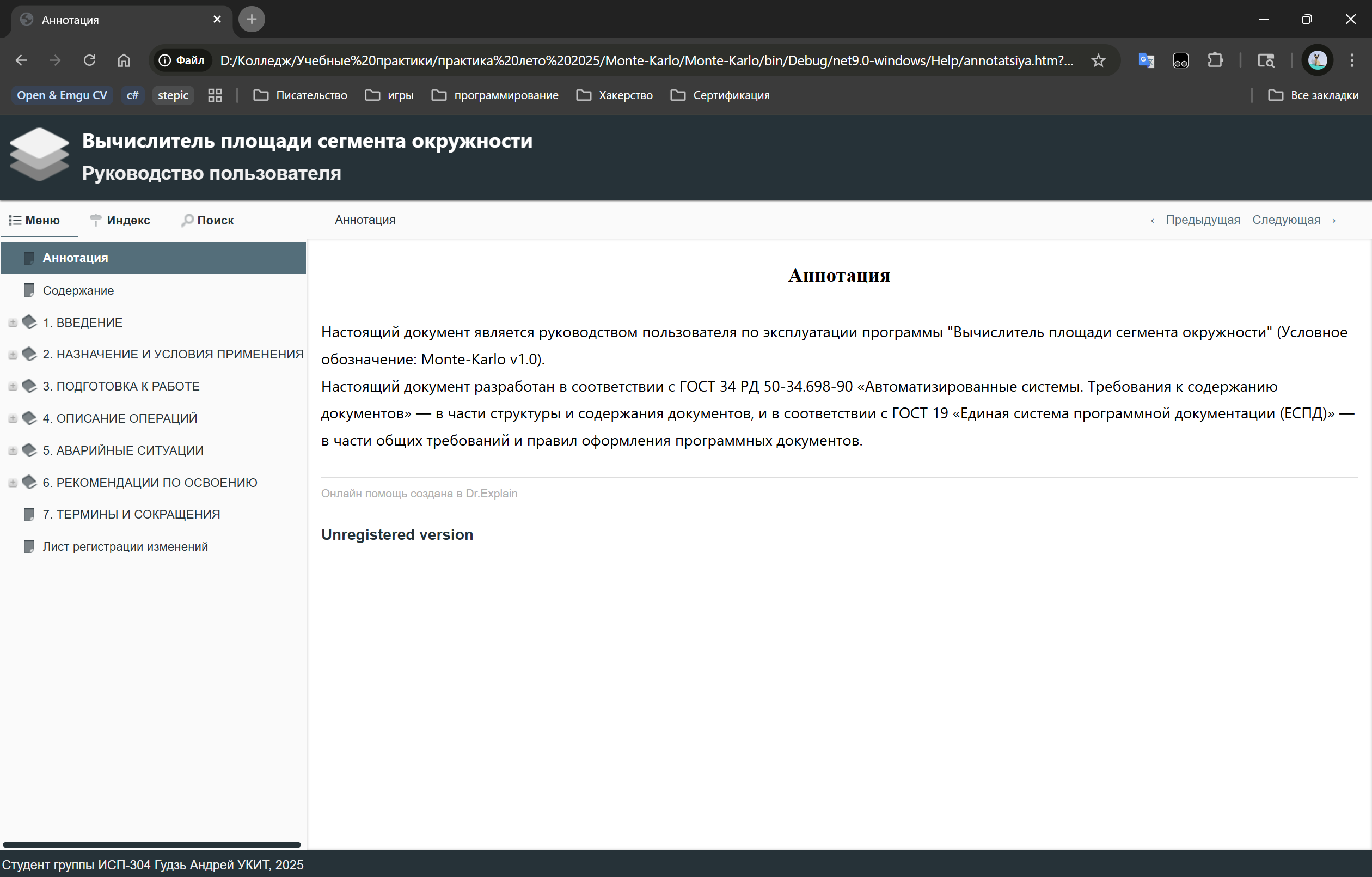


Рисунок В3 – справка, открываемая в браузере

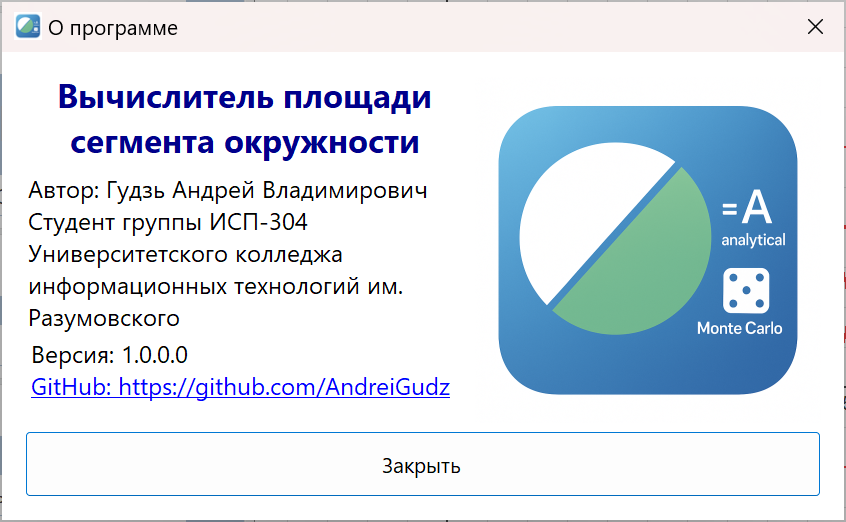


Рисунок В4 – окно «О программе»

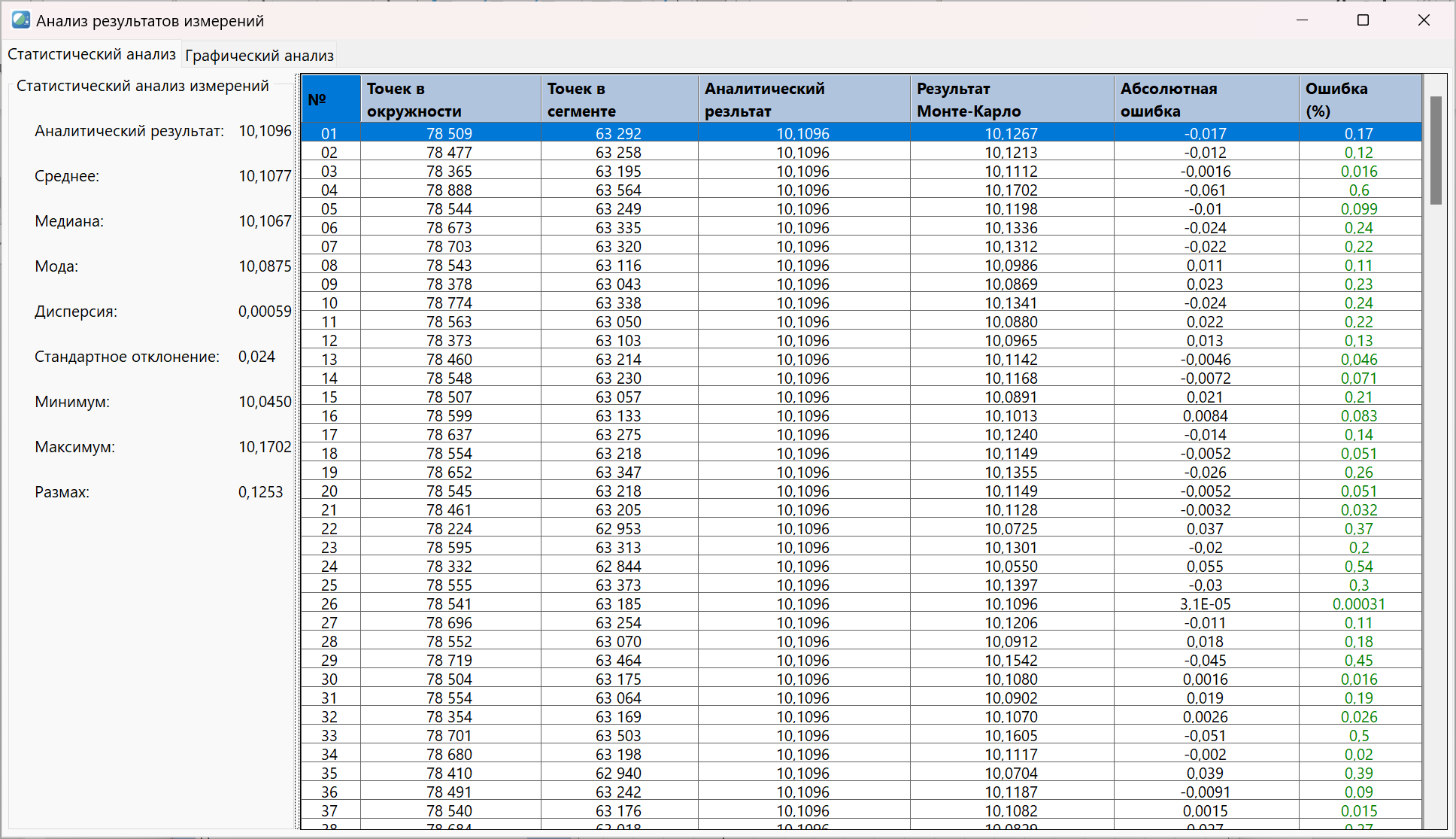


Рисунок В5 – вкладка статистического анализа в окне анализа результатов измерений

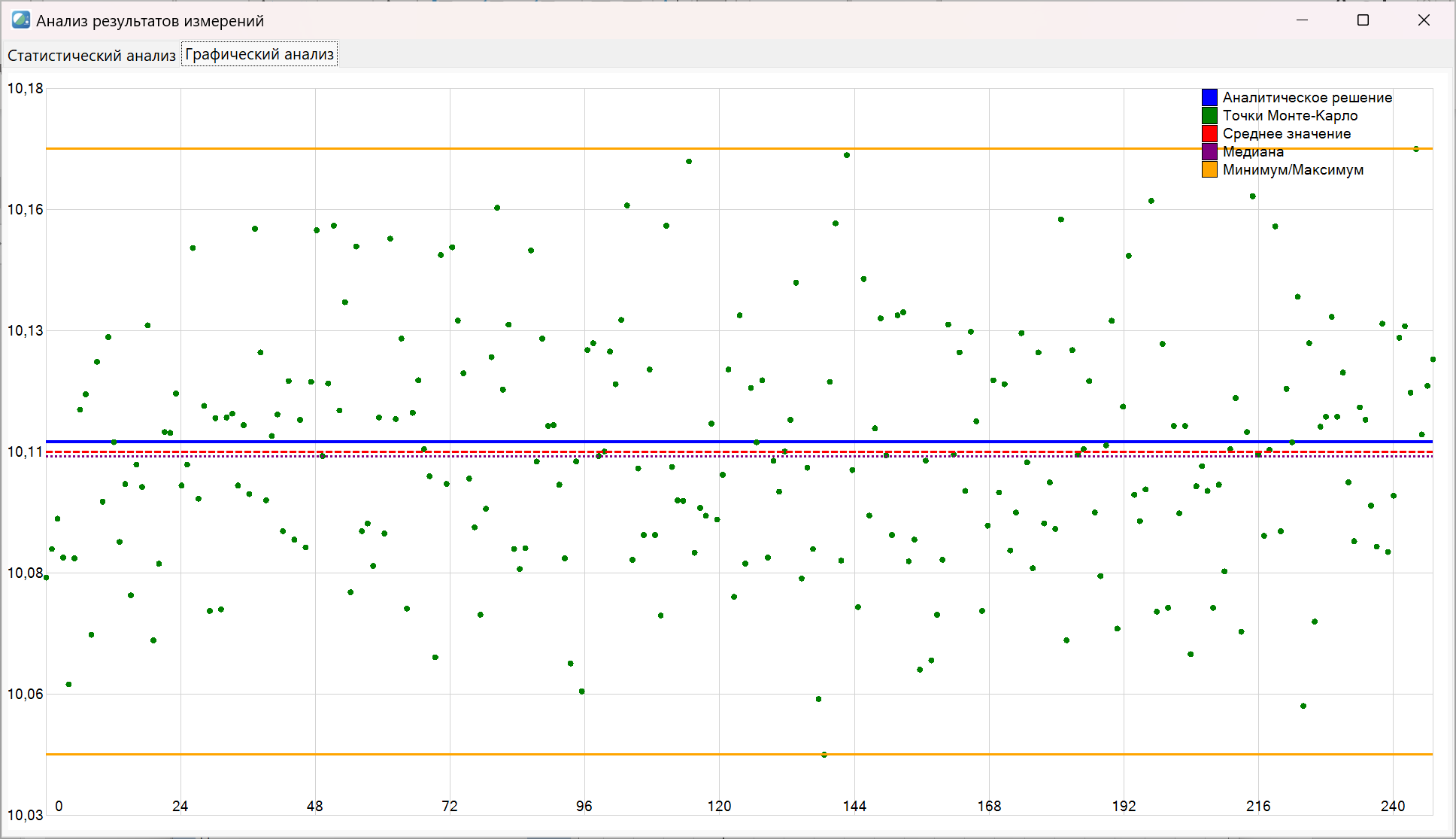


Рисунок В6 – вкладка графического анализа в окне анализа результатов измерений

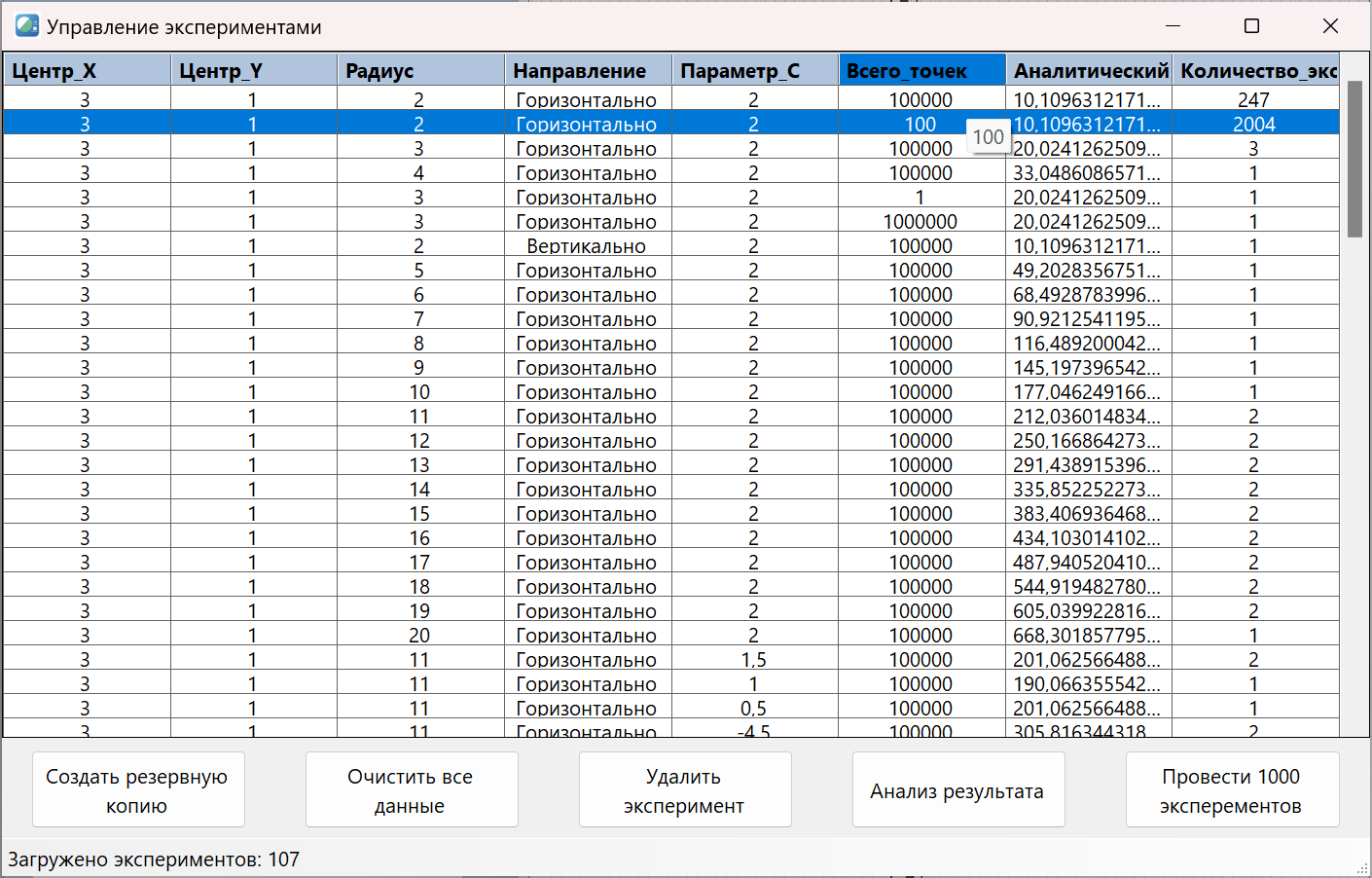


Рисунок В7 – окно управления экспериментами

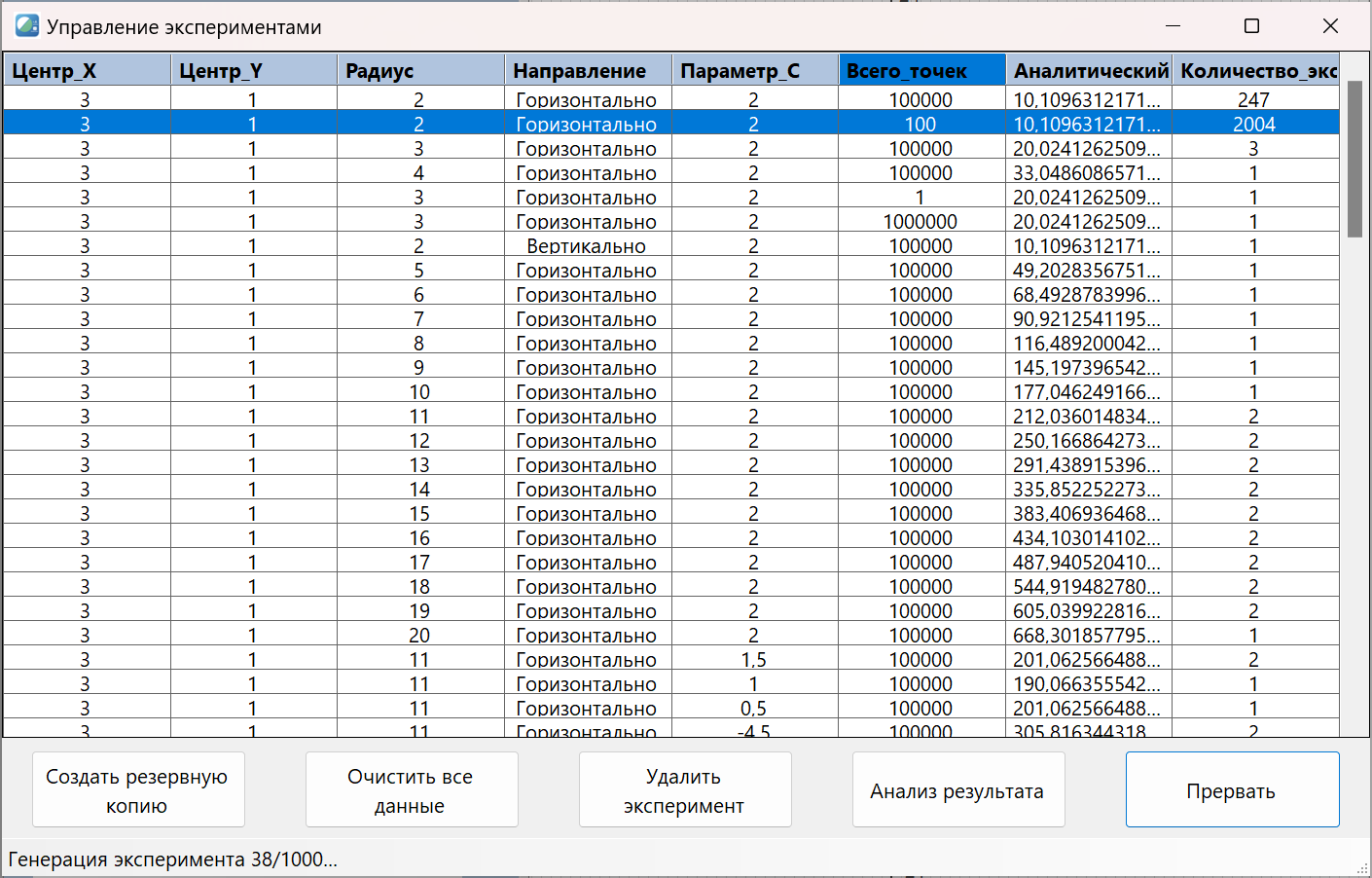


Рисунок В8 – проведение 1000 экспериментов