**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–22/1 |

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.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc201919113)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc201919114)

[1. Анализ предметной области 6](#_Toc201919115)

[1.1 Общая постановка задач 6](#_Toc201919116)

[1.2 Уточнение постановки задачи 6](#_Toc201919117)

[1.3 Математическая модель 9](#_Toc201919118)

[2. Обоснование выбора средств методов и средств 9](#_Toc201919119)

[2.1 Описание метода решения 9](#_Toc201919120)

[2.2 Анализ средств разработки 10](#_Toc201919121)

[3. Проектирование приложения 12](#_Toc201919122)

[3.1 UML - диаграммы 12](#_Toc201919123)

[3.2 Wireframe-эскизы приложения 15](#_Toc201919124)

[3.3 Описание выбранных форматов данных 21](#_Toc201919125)

[4. Разработка алгоритмов 23](#_Toc201919126)

[5. Разработка приложения 24](#_Toc201919127)

[6. Тестирование приложения 25](#_Toc201919128)

[7. Анализ качества моделирования 26](#_Toc201919129)

[ЗАКЛЮЧЕНИЕ 29](#_Toc201919130)

[СПИСОК ЛИТЕРАТУРЫ 30](#_Toc201919131)

[ПРИЛОЖЕНИЕ А 32](#_Toc201919132)

[Приложение Б 41](#_Toc201919133)

[ПРИЛОЖЕНИЕ В 82](#_Toc201919134)

# ВВЕДЕНИЕ

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

Цель практики: разработка автоматизированной системы (АС), которая включает модули для моделирования и аналитической обработки данных. Система предназначена для наглядного представления методов анализа и визуализации при вычислении площади крупного сегмента, ограниченного окружностью радиуса R (с центром в точке (x₀, y₀)) и прямой, параллельной оси ординат и расположенной на расстоянии C от неё. Расчёты выполнялись с применением как аналитических формул, так и метода Монте-Карло.

В рамках практики решались следующие задачи:

* определение требований, необходимых для выполнения поставленной преподавателем задачи;
* разработка технического задания на основе исходной формулировки;
* изучение теоретических основ расчёта площади сегмента указанной фигуры;
* проектирование и реализация пользовательского интерфейса, обеспечивающего удобное и интуитивно понятное взаимодействие с программой;
* написание программного кода, реализующего алгоритм Монте-Карло для вычисления площади заданной области;
* создание интерфейсных элементов приложения, включая стартовый экран, основной модуль для ввода количества точек, интерфейсы численного и графического моделирования, формы вывода результатов, модуль анализа данных и окно с информацией о программе;
* интеграция встроенной справочной системы для помощи пользователям в работе с приложением.

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:

* аналитический - точный математический расчёт;
* 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**

На основании технического задания (ТЗ) стандарт IDEF0 применяется для наглядного отображения логических взаимосвязей между процессами. Контекстная диаграмма и диаграмма декомпозиции, разработанные в соответствии с ТЗ, позволяют системно проанализировать структуру и взаимодействие всех компонентов системы.

В соответствии с техническим заданием, общая контекстная диаграмма (A0), представленная на рисунке 1, отображает основную функцию системы – «Вычислить площадь методом Монте-Карло» (A0).

Входные данные:

* количество точек

Выходные данные:

* результаты анализа вычислений
* графическое представление математического моделирования
* сохранённые результаты вычислений

Механизмы выполнения:

* программа Monte-Karlo
* база данных SQLite (для хранения результатов)

Управляющие элементы:

* методы статистического анализа
* техническое задание (ТЗ)
* вычислительные методы



Рисунок 1 – диаграмма IDEF0

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

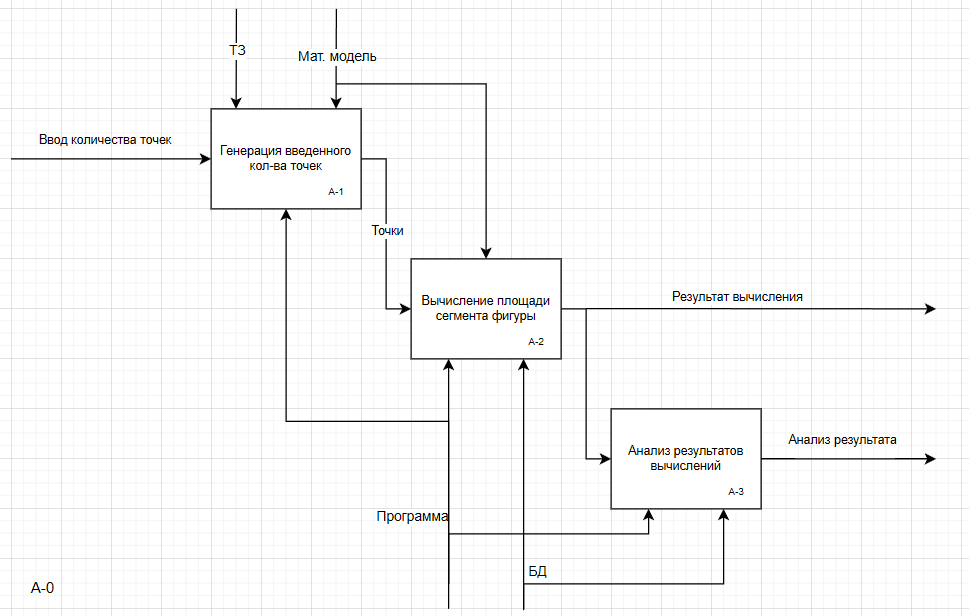


Рисунок 2 – диаграмма IDEF0 с декомпозицией

* A1 «Генерация введенного количества точек». Принимает на вход «количество точек». Результатом работы являются «Точки». Механизм – «Программа Monte-Karlo»;
* A2 «Вычисление площади сегмента фигуры». Использует «Точки» (выход A1). Для выполнения используется механизм «Программа Monte-Karlo», применяющий «Мат. модель». Возвращает «Результат вычислений»;
* A3 «Анализ результатов вычислений». Принимает результаты вычислений. Возвращает анализ результата. 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 вертикально направленной линии.

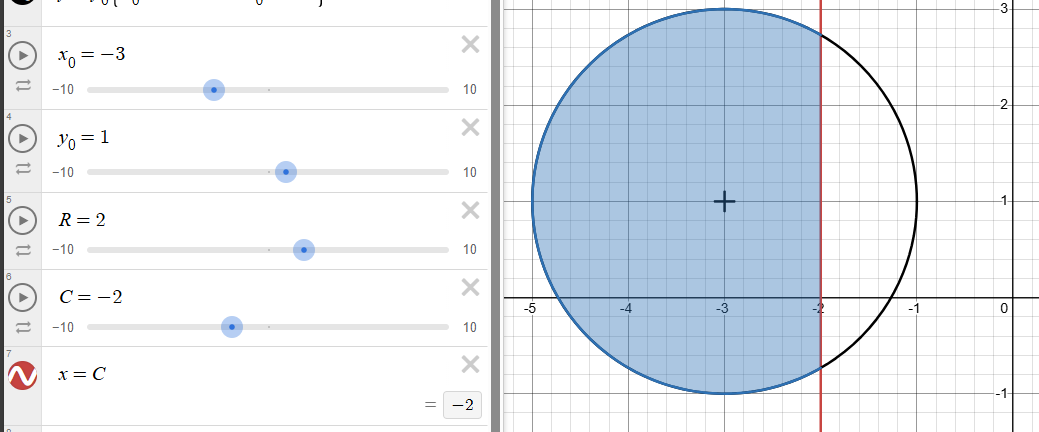


Рисунок 3 – математическое моделирование с помощью Desmos

Математическая модель наглядно иллюстрирует:

* Вывод расчётной формулы площади сегмента
* Фундаментальное свойство - суммарная площадь сегментов в точности соответствует площади всей окружности

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

## **Описание метода решения**

Для определения площади большего сегмента будут использоваться 2 подхода:

* аналитический. Вычисляет точный результат площади сегмента. Формула площади сегмента для данного метода: S = R²arccos(d/R) - d√(R²-d²), где d = |C - x0| (или y0);
* стохастический. Будет использоваться метод Монте-Карло, основанный на генерации случайных точек в квадрате, и подсчёте точек, попавших в сегмент окружности. Формула для метода Монте-Карло: S ≈ (Nточек\_внутри/Nточек\_всего) \* Sквадрата.

Важно отметить, что так как метод Монте-Карло стохастический, он даёт приближённый результат, точность которого повышается с увеличением количества точек и проведённых экспериментов.

## **Анализ средств разработки**

Разработка программного обеспечения выполняется на языке C# в среде Microsoft Visual Studio, что обеспечивает тесную интеграцию с платформой .NET и доступ к современным инструментам разработки. C# был выбран благодаря своей производительности, безопасности и поддержке различных парадигм программирования, включая объектно-ориентированный подход и асинхронные операции. Богатая стандартная библиотека и активное сообщество разработчиков делают этот язык оптимальным выбором для реализации сложных вычислительных алгоритмов.

Для хранения данных используется легковесная СУБД SQLite в сочетании с Entity Framework Core. Такой подход позволяет эффективно работать с базой данных, используя преимущества объектно-реляционного отображения. Entity Framework Core значительно упрощает выполнение операций с данными и управление миграциями, что особенно важно при работе с результатами вычислительных экспериментов.

Средой разработки выбран Microsoft Visual Studio, так как он предоставляет разнообразные инструменты для написания, отладки и тестирования кода. Среди ключевых преимуществ:

* встроенный визуальный конструктор 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. Комбинация SQLite и Entity Framework Core представляет собой мощное решение для работы с данными, обладающее рядом ключевых преимуществ:

* SQLite обеспечивает легковесное хранение информации, не требуя развертывания отдельного серверного приложения - все данные содержатся в едином файле, что значительно упрощает развертывание и распространение приложения. Несмотря на свою компактность, эта СУБД демонстрирует высокую производительность при работе с различными объемами данных.
* Entity Framework Core расширяет возможности работы с SQLite, предоставляя удобный механизм объектно-реляционного отображения. Благодаря поддержке LINQ разработчики получают гибкий инструмент для написания выразительных запросов непосредственно на языке C#, что повышает читаемость кода и ускоряет процесс разработки.

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

Как итог, выбранная комбинация технологий представляет собой оптимальное решение для создания десктопного приложения с требованиями к хранению структурированных данных.

## **Проектирование приложения**

## **3.1 UML - диаграммы**

UML - диаграммы служат мощным инструментом визуализации архитектуры программного комплекса. Они позволяют комплексно отобразить:

* Структурные аспекты
* Поведенческие модели
* Логику взаимодействия

**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.

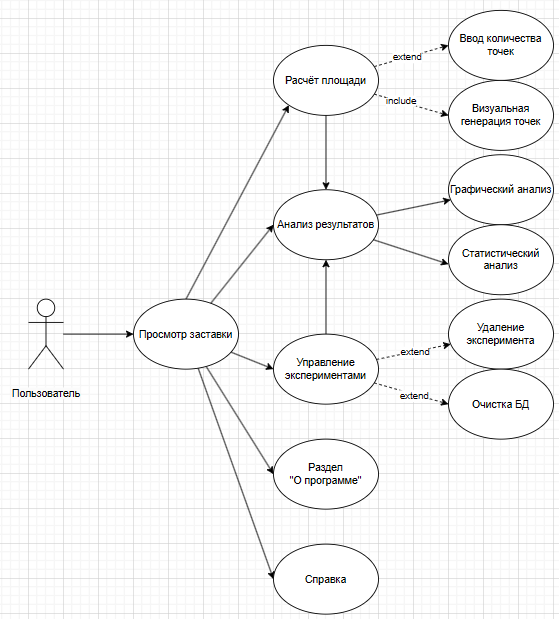


Рисунок 4 – диаграмма прецедентов

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

**3.1.2 Диаграмма деятельности**

Диаграмма деятельности UML визуализирует логику работы системы, отображая последовательность выполняемых операций и их взаимосвязи. Этот тип диаграммы детально показывает все шаги процесса, включая ветвления и параллельные потоки выполнения. На рисунке 5 представлена обобщенная диаграмма деятельности для разрабатываемой автоматизированной системы.

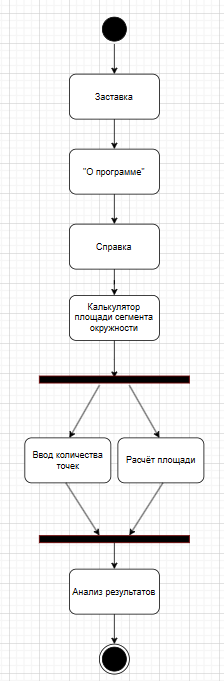


Рисунок 5 – диаграмма деятельности

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

**3.1.3 Диаграмма последовательности**

Диаграмма последовательностей UML визуализирует динамику взаимодействия объектов в системе, отражая временной порядок выполнения операций. Этот тип диаграммы играет ключевую роль в проектировании, поскольку четко демонстрирует цепочку вызовов и обмена сообщениями между компонентами. В рамках автоматизированной системы, где задействовано множество взаимосвязанных объектов, на рисунке 6 (созданном в PlantUML) представлен наиболее сложный сценарий взаимодействия, а именно инициализация объектов и возможная генерация точек с отображением результата.

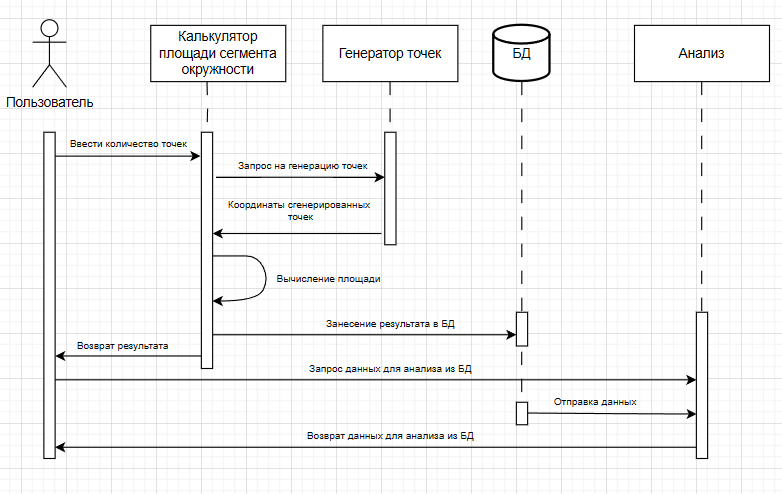


Рисунок 6 – диаграмма последовательности для инициализации объектов и возможной генерации точек

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

## **3.2 Wireframe-эскизы приложения**

Программа проектируется как многооконное приложение с продуманной структурой форм. Разработанные заранее Wireframe-эскизы служат наглядным прототипом интерфейса, позволяя до начала разработки определить внешний вид и логику работы каждой формы. Это существенно упрощает последующий процесс реализации пользовательского интерфейса.

Стартовая заставка, представленная на рисунке 7, формирует первое впечатление пользователя о приложении. Как видно из эскиза, она включает три ключевых элемента: предустановленные параметры для расчетов, информацию об авторе программы, а также пример графической визуализации. На рисунке 8 показан образец будущего графика с точками, который поможет пользователю сразу понять принцип отображения результатов моделирования.

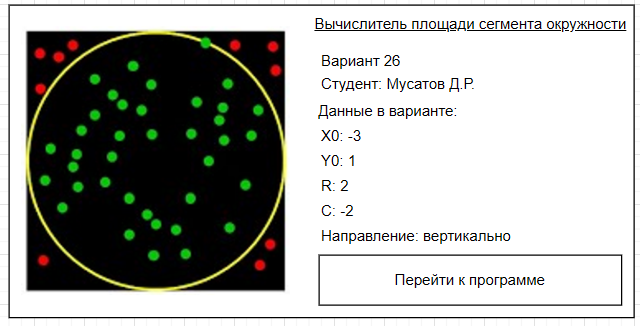


Рисунок 7 – эскиз заставки приложения

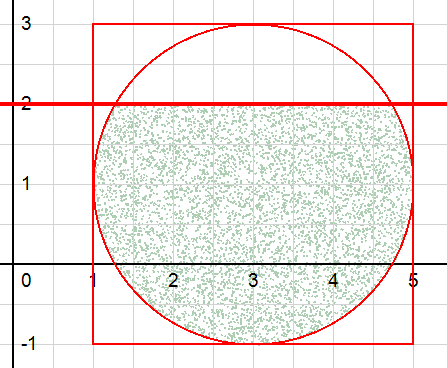


Рисунок 8 – картинка на заставке приложения

После стартовой заставки пользователь переходит в основное окно программы, представленное на рисунке 9. Этот центральный интерфейсный элемент включает несколько ключевых компонентов, организованных для удобства работы.

В правой части расположена панель управления, предназначенная для настройки параметров математического моделирования. Сверху находится навигационное меню, обеспечивающее доступ ко всем функциональным формам приложения. Основную площадь окна занимает область визуализации, расположенная справа от панели управления.

В графической зоне будет отображаться координатная плоскость с соответствующими осями и подписями. На этом поле программа рисует ограничивающий квадрат, внутри которого располагаются элементы моделирования: окружность заданного радиуса, секущая линия и сгенерированные точки. Особенностью визуализации является отображение только тех точек, которые попадают в больший сегмент окружности, что позволяет наглядно демонстрировать принцип работы метода Монте-Карло.

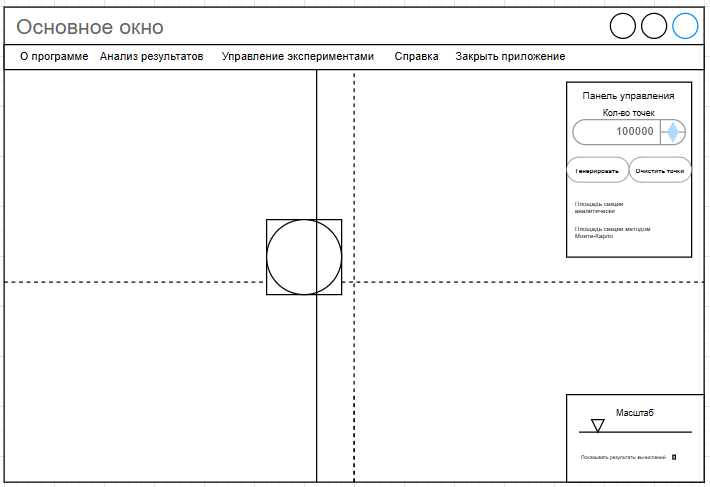


Рисунок 9 – эскиз основного окна

Справочная документация будет отображаться в браузере пользователя, используя стандартный дизайн, сгенерированный программой Dr.Explain. Это решение позволяет создать удобную и функциональную справку без необходимости разработки собственного интерфейса для документации.

Окно «О программе» разработано в минималистичном стиле и содержит всю необходимую информацию о приложении. В верхней части окна расположено название программы и крупноформатная иконка приложения, которая сразу привлекает внимание пользователя. Ниже указаны данные об авторе, текущая версия программного обеспечения и активная ссылка на GitHub-репозиторий, где можно ознакомиться с исходным кодом проекта.

Визуальное оформление окна «О программе» и дизайн иконки приложения можно увидеть на рисунках 10 и 11 соответственно. Эти элементы интерфейса созданы в едином стиле с основной программой, что обеспечивает целостность пользовательского опыта.

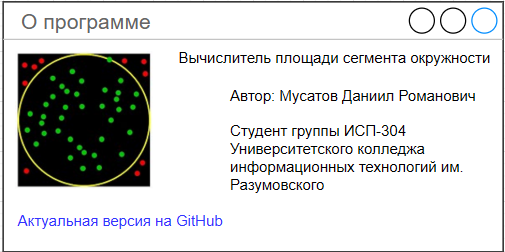


Рисунок 10 – эскиз окно «О программе»

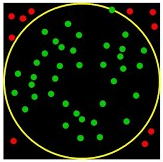
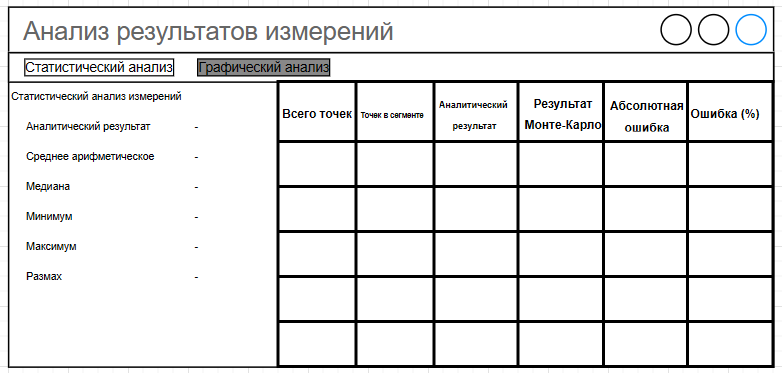


Рисунок 11 – иконка приложения

Окно «Анализ результатов» будет реализовано с интерфейсом на основе двух вкладок, позволяющих переключаться между различными представлениями данных. Первая вкладка посвящена числовой обработке результатов - здесь будет представлена детальная статистическая информация по проведенным измерениям, а также таблица с итоговыми значениями эксперимента.

Вторая вкладка, «Графический анализ», содержит визуальное представление данных. Основное пространство занимает точечная диаграмма, где каждая точка соответствует отдельному вычислению площади методом Монте-Карло. На этом же графике будут отображены линии, демонстрирующие результаты статистической обработки данных. Для удобства интерпретации в правом верхнем углу размещена цветовая легенда, поясняющая значение каждого элемента графика.



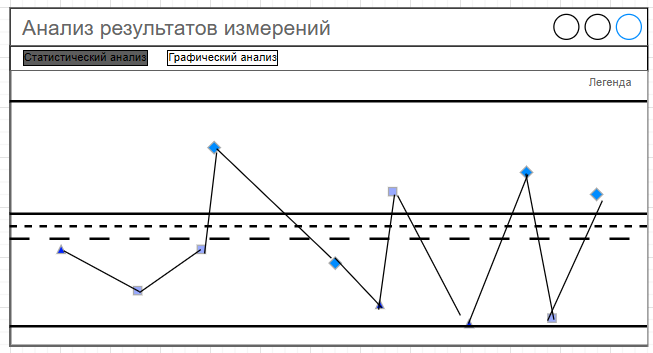


Рисунок 12 – эскиз вкладок в окне с анализом результатов

Окно управления экспериментами спроектировано как центральный узел для работы с данными исследований. Основное пространство интерфейса занимает таблица со списком всех проведенных экспериментов, где пользователь может выбирать нужные записи для дальнейших действий.

Нижняя панель содержит группу кнопок для операций над экспериментами:

* полная очистка всех экспериментальных результатов.
* удаление конкретного результата
* анализ конкретного результата

Визуальное оформление и компоновка элементов управления представлены на рисунке 13.



Рисунок 13 – эскиз окна для управления экспериментами

В целом дизайн приложения планируется простым и понятным для пользователя.

## **3.3 Описание выбранных форматов данных**

Для надежного сохранения результатов вычислений методом Монте-Карло была спроектирована реляционная модель данных, основанная на двух взаимосвязанных таблицах. Эта структура обеспечивает логичное разделение информации и поддерживает связи между сущностями.

Логическая структура базы данных с указанием типов полей, первичных ключей и связей между таблицами представлена на рисунке 14. Эта схема демонстрирует нормализованную структуру, оптимизированную для выполнения типичных запросов приложения.

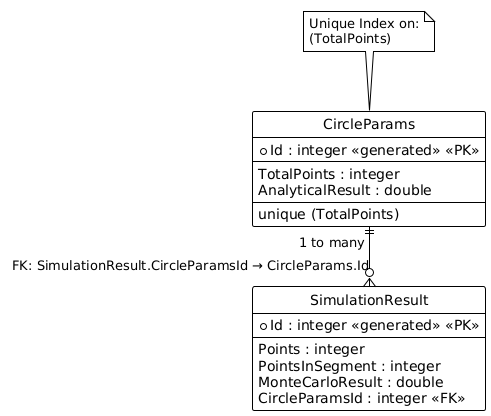


Рисунок 14 – логическая модель базы данных

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

* Id (int, PK) – уникальный идентификатор записи;
* TotalPoints (int) – общее количество точек в эксперименте;
* AnalyticalResult (double) – аналитически рассчитанная площадь.

Таблица SimulationResult (Результаты симуляций) фиксирует статистику экспериментов:

* Id (int, PK) – уникальный идентификатор;
* Points (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.

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**

Блок-схема, представленная на рисунке 15, отражает ключевые этапы работы основного алгоритма программы. Схематично показана логика выполнения операций на главной форме приложения, начиная от инициализации компонентов до вывода конечных результатов.

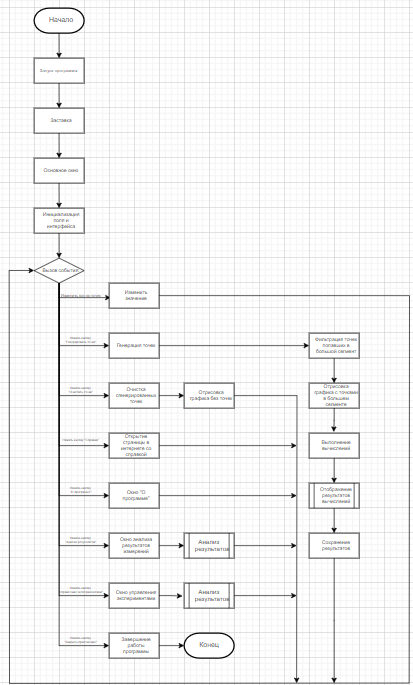


Рисунок 15 – общая блок схем приложения

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, повторный дубликат программы не запустится, и активирует окно предыдущей запущенной версии.

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. В числовое поле количества точек, доступное для ввода с клавиатуры не получится ввести некорректные значения, такие как буквы или символы, а количество не получится задать меньше 1 и больше 108.

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 удалить все результаты. Выбрав эксперимент нажатием на строчку с экспериментом, можно удалить его или посмотреть анализ.

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 44 юнит-тестов и 1 интеграционный тест, что подтверждает высокую степень покрытия кода тестами. Результаты успешного прохождения тестов представлены на рисунке 16.

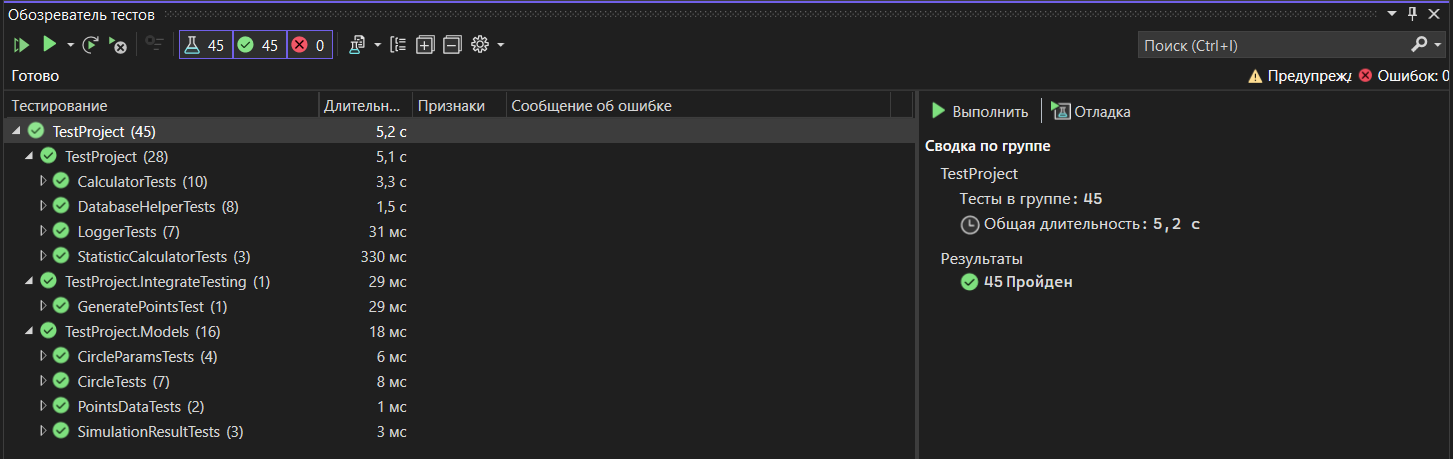


Рисунок 16 – результат прохождения авто тестов

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;
* 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 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.

Основной целью практики стало приобретение профессиональных навыков и компетенций через выполнение задач, соответствующих будущей специальности. В рамках учебной практики был получен ценный опыт интеграции программных модулей, а также освоены ключевые профессиональные компетенции.

В процессе работы проведен анализ требований к решению задачи вычисления площади сегмента окружности. На основании условий, предоставленных преподавателем, разработано техническое задание. Изучены математические основы расчетов с применением аналитического подхода и метода Монте-Карло.

Реализован удобный графический интерфейс, содержащий стартовый экран, основной рабочий модуль, компоненты численного и графического анализа, систему сохранения результатов и справочный раздел. Программно воплощен метод Монте-Карло с функцией визуализации процесса вычислений. Дополнительно интегрирована встроенная справочная система для помощи пользователям.

Завершающим этапом стало тестирование приложения и проверка точности расчетов. В результате успешно создано программное обеспечение, сочетающее аналитические и статистические методы вычислений с продуманным интерфейсом. Приобретенные навыки проектирования и разработки программных решений способствуют профессиональному росту в сфере создания программного обеспечения. Полученный опыт имеет значительную практическую ценность для дальнейшего обучения и профессиональной деятельности.

# СПИСОК ЛИТЕРАТУРЫ

1. Берестнев, В.Ф. «Численные методы: учебное пособие». 2-е изд., перераб. и доп. – М.: КНОРУС, 2021. – 296 с.
2. Meyers, S. «Effective C#: 50 Specific Ways to Improve Your C#». 3rd Edition. Pearson, 2021.
3. Freeman, A. «Pro .NET 8 on Windows: Includes WinForms, WPF, and UWP». Apress, 2024.
4. Джеймс Фаррелл. «WinForms для профессионалов. Современная разработка десктопных приложений». Питер, 2023.
5. Grokhotkov, A.A. «Метод Монте-Карло: современный подход». Вестник МФТИ, 2021, №4.
6. Марчук Г.И., Рыжов Ю.А. «Вычислительные методы и прикладное программирование». М.: МГУ, 2022.
7. Dyer, M., «Monte Carlo Methods in Practice». Cambridge University Press, 2022.
8. Platt, D. «Introducing Entity Framework Core: Getting Started with Microsoft's Modern Data Access». O’Reilly, 2021.
9. Wise, D. «Practical Entity Framework Core 6: Database Access for Enterprise Applications». Apress, 2022.
10. Kline, K. "SQL in a Nutshell: A Desktop Quick Reference". 4th Edition. O’Reilly, 2022.
11. McCabe, J. «Testing .NET Applications». Microsoft Press, 2021.
12. ГОСТ 34.602-2020. Информационные технологии. Комплекс стандартов на автоматизированные системы. Требования к содержанию документов.
13. ГОСТ Р 7.0.100-2020. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись.
14. Fowler, M. «UML Distilled: A Brief Guide to the Standard Object Modeling Language». 4th Edition. Addison-Wesley, 2022.
15. Lidwell, W., Holden, K., Butler, J. «Universal Principles of Design». 3rd Edition. Rockport Publishers, 2023.
16. Шальнов А.И., Киселев Д.Ю. «Основы тестирования и качества программного обеспечения». СПб.: БХВ-Петербург, 2021.
17. Nielsen, J. «Usability Engineering». Revised Edition. Morgan Kaufmann, 2020.
18. Microsoft. «Документация по Windows Forms». [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/ (дата обращения: 23.06.2025).
19. DrExplain Documentation Generator. [Электронный ресурс]. URL: https://www.drexplain.com/ru/ (дата обращения: 25.06.2025).
20. Fenton, N., Bieman, J. "Software Metrics: A Rigorous and Practical Approach". 4th Edition. CRC Press, 2022.
21. Math.NET Numerics. [Электронный ресурс]. URL: https://numerics.mathdotnet.com/ (дата обращения: 26.06.2025).
22. Спирин, А.В. «Практика проектирования интерфейсов». М.: ДМК Пресс, 2022.

**ПРИЛОЖЕНИЕ А**

**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: Мусатов Даниил Романович

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.
  1. **Назначение АС**

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

**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

Структура программы:

* заставка;
* основная форма;
* форма анализа моделирования;
* форма «О программе»;
* форма управления экспериментами;
* встроенная справка.

4.1.1 Компоненты взаимодействуют через общий модуль данных.

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

4.1.3 Требование к режимам функционирования АС:

* интерактивный режим с визуализацией.

4.1.4. 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.

4.1.5 Перспективы развития, модернизации АС:

* добавление поддержки других геометрических фигур;
* возможность задания координат центра окружности
* возможность задания радиуса окружности
* 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:

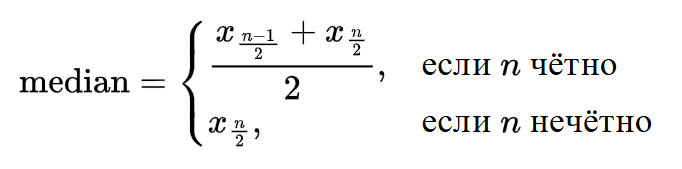
* формула площади круга: S = πR2
* формула площади квадрата, описывающего круг Sквадрата = 4R2
* формула площади сегмента: S = R²arccos(d/R) - d√(R²-d²), где d = |C - x0| (или y0)
* формула для метода Монте-Карло: S ≈ (Nвнутри/Nвсего) \* Sквадрата
* алгоритм генерации случайных точек: координата точки 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.

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;
  + библиотека 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.

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.

4.4.8 Требования по сохранности информации при авариях:

* создание резервной копии пользователем.

4.4.9 Требования к защите от влияния внешних воздействий не требуются.

4.4.10 Требования к патентной чистоте и патентоспособности не требуются.

4.4.11 Требования по стандартизации и унификации

* использование стандартных и сторонних библиотек C#.

4.4.12 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.

Документ: Техническое задание (разделы 1 и 2 по ГОСТ 34.602-89).

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;
* сценарии ручного тестирования интерфейса;
* проведение нагрузочных и граничных тестов алгоритмов Монте-Карло;
* 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**

Код класса AboutProgramForm.cs

using System.Diagnostics;

namespace Monte\_Karlo

{

// Частичная реализация формы "О программе"

public partial class AboutProgramForm : Form

{

// Конструктор формы

public AboutProgramForm()

{

InitializeComponent(); // Инициализация компонентов формы (создаётся автоматически дизайнером)

}

// Обработчик события клика по ссылке на GitHub

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

{

try

{

// Создание процесса для открытия ссылки в браузере по умолчанию

Process.Start(new ProcessStartInfo

{

FileName = "https://github.com/rpo6oBLLjuk/EducationalPractice\_ISP-304", // URL репозитория

UseShellExecute = true // Указывает системе использовать оболочку для открытия ссылки

});

}

catch (Exception ex)

{

// В случае ошибки — вывод сообщения пользователю

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

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

Код класса AnalysisForm.cs

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

using Monte\_Karlo.Utilites.View;

using System.ComponentModel;

using System.Data;

namespace Monte\_Karlo

{

// Форма анализа результатов расчетов Монте-Карло

public partial class AnalysisForm : Form

{

// Список результатов симуляций, отображаемых в таблице

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

// Текущие параметры круга (входные данные и результаты)

private CircleParams \_currentParams;

// Представление для визуализации анализа

private AnalysisView \_view;

// Конструктор по умолчанию

public AnalysisForm()

{

InitializeComponent(); // Инициализация компонентов формы

\_view = new AnalysisView(); // Создание объекта визуализации

}

// Конструктор с параметрами круга

public AnalysisForm(CircleParams circleParams) : this()

{

if (circleParams is not null)

{

\_currentParams = circleParams; // Сохраняем текущие параметры

\_results = circleParams.Results; // Загружаем список результатов

}

}

// Обработчик события загрузки формы

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

{

SetupDataGridView(); // Настройка таблицы для отображения результатов

CalculateStatistics(); // Вычисление и отображение статистики

}

// Настройка столбцов и данных DataGridView

private void SetupDataGridView()

{

dataGridViewResults.Columns.Clear(); // Очистка всех столбцов перед настройкой

// Добавление столбцов с разными параметрами: имя, заголовок, имя свойства данных, формат и т.д.

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

AddColumn("PointsInCircle", "Всего Точек", "Points", true, "N0", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

AddColumn("PointsInSegment", "Точек в сегменте", "PointsInSegment", true, "N0", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

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

\_currentParams?.AnalyticalResult.ToString("F4") ?? "N/A", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

AddColumn("MonteCarloResult", "Результат Монте-Карло", "MonteCarloResult", true, "F4", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

AddColumn("AbsoluteError", "Абсолютная погрешность", "AbsoluteError", true, "F2", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

AddColumn("RelativeError", "Ошибка (%)", "RelativeError", true, "F2", AutoSizeMode: DataGridViewAutoSizeColumnMode.Fill);

// Если есть параметры и результаты, заполняем таблицу данными

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

{

int id = 0;

var displayResults = \_currentParams.Results

.OrderByDescending(r => r.Id) // Сортируем по убыванию Id

.Select(r => new

{

Id = ++id, // Порядковый номер записи для отображения

r.Points,

r.PointsInSegment,

\_currentParams.AnalyticalResult,

r.MonteCarloResult,

// Вычисление абсолютной и относительной ошибки с помощью калькулятора

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

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

})

.ToList();

// Создаем источник данных для DataGridView и присваиваем его

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

dataGridViewResults.DataSource = bindingSource;

}

else

{

dataGridViewResults.DataSource = null; // Если данных нет - очищаем таблицу

}

}

// Форматирование ячеек, например, цвет текста в зависимости от значения ошибки

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

{

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

{

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

// Меняем цвет текста: красный при ошибке > 10%, оранжевый > 5%, иначе зеленый

e.CellStyle.ForeColor = error switch

{

> 10 => Color.Red,

> 5 => Color.Orange,

\_ => Color.Green

};

}

}

// Метод добавления столбца в DataGridView с параметрами форматирования и отображения

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, // Имя столбца

HeaderText = header, // Текст заголовка

DataPropertyName = dataPropertyName, // Свойство объекта для отображения

ReadOnly = isReadOnly, // Только для чтения

DefaultCellStyle = new DataGridViewCellStyle

{

Format = format, // Формат отображения данных (числовой, текстовый и т.д.)

Alignment = DataGridViewContentAlignment.MiddleCenter, // Выравнивание по центру

},

SortMode = DataGridViewColumnSortMode.Programmatic, // Сортировка программная (обрабатывается вручную)

AutoSizeMode = AutoSizeMode

};

// Если указан дефолтный текст для пустых значений

if (defaultValue != null)

{

col.DefaultCellStyle.NullValue = defaultValue;

}

dataGridViewResults.Columns.Add(col); // Добавляем столбец в таблицу

}

// Обработчик клика по заголовку столбца для сортировки данных

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

{

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

// Определяем направление сортировки (переключаем между Ascending и Descending)

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

ListSortDirection.Descending :

ListSortDirection.Ascending;

// Выполняем сортировку по выбранному столбцу и направлению

SortData(column.Name, direction);

// Сброс иконок сортировки у всех столбцов

dataGridViewResults.Columns.Cast<DataGridViewColumn>()

.ToList()

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

// Устанавливаем иконку сортировки для текущего столбца

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

SortOrder.Ascending :

SortOrder.Descending;

}

// Метод сортировки данных по имени столбца и направлению сортировки

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:

// По умолчанию сортируем по Id

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

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

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

break;

}

}

}

// Вычисление и отображение статистических характеристик результатов

private void CalculateStatistics()

{

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

return; // Если результатов нет - выход

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

// Отображение аналитического результата

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

// Среднее значение результатов Монте-Карло

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

// Медиана

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

// Минимальное значение

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

// Максимальное значение

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

// Размах (максимум - минимум)

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

}

// Отрисовка панели с визуализацией анализа

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

{

\_view.RenderAnalysis(paintPanel, e, \_currentParams); // Рендер визуализации на панели

dataGridViewResults.Invalidate(true); // Перерисовка таблицы для обновления визуала

base.OnPaint(e); // Вызов базового обработчика отрисовки

}

}

}

Код класса AnalysisView.cs

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites.Calculators;

using System.Drawing.Drawing2D;

namespace Monte\_Karlo.Utilites.View

{

// Класс, отвечающий за отображение (визуализацию) результатов анализа на элементе управления Panel

public class AnalysisView

{

// Цвета для различных элементов графика

private static readonly Color \_analyticalColor = Color.LawnGreen; // Цвет линии аналитического решения

private static readonly Color \_pointsColor = Color.Black; // Цвет точек Монте-Карло

private static readonly Color \_pointsLinesColor = Color.LightSlateGray; // Цвет линий, соединяющих точки

private static readonly Color \_meanColor = Color.Orange; // Цвет линии среднего значения

private static readonly Color \_medianColor = Color.Blue; // Цвет линии медианы

private static readonly Color \_minMaxColor = Color.Red; // Цвет линий минимума и максимума

private static readonly Color \_backgroundColor = Color.White; // Цвет фона панели

private static readonly Color \_gridColor = Color.LightGray; // Цвет сетки графика

private static readonly Color \_legendBackgroundColor = Color.LightGray; // Цвет фона легенды

private static readonly Padding \_padding = new Padding(100, 20, 70, 40); // Отступы графика внутри панели (слева, сверху, справа, снизу)

private static readonly double \_percentYPadding = 0.1; // Дополнительный отступ по оси Y (10%)

private static readonly Font \_textFont = new Font("SegoUI", emSize: 8); // Шрифт для текста

private static readonly Brush \_textBrush = Brushes.Black; // Кисть для текста

private static readonly float \_pointRadius = 5; // Радиус точек

// Основной публичный метод для рендера анализа на панели

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

{

var g = e.Graphics;

g.Clear(\_backgroundColor); // Очистка области рисования белым цветом

// Проверка, есть ли результаты для отображения

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

{

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

return;

}

OnPaint(panel, g, circleParams); // Отрисовка графика

}

// Вспомогательный приватный метод, который выполняет всю отрисовку

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

{

g.SmoothingMode = SmoothingMode.AntiAlias; // Включаем сглаживание для более красивой отрисовки линий и фигур

// Извлекаем результаты Монте-Карло

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

double analyticalValue = circleParams.AnalyticalResult;

// Вычисляем основные статистики

double mean = mcResults.Average();

double median = StatisticCalculator.CalculateMedian(mcResults);

double min = mcResults.Min();

double max = mcResults.Max();

// Задаём область отрисовки графика с учётом отступов

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

// Рассчитываем диапазон по оси Y с дополнительным паддингом (отступом)

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); // Рисуем сетку

DrawAnalyticalLine(g, plotArea, analyticalValue, yMin, yRange); // Рисуем линию аналитического решения

DrawMeanLine(g, plotArea, mean, yMin, yRange); // Рисуем линию среднего значения

DrawMedianLine(g, plotArea, median, yMin, yRange); // Рисуем линию медианы

DrawMinMaxLines(g, plotArea, min, max, yMin, yRange); // Рисуем линии минимума и максимума

DrawMonteCarloPoints(g, plotArea, mcResults, yMin, yRange); // Рисуем точки Монте-Карло и соединяющие линии

DrawLegend(g, plotArea, median); // Рисуем легенду

}

// Метод отрисовки сетки графика с подписями по оси X (номер опыта) и Y (значения)

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

{

Pen girdPen = new Pen(\_gridColor);

g.DrawRectangle(girdPen, plotArea); // Рисуем рамку области графика

// Вертикальные линии — каждые 10% от количества точек

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);

// Подписываем номера экспериментов под осью X

string text = (i + 1).ToString();

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

float textX = x - textSize.Width / 2;

float textY = plotArea.Bottom - textSize.Height + 20;

// Для первой линии сдвигаем подпись вправо, чтобы не залезала на край

if (i == 0)

textX += textSize.Width;

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

}

// Горизонтальные линии сетки (6 интервалов — 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 по оси Y

double yRange = yMax - yMin;

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

string text = value.ToString("F6"); // Формат с 6 знаками после запятой

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

float textX = plotArea.Left - textSize.Width - 15;

float textY = y - textSize.Height / 2;

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

}

}

// Рисует горизонтальную линию аналитического решения по всей ширине графика

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);

}

// Отрисовка точек Монте-Карло и линий, соединяющих их

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;

float xStep = Math.Max(1f, count);

float yScale = height / (float)yRange;

float currentX = 0;

float currentY = 0;

float previousX;

float previousY;

Pen pointsLinesPen = new(\_pointsLinesColor, 1);

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

{

float diameter = 2 \* \_pointRadius;

// Рисуем линии между точками

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

{

try

{

previousX = currentX + 1;

previousY = currentY;

currentX = left + width \* i / xStep - 0.5f;

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

if (i == 0)

{

previousX = currentX;

previousY = currentY;

}

g.DrawLine(pointsLinesPen, new PointF(previousX, previousY), new PointF(currentX, currentY));

}

catch (DivideByZeroException)

{

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

return; // Останавливаем отрисовку при ошибке

}

}

// Рисуем сами точки

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

{

try

{

currentX = left + width \* i / xStep;

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

g.FillEllipse(brush, currentX - \_pointRadius, currentY - \_pointRadius, diameter, diameter);

}

catch (DivideByZeroException)

{

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

return;

}

}

}

}

// Рисует пунктирную 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 = DashStyle.Dash },

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

}

// Рисует пунктирную точечную линию медианы

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 = DashStyle.Dot },

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

}

// Рисует линии минимума и максимума 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);

}

// Отрисовка легенды с пояснениями цветов и линий

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;

// Фон легенды

g.FillRectangle(new SolidBrush(\_legendBackgroundColor), startX, startY, 260, 100);

// Заголовок легенды

g.DrawString("Легенда", \_textFont, \_textBrush, startX + 5, startY);

// Пояснения с цветами и стилями

DrawLegendItem(g, "Аналитическое решение", \_analyticalColor, startX, startY + 1 \* itemHeight + 5, 4, DashStyle.Solid);

DrawLegendItem(g, "Среднее значение", \_meanColor, startX, startY + 2 \* itemHeight + 5, 3, DashStyle.Dash);

DrawLegendItem(g, "Медиана", \_medianColor, startX, startY + 3 \* itemHeight + 5, 3, DashStyle.Dot);

DrawLegendItem(g, "Минимум/Максимум", \_minMaxColor, startX, startY + 4 \* itemHeight + 5, 3, DashStyle.Solid);

DrawLegendItem(g, "Точки Монте-Карло", \_pointsColor, startX, startY + 5 \* itemHeight + 5, 10, DashStyle.Solid, true);

}

// Вспомогательный метод отрисовки одного элемента легенды

private void DrawLegendItem(Graphics g, string text, Color color, float x, float y, int lineWidth, DashStyle dashStyle, bool drawCircle = false)

{

using (Pen pen = new Pen(color, lineWidth) { DashStyle = dashStyle })

{

if (drawCircle)

{

g.FillEllipse(new SolidBrush(color), x + 5, y + 5, 10, 10);

}

else

{

g.DrawLine(pen, x + 5, y + 10, x + 25, y + 10);

}

g.DrawString(text, \_textFont, \_textBrush, x + 30, y);

}

}

}

}

Код класса AppDbContext.cs

using Microsoft.EntityFrameworkCore;

using Monte\_Karlo.Models;

namespace Monte\_Karlo.DataBase

{

public class AppDbContext : DbContext

{

// Таблица параметров круга

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

// Таблица результатов моделирования

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

// Настройка подключения к базе данных

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

string databasePath = "DataBase.db"; // Путь к файлу базы данных

optionsBuilder.UseSqlite($"Data Source={databasePath}"); // Использование SQLite как источника данных

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

}

// Конфигурация модели данных

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<CircleParams>()

.HasIndex(cp => new

{

cp.TotalPoints // Уникальный индекс по количеству точек

})

.IsUnique();

}

}

}

Код класса Calculator.cs

using Monte\_Karlo.Models;

namespace Monte\_Karlo.Utilites.Calculators

{

// Статический класс для вычислений, связанных с кругом и методом Монте-Карло

public static class Calculator

{

// Метод для аналитического вычисления площади фигуры, образованной пересечением круга и вертикальной прямой x = C

public static double CalculateAnalyticArea(Circle circle)

{

Point center = circle.circleCenter;

double R = circle.radius;

double C = circle.C;

if (R == 0)

throw new ArgumentException("R == 0"); // Радиус не может быть нулём

double xLine = C; // Положение вертикальной линии x = C

double d = Math.Abs(center.X - xLine); // Расстояние от центра круга до этой линии (хорды)

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

double CircleArea = Math.PI \* R \* R; // Полная площадь круга

if (d >= R)

return CircleArea; // Если линия не пересекает круг, площадь равна полной площади круга

if (h == R)

return CircleArea / 2; // Если линия проходит через центр круга (хорда — диаметр), площадь — половина круга

double segmentArea = GetSegmentArea(R, d); // Площадь сегмента круга, отсекаемого линией

return CircleArea - segmentArea; // Площадь части круга слева от линии (или справа, в зависимости от контекста)

}

// Вспомогательный метод для вычисления площади сегмента круга по радиусу и расстоянию до хорды (https://en.wikipedia.org/wiki/Circular\_segment)

private static double GetSegmentArea(double R, double d)

{

// Формула площади сегмента круга

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

}

// Метод для вычисления площади круга по радиусу

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

// Метод для оценки площади методом Монте-Карло

// radius - радиус круга, allPoints - всего точек, cuttedPoints - точек, попавших внутрь круга

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

{

double squareArea = 4 \* radius \* radius; // Площадь квадратной области, в которой генерируются точки (-R до R по X и Y)

return cuttedPoints / (double)allPoints \* squareArea; // Оценка площади круга через отношение точек внутри к общему числу, умноженная на площадь квадрата

}

// Метод для вычисления абсолютной ошибки между ожидаемым и полученным результатом

public static double CalculateAbsoluteError(double expectedResult, double actualResult)

{

var result = expectedResult - actualResult;

result = RoundToTwoSignificantDigits(result, 2); // Округление результата до 2 значимых цифр

return result;

}

// Метод для вычисления относительной ошибки в процентах

public static double CalculateRelativeError(double expectedResult, double actualResult)

{

if (expectedResult <= 0)

throw new ArgumentException("Ожидаемое значение не может быть <= 0"); // Проверка корректности ожидаемого результата

if (actualResult < 0)

throw new ArgumentException("Полученное значение не может быть < 0"); // Проверка корректности полученного результата

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

result = RoundToTwoSignificantDigits(result, 2); // Округление до 2 значимых цифр

return result;

}

// Метод округления числа до заданного количества значимых 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))); // Позиция первой значимой цифры

double scale = Math.Pow(10, significantDigits - log10 - 1); // Масштаб для округления

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

// Убираем возможные артефакты округления (например, 0.30000000000000004)

return BitConverter.Int64BitsToDouble(BitConverter.DoubleToInt64Bits(rounded));

}

}

}

Код класса CalculatorTests.cs

namespace TestProject

{

[TestClass]

public class CalculatorTests

{

// Проверяем правильность вычисления площади круга

[TestMethod]

public void CircleSuare\_CalculatesCorrectArea()

{

// Arrange

double radius = 5;

double expected = Math.PI \* 25;

// Act

double result = Calculator.CircleSuare(radius);

// Assert

Assert.AreEqual(expected, result);

}

// Проверяем правильность вычисления метод Монте-Карло

[TestMethod]

public void CalculateMonteCarloArea\_ReturnsCorrectValue()

{

// Arrange

float radius = 2;

int allPoints = 1000;

int cuttedPoints = 785;

double expected = (785d / 1000d) \* 16; // (cutted/all) \* (4\*radius^2)

// Act

double result = Calculator.CalculateMonteCarloArea(radius, allPoints, cuttedPoints);

// Assert

Assert.AreEqual(expected, result);

}

// Проверяем правильность вычисления абсолютной погрешности

[TestMethod]

public void CalculateAbsoluteError\_ReturnsCorrectValue()

{

// Arrange

double expected = 10.5;

double actual = 9.8;

double expectedError = 0.7;

// Act

double result = Calculator.CalculateAbsoluteError(expected, actual);

// Assert

Assert.AreEqual(expectedError, result, 0.0001);

}

// Проверяем правильность вычисления относительной погрешности

[TestMethod]

public void CalculateRelativeError\_ReturnsCorrectPercentage()

{

// Arrange

double expected = 100;

double actual = 95;

double expectedError = 5; // 5%

// Act

double result = Calculator.CalculateRelativeError(expected, actual);

// Assert

Assert.AreEqual(expectedError, result, 0.0001);

}

// Проверяем реакцию вычисления погрешности при некорректных значениях

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void CalculateRelativeError\_ThrowsForZeroExpected()

{

// Arrange

double expected = 0;

double actual = 95;

// Act

Calculator.CalculateRelativeError(expected, actual);

}

// Проверяем правильность округления до двух значащих цифр

[TestMethod]

public void RoundToTwoSignificantDigits\_RoundsCorrectly()

{

// Arrange

double value1 = 123.456;

double value2 = 0.0123456;

double expected1 = 120;

double expected2 = 0.012;

// Act

double result1 = Calculator.RoundToTwoSignificantDigits(value1, 2);

double result2 = Calculator.RoundToTwoSignificantDigits(value2, 2);

// Assert

Assert.AreEqual(expected1, result1);

Assert.AreEqual(expected2, result2);

}

// Проверяем вычисление площади, когда секущая линия вне окружности

[TestMethod]

public void CalculateAnalyticArea\_HorizontalCut\_ReturnsFullCircleWhenLineOutside()

{

// Arrange

var circle = new Circle(new Point(0, 0), 5, 10);

double expected = Math.PI \* 25;

// Act

double result = Calculator.CalculateAnalyticArea(circle);

// Assert

Assert.AreEqual(expected, result, 0.0001);

}

// Проверяем вычисление площади, когда секущая линия проходит через центр окружности

[TestMethod]

public void CalculateAnalyticArea\_VerticalCut\_ReturnsHalfCircleWhenLineThroughCenter()

{

// Arrange

var circle = new Circle(new Point(0, 0), 5, 0);

double expected = Math.PI \* 25 / 2;

// Act

double result = Calculator.CalculateAnalyticArea(circle);

// Assert

Assert.AreEqual(expected, result, 0.0001);

}

// Проверяем правильность вычисление площади большей секции окружности разделённой секущей линией

// параллельной оси OX

[TestMethod]

public void CalculateAnalyticArea\_HorizontalCut\_ReturnsCorrectSegmentArea()

{

// Arrange

var circle = new Circle(new Point(0, 0), 5, 3);

double circleArea = Math.PI \* 25;

double segmentArea = 25 \* Math.Acos(3.0 / 5.0) - 3 \* Math.Sqrt(25 - 9);

double expected = circleArea - segmentArea;

// Act

double result = Calculator.CalculateAnalyticArea(circle);

// Assert

Assert.AreEqual(expected, result, 0.0001);

}

// Проверяем рекцию вычисления площади при некорректных значениях

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void CalculateAnalyticArea\_ThrowsForZeroRadius()

{

// Arrange

var circle = new Circle(new Point(0, 0), 0, 0);

// Act

Calculator.CalculateAnalyticArea(circle);

}

}

}

Код класса Circle.cs

namespace Monte\_Karlo.Models

{

// Класс, представляющий окружность с центром, радиусом и параметром C

public class Circle

{

// Центр окружности (координаты x, y)

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

// Радиус окружности (по умолчанию 2)

public float radius = 2;

// Дополнительный параметр C (по умолчанию -2)

public float C = -2;

// Конструктор по умолчанию - инициализирует значения по умолчанию

public Circle() { }

// Конструктор с параметрами для установки центра, радиуса и параметра C

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

{

this.circleCenter = circleCenter;

this.radius = radius;

this.C = c;

}

// Переопределение метода Equals для сравнения двух объектов Circle

// Сравнивает по координатам центра, радиусу и параметру C

public override bool Equals(object obj)

{

return obj is Circle other &&

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

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

radius == other.radius &&

C == other.C;

}

// Переопределение GetHashCode для корректной работы с хэш-таблицами и словарями

// Использует встроенный метод HashCode.Combine для создания хэш-кода на основе всех полей

public override int GetHashCode()

{

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

}

// Переопределение ToString для удобного вывода информации об окружности в виде строки

public override string ToString()

{

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

}

}

}

Код класса CircleParams.cs

namespace Monte\_Karlo.Models

{

// Класс, представляющий параметры окружности и связанные с ней результаты моделирования

public class CircleParams

{

// Атрибут [Key] указывает, что это первичный ключ для базы данных

public int Id { get; set; }

// Общее количество точек, используемых в эксперименте или симуляции

public int TotalPoints { get; set; }

// Аналитический (теоретический) результат, рассчитанный для окружности

public double AnalyticalResult { get; set; }

// Коллекция результатов моделирования (список объектов SimulationResult)

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

// Переопределённый метод ToString для удобного вывода основных свойств объекта

public override string ToString()

{

return $"""

Id: {Id},

TotalPoints: {TotalPoints},

AnalyticalResult: {AnalyticalResult},

ResultsCount: {Results.Count}

""";

}

}

}

Код класса CircleParamsTests.cs

namespace TestProject.Models

{

[TestClass]

public class CircleParamsTests

{

// Проверяет, что конструктор по умолчанию инициализирует коллекцию Results

[TestMethod]

public void CircleParams\_DefaultConstructor\_InitializesResultsList()

{

// Arrange & Act

var circleParams = new CircleParams();

// Assert

Assert.IsNotNull(circleParams.Results);

Assert.AreEqual(0, circleParams.Results.Count);

}

// Проверяет корректность установки и получения значений всех свойств класса

[TestMethod]

public void CircleParams\_Properties\_CanBeSetAndGet()

{

// Arrange

var circleParams = new CircleParams

{

Id = 1,

TotalPoints = 10000,

AnalyticalResult = 78.54

};

// Act & Assert

Assert.AreEqual(1, circleParams.Id);

Assert.AreEqual(10000, circleParams.TotalPoints);

Assert.AreEqual(78.54, circleParams.AnalyticalResult);

}

// Проверяет возможность добавления элементов в коллекцию Results

[TestMethod]

public void CircleParams\_Results\_CanBeAdded()

{

// Arrange

var circleParams = new CircleParams();

var result = new SimulationResult { Id = 1 };

// Act

circleParams.Results.Add(result);

// Assert

Assert.AreEqual(1, circleParams.Results.Count);

Assert.AreEqual(1, circleParams.Results[0].Id);

}

// Проверяет, что метод ToString возвращает строку с ожидаемым форматом и данными

[TestMethod]

public void CircleParams\_ToString\_ReturnsCorrectFormat()

{

// Arrange

var circleParams = new CircleParams

{

Id = 1,

TotalPoints = 10000,

AnalyticalResult = 78.54

};

// Act

var result = circleParams.ToString();

// Assert

StringAssert.Contains(result, "Id: 1");

StringAssert.Contains(result, "TotalPoints: 10000");

StringAssert.Contains(result, "AnalyticalResult: 78,54");

StringAssert.Contains(result, "ResultsCount: 0");

}

}

}

Код класса CircleTests.cs

namespace TestProject.Models

{

[TestClass]

public class CircleTests

{

// Проверяем конструкторы по умолчанию

[TestMethod]

public void Circle\_DefaultConstructor\_SetsDefaultValues()

{

// Arrange & Act

var circle = new Circle();

// Assert

Assert.AreEqual(-3, circle.circleCenter.X);

Assert.AreEqual(1, circle.circleCenter.Y);

Assert.AreEqual(2, circle.radius);

Assert.AreEqual(-2, circle.C);

}

// Проверяем конструкторы со своими значениями

[TestMethod]

public void Circle\_ParameterizedConstructor\_SetsCorrectValues()

{

// Arrange

var center = new Point(5, 5);

float radius = 10;

float c = 3;

// Act

var circle = new Circle(center, radius, c);

// Assert

Assert.AreEqual(center.X, circle.circleCenter.X);

Assert.AreEqual(center.Y, circle.circleCenter.Y);

Assert.AreEqual(radius, circle.radius);

Assert.AreEqual(c, circle.C);

}

// Тестируем метод Equals для сравнения объектов

[TestMethod]

public void Circle\_Equals\_ReturnsTrueForEqualCircles()

{

// Arrange

var circle1 = new Circle(new Point(1, 2), 3, 4);

var circle2 = new Circle(new Point(1, 2), 3, 4);

// Act & Assert

Assert.IsTrue(circle1.Equals(circle2));

}

// Тестируем метод Equals для сравнения объектов

[TestMethod]

public void Circle\_Equals\_ReturnsFalseForDifferentCircles()

{

// Arrange

var circle1 = new Circle(new Point(1, 2), 3, 4);

var circle2 = new Circle(new Point(1, 2), 3, 4);

var circle3 = new Circle(new Point(1, 3), 3, 4);

var circle4 = new Circle(new Point(1, 2), 4, 4);

var circle5 = new Circle(new Point(1, 2), 3, 5);

// Act & Assert

Assert.IsTrue(circle1.Equals(circle2));

Assert.IsFalse(circle1.Equals(circle3));

Assert.IsFalse(circle1.Equals(circle4));

Assert.IsFalse(circle1.Equals(circle5));

}

// Тестируем метод Equals на null

[TestMethod]

public void Circle\_Equals\_ReturnsFalseForNull()

{

// Arrange

var circle = new Circle();

// Act & Assert

Assert.IsFalse(circle.Equals(null));

}

// Проверяем GetHashCode для равных объектов

[TestMethod]

public void Circle\_GetHashCode\_ReturnsSameForEqualObjects()

{

// Arrange

var circle1 = new Circle(new Point(1, 2), 3, 4);

var circle2 = new Circle(new Point(1, 2), 3, 4);

// Act & Assert

Assert.AreEqual(circle1.GetHashCode(), circle2.GetHashCode());

}

// Тестируем ToString на корректность форматирования

[TestMethod]

public void Circle\_ToString\_ReturnsCorrectFormat()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

// Act

var result = circle.ToString();

// Assert

StringAssert.Contains(result, "CircleCenter: {X=1,Y=2}");

StringAssert.Contains(result, "Radius: 3");

StringAssert.Contains(result, "C: 4");

}

}

}

Код класса DatabaseHelper.cs

using Microsoft.EntityFrameworkCore;

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites;

namespace Monte\_Karlo.DataBase

{

public class DatabaseHelper

{

private Logger logger;

public DatabaseHelper()

{

logger = new Logger(); // Инициализация логгера

InitializeDatabase(); // Проверка или создание базы данных

}

// Инициализация базы данных

public void InitializeDatabase()

{

using var context = new AppDbContext();

if (context.Database.CanConnect()) // Проверка подключения

return;

context.Database.EnsureCreated(); // Создание базы данных при отсутствии

logger.Log("Создание базы данных");

}

// Полная очистка базы данных с пересозданием

public void ClearDatabase()

{

using (var context = new AppDbContext())

{

context.Database.EnsureDeleted(); // Удаление существующей базы

context.Database.EnsureCreated(); // Создание новой базы

}

logger.Log("Очистка базы данных");

}

// Сохранение результатов моделирования в базу данных

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

{

using var context = new AppDbContext();

int totalPoints = pointsData.Points.Count; // Общее количество точек

int pointsInSegment = pointsData.CuttedPoints.Count; // Количество точек в сегменте

// Попытка найти уже существующие параметры круга

var circleParams = context.CircleParams

.Include(cp => cp.Results)

.FirstOrDefault(cp =>

cp.TotalPoints == totalPoints);

// Если не найдено — создаём новый объект

if (circleParams == null)

{

circleParams = new CircleParams

{

TotalPoints = totalPoints,

AnalyticalResult = analyticalResult

};

context.CircleParams.Add(circleParams);

}

// Создание результата моделирования

var result = new SimulationResult

{

CircleParams = circleParams,

Points = totalPoints,

PointsInSegment = pointsInSegment,

MonteCarloResult = monteCarloResult

};

context.SimulationResults.Add(result); // Добавление в базу

context.SaveChanges(); // Сохранение изменений

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

}

// Получение данных по количеству точек

public CircleParams GetData(Circle circle, int totalPoints)

{

using var context = new AppDbContext();

var query = context.CircleParams

.Include(cp => cp.Results)

.Where(cp =>

cp.TotalPoints == totalPoints);

return query.FirstOrDefault();

}

// Получение данных по ID записи

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;

}

// Получение всех записей из базы

public List<CircleParams> GetAllData()

{

using var context = new AppDbContext();

var results = context.CircleParams

.Include(cp => cp.Results)

.ToList();

return results;

}

// Удаление записи по 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}");

}

//// Создание резервной копии файла 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);

//}

}

}

Код класса DatabaseHelperTests.cs

namespace TestProject

{

[TestClass]

[DoNotParallelize]

public class DatabaseHelperTests

{

private DatabaseHelper \_dbHelper;

[TestInitialize]

public void TestInitialize()

{

\_dbHelper = new DatabaseHelper();

}

[TestCleanup]

public void TestCleanup()

{

using var context = new AppDbContext();

context.Database.EnsureDeleted();

}

// Проверяет, что база данных инициализируется корректно

[TestMethod]

public void InitializeDatabase\_CreatesDatabaseIfNotExists()

{

// Arrange

using var context = new AppDbContext();

context.Database.EnsureCreated();

// Act

\_dbHelper.InitializeDatabase();

// Assert

Assert.IsTrue(context.Database.CanConnect());

}

// Проверяет, что метод ClearDatabase корректно очищает базу данных

[TestMethod]

public void ClearDatabase\_DeletesAndRecreatesDatabase()

{

// Arrange

\_dbHelper.InitializeDatabase();

// Act

\_dbHelper.ClearDatabase();

// Assert

using var context = new AppDbContext();

Assert.IsTrue(context.Database.CanConnect());

Assert.AreEqual(0, context.CircleParams.Count());

}

// Проверяет сохранение результатов в базу данных

[TestMethod]

public void SaveResults\_SavesNewCircleParamsAndResults()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

double analyticalResult = 28.2743;

double monteCarloResult = 28.26;

// Act

\_dbHelper.SaveResults(circle, pointsData, analyticalResult, monteCarloResult);

// Assert

using var context = new AppDbContext();

var savedParams = context.CircleParams.First();

var savedResult = context.SimulationResults.First();

Assert.AreEqual(1, context.CircleParams.Count());

Assert.AreEqual(1, context.SimulationResults.Count());

Assert.AreEqual(pointsData.Points.Count, savedParams.TotalPoints);

Assert.AreEqual(monteCarloResult, savedResult.MonteCarloResult);

}

// Проверяет, что при повторном сохранении с одинаковыми параметрами добавляется только новый результат

[TestMethod]

public void SaveResults\_AddsNewResultForExistingParams()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

// Первое сохранение

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.26);

// Act - второе сохранение с теми же параметрами

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.30);

// Assert

using var context = new AppDbContext();

Assert.AreEqual(1, context.CircleParams.Count(), string.Join("\n", context.CircleParams));

Assert.AreEqual(2, context.SimulationResults.Count(), string.Join("\n", context.SimulationResults));

}

// Проверяет получение данных по параметрам круга

[TestMethod]

public void GetData\_ReturnsCorrectCircleParams()

{

// Arrange

var circle = new Circle(new Point(0, 0), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.26);

// Act

var result = \_dbHelper.GetData(circle, 1000);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Results.Count, string.Join(", ", result.Results));

}

// Проверяет получение данных по ID

[TestMethod]

public void GetDataById\_ReturnsCorrectCircleParams()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.26);

int id = 1; // Первая запись обычно имеет ID = 1

// Act

var result = \_dbHelper.GetDataById(id);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(id, result.Id);

}

// Проверяет получение всех данных из базы

[TestMethod]

public void GetAllData\_ReturnsAllSavedRecords()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.26);

// Act

var results = \_dbHelper.GetAllData();

// Assert

Assert.AreEqual(1, results.Count);

}

// Проверяет удаление записи по ID

[TestMethod]

public void RemoveCircleParamsById\_DeletesRecord()

{

// Arrange

var circle = new Circle(new Point(1, 2), 3, 4);

var pointsData = new PointsData

{

Points = Enumerable.Repeat(new PointF(0, 0), 1000).ToList(),

IncludedPoints = Enumerable.Repeat(new PointF(0, 0), 785).ToList(),

CuttedPoints = Enumerable.Repeat(new PointF(0, 0), 500).ToList()

};

\_dbHelper.SaveResults(circle, pointsData, 28.2743, 28.26);

int id = 1;

// Act

\_dbHelper.RemoveCircleParamsById(id);

// Assert

using var context = new AppDbContext();

Assert.IsNull(context.CircleParams.Find(id));

Assert.AreEqual(0, context.SimulationResults.Count(), string.Join("\n", context.SimulationResults));

}

}

}

Код класса DataManagementForm.cs

using Monte\_Karlo.DataBase;

using Monte\_Karlo.Utilites;

using System.ComponentModel;

using System.Data;

namespace Monte\_Karlo.Forms

{

// Класс формы для управления экспериментальными данными

public partial class DataManagementForm : Form

{

// Объект для работы с базой данных

private DatabaseHelper databaseHelper = new DatabaseHelper();

// Объект для логирования ошибок и информации

private Logger logger = new Logger();

// Источник отмены для возможных асинхронных операций (не используется явно в коде)

private CancellationTokenSource \_cts;

// Конструктор формы

public DataManagementForm()

{

InitializeComponent(); // Инициализация компонентов формы

}

// Обработчик события загрузки формы

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

{

LoadExperiments(); // Загружаем данные экспериментов при старте формы

}

// Метод для загрузки списка экспериментов и отображения в таблице

private void LoadExperiments()

{

try

{

dgvExperiments.Columns.Clear(); // Очищаем столбцы в DataGridView

var data = databaseHelper.GetAllData(); // Получаем все данные из базы

var experiments = data

.Select(cp => new

{

ID = cp.Id, // Идентификатор эксперимента

Всего\_точек = cp.TotalPoints, // Общее количество точек в эксперименте

Аналитический\_результат = cp.AnalyticalResult, // Результат аналитического расчёта

Количество\_экспериментов = cp.Results.Count // Количество отдельных результатов внутри эксперимента

})

.ToList();

// Привязываем данные к таблице через BindingSource для удобства обновления

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

dgvExperiments.DataSource = bindingSource;

dgvExperiments.Columns["ID"].Visible = false; // Скрываем колонку с ID, чтобы не показывать пользователю

lblStatus.Text = $"Загружено экспериментов: {experiments.Count}"; // Обновляем статус внизу формы

}

catch (Exception ex)

{

// При ошибке загрузки показываем сообщение и логируем исключение

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

MessageBoxButtons.OK, MessageBoxIcon.Error);

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

}

}

// Обработчик кнопки "Удалить все данные"

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

{

// Спрашиваем подтверждение удаления всех данных

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(); // Очищаем всю базу данных

LoadExperiments(); // Обновляем таблицу

lblStatus.Text = "Все данные экспериментов удалены"; // Обновляем статус

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

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

// При ошибке удаления показываем сообщение и логируем

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

MessageBoxButtons.OK, MessageBoxIcon.Error);

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

}

}

}

// Обработчик кнопки "Удалить выбранный эксперимент"

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

{

// Если не выбран ни один эксперимент - предупреждаем пользователя

if (dgvExperiments.SelectedRows.Count == 0)

{

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

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

// Получаем ID выбранного эксперимента

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

// Спрашиваем подтверждение удаления выбранного эксперимента

if (MessageBox.Show($"Вы уверены, что хотите удалить все данные для эксперимента?",

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

{

try

{

databaseHelper.RemoveCircleParamsById(selectedId); // Удаляем данные по ID

LoadExperiments(); // Обновляем таблицу

lblStatus.Text = "Выбранный эксперимент удалён"; // Обновляем статус

}

catch (Exception ex)

{

// При ошибке удаления показываем сообщение и логируем

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

MessageBoxButtons.OK, MessageBoxIcon.Error);

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

}

}

}

// Обработчик кнопки "Анализ результатов"

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

{

// Проверяем, выбран ли эксперимент для анализа

if (dgvExperiments.SelectedRows.Count == 0)

{

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

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

// Получаем ID выбранного эксперимента

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

// Получаем данные эксперимента из базы

var circleParam = databaseHelper.GetDataById(selectedId);

// Открываем форму анализа с этими данными

var form = new AnalysisForm(circleParam);

form.ShowDialog();

}

// Обработчик клика по заголовку столбца для сортировки

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

{

// Если таблица пустая - выходим

if (dgvExperiments.RowCount == 0)

return;

// Определяем столбец, по которому кликнули

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

// Определяем направление сортировки (переключаем между Asc и Desc)

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

ListSortDirection.Descending :

ListSortDirection.Ascending;

// Вызываем сортировку данных по выбранному столбцу и направлению

SortData(column.Name, direction);

// Сбрасываем все иконки сортировки на None

dgvExperiments.Columns.Cast<DataGridViewColumn>()

.ToList()

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

// Устанавливаем иконку сортировки у выбранного столбца

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

SortOrder.Ascending :

SortOrder.Descending;

}

// Метод сортировки данных в таблице

private void SortData(string columnName, ListSortDirection direction)

{

// Проверяем, что источник данных является BindingSource

if (dgvExperiments.DataSource is BindingSource bindingSource)

{

// Получаем текущий список данных

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

// Сортируем по нужному столбцу и направлению

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:

// По умолчанию сортируем по ID

bindingSource.DataSource = direction == ListSortDirection.Ascending ?

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

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

break;

}

}

}

// Обработчик закрытия формы

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

{

// Отменяем и освобождаем ресурсы CancellationTokenSource, если он используется

\_cts?.Cancel();

\_cts?.Dispose();

\_cts = null;

}

}

}

Код класса Logger.cs

namespace Monte\_Karlo.Utilites

{

public class Logger

{

// Путь к папке с логами — в папке "Logs" рядом с исполняемым файлом приложения

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

// Метод записи общего лог-сообщения в файл с текущей датой

public void Log(string message)

{

// Путь к файлу лога для текущей даты, формат: "yyyy-MM-dd.log"

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

try

{

// Создаем папку Logs, если ее нет

if (!Directory.Exists(logDir))

Directory.CreateDirectory(logDir);

// Формируем строку с временной меткой (часы:минуты:секунды) и сообщением

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

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

// Добавляем строку в файл с новой строкой

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

// Также выводим сообщение в отладочный вывод (Visual Studio Output)

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

}

catch (Exception ex)

{

// Если произошла ошибка при логировании, выводим сообщение об ошибке

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

// Пытаемся зафиксировать ошибку логирования в отдельном лог-файле

LogException(ex, message);

}

}

// Метод записи исключений (ошибок) в отдельный лог

// Можно передать дополнительное сообщение

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

{

// Сначала выводим сообщение об ошибке и стек вызовов в отладочный вывод

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

// Путь к файлу логов ошибок (с меткой "error")

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

try

{

// Создаем папку Logs, если ее нет

if (!Directory.Exists(logDir))

Directory.CreateDirectory(logDir);

// Формируем строку с временной меткой и полной информацией об исключении

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

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

// Добавляем строку в файл логов ошибок

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

// И выводим ее в отладочный вывод

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

}

catch (Exception ex)

{

// Если и здесь ошибка, выводим ее в отладочный вывод, чтобы не терять информацию

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

}

}

}

}

Код класса LoggerTests.cs

using File = System.IO.File;

namespace TestProject

{

[TestClass]

[DoNotParallelize]

public class LoggerTests

{

private Logger \_logger;

private string \_testLogDir;

[TestInitialize]

public void TestInitialize()

{

\_logger = new Logger();

\_testLogDir = Path.Combine(Application.StartupPath, "Logs");

// Очищаем директорию с логами перед каждым тестом

if (Directory.Exists(\_testLogDir))

{

Directory.Delete(\_testLogDir, true);

}

}

[TestCleanup]

public void TestCleanup()

{

// Удаляем тестовую директорию после каждого теста

if (Directory.Exists(\_testLogDir))

{

Directory.Delete(\_testLogDir, true);

}

}

// Проверяет, что метод Log создает директорию для логов, если она не существует

[TestMethod]

public void Log\_CreatesLogDirectoryIfNotExists()

{

// Arrange

string testMessage = "Test log message";

// Act

\_logger.Log(testMessage);

// Assert

Assert.IsTrue(Directory.Exists(\_testLogDir), "Log directory should be created");

}

// Проверяет, что метод Log создает файл лога с текущей датой в имени

[TestMethod]

public void Log\_CreatesLogFileWithCurrentDate()

{

// Arrange

string testMessage = "Test log message";

string expectedFileName = DateTime.Now.ToString("yyyy-MM-dd") + ".log";

string expectedFilePath = Path.Combine(\_testLogDir, expectedFileName);

// Act

\_logger.Log(testMessage);

// Assert

Assert.IsTrue(File.Exists(expectedFilePath), "Log file should be created with current date");

}

// Проверяет, что метод Log добавляет сообщение в файл лога с правильным форматом

[TestMethod]

public void Log\_WritesMessageWithTimestamp()

{

// Arrange

string testMessage = "Test log message";

string expectedFileName = DateTime.Now.ToString("yyyy-MM-dd") + ".log";

string expectedFilePath = Path.Combine(\_testLogDir, expectedFileName);

// Act

\_logger.Log(testMessage);

// Assert

string logContent = File.ReadAllText(expectedFilePath);

StringAssert.Contains(logContent, DateTime.Now.ToString("HH:mm:ss"));

StringAssert.Contains(logContent, testMessage);

}

// Проверяет, что метод LogException создает файл ошибки с правильным именем

[TestMethod]

public void LogException\_CreatesErrorLogFile()

{

// Arrange

var exception = new Exception("Test exception");

string testMessage = "Test error message";

string expectedFileName = DateTime.Now.ToString("error") + ".log";

string expectedFilePath = Path.Combine(\_testLogDir, expectedFileName);

// Act

\_logger.LogException(exception, testMessage);

// Assert

Assert.IsTrue(File.Exists(expectedFilePath), "Error log file should be created");

}

// Проверяет, что метод LogException записывает полную информацию об исключении

[TestMethod]

public void LogException\_WritesFullExceptionDetails()

{

// Arrange

var exception = new Exception("Test exception");

string testMessage = "Test error message";

string expectedFileName = "error.log";

string expectedFilePath = Path.Combine(\_testLogDir, expectedFileName);

// Act

\_logger.LogException(exception, testMessage);

// Assert

string logContent = File.ReadAllText(expectedFilePath);

StringAssert.Contains(logContent, exception.Message);

StringAssert.Contains(logContent, testMessage);

}

// Проверяет, что метод Log обрабатывает ошибки записи в лог

[TestMethod]

public void Log\_HandlesWriteErrorsGracefully()

{

// Arrange

string testMessage = "Test log message";

// Создаем директорию без прав на запись

Directory.CreateDirectory(\_testLogDir);

File.SetAttributes(\_testLogDir, FileAttributes.ReadOnly);

// Act

try

{

\_logger.Log(testMessage);

// Assert

// Если мы дошли сюда, значит исключение не было брошено

Assert.IsTrue(true);

}

finally

{

File.SetAttributes(\_testLogDir, FileAttributes.Normal);

}

}

// Проверяет, что метод LogException обрабатывает ошибки записи в лог

[TestMethod]

public void LogException\_HandlesWriteErrorsGracefully()

{

// Arrange

var exception = new Exception("Test exception");

string testMessage = "Test error message";

// Создаем директорию без прав на запись

Directory.CreateDirectory(\_testLogDir);

File.SetAttributes(\_testLogDir, FileAttributes.ReadOnly);

// Act

try

{

\_logger.LogException(exception, testMessage);

// Assert

// Если мы дошли сюда, значит исключение не было брошено

Assert.IsTrue(true);

}

finally

{

File.SetAttributes(\_testLogDir, FileAttributes.Normal);

}

}

}

}

Код класса MainForm.cs

using Monte\_Karlo.DataBase;

using Monte\_Karlo.Forms;

using Monte\_Karlo.Models;

using Monte\_Karlo.Utilites;

using Monte\_Karlo.Utilites.Calculators;

using Monte\_Karlo.Utilites.View;

using System.Diagnostics;

using System.Reflection;

namespace Monte\_Karlo

{

// Главная форма приложения

public partial class MainForm : Form

{

// Коэффициенты масштабирования и деления

private float cofficient = 2;

private float divisionScale = 0.5f;

// Модель круга, по которому будет проводиться расчёт

private Circle circle = new Circle();

// Количество точек для метода Монте-Карло

private int pointsCount = 100\_000;

// Служебные переменные и вспомогательные классы

private CancellationTokenSource \_generationCts;

private PointsGenerator \_pointsGenerator;

private MonteCarloView \_view;

private DatabaseHelper \_databaseHelper;

private Logger \_logger;

// Конструктор формы

public MainForm()

{

InitializeComponent();

\_pointsGenerator = new PointsGenerator();

\_databaseHelper = new DatabaseHelper();

\_view = new MonteCarloView();

\_logger = new Logger();

// Включаем двойную буферизацию для избежания мерцания

DoubleBuffered = true;

typeof(Panel).InvokeMember("DoubleBuffered",

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

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

InitializeControlPanel();

\_databaseHelper.InitializeDatabase();

\_logger.Log("Приложение запущено");

FirstCalculation();

}

// Инициализация элементов управления панели управления

private void InitializeControlPanel()

{

scaleTrackBar.Value = \_view.GridStep;

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

pointsCountUpdown.Value = pointsCount;

}

// Первоначальный запуск вычислений

private async void FirstCalculation()

{

await Task.Delay(100);

await MonteCarloCalculate(true);

}

// Отрисовка панели с результатами расчёта

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");

Thread.Sleep(100);

\_view.RenderToBuffer(

paintPanel,

e,

circle,

\_pointsGenerator.GetCurrentPoints()

);

}

base.OnPaint(e);

}

// Обработка изменения масштаба через трекбар

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

{

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

\_view.GridStep = scaleTrackBar.Value;

paintPanel.Invalidate();

}

// Изменение количества точек через numeric updown

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

{

pointsCount = (int)pointsCountUpdown.Value;

await MonteCarloCalculate(true);

}

// Кнопка запуска генерации точек

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

{

await MonteCarloCalculate(true);

}

// Основной метод вычисления площади методом Монте-Карло

private async Task MonteCarloCalculate(bool generateNewPoints)

{

if (this.Visible != true)

return;

\_generationCts?.Cancel();

\_generationCts = new CancellationTokenSource();

try

{

var token = \_generationCts.Token;

if (generateNewPoints)

{

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

}

else

{

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

}

if (token.IsCancellationRequested)

return;

paintPanel.Invalidate();

double realSquare = Calculator.CalculateAnalyticArea(circle);

var roundedRealSquare = Math.Round(realSquare, 6);

var currentPoints = \_pointsGenerator.GetCurrentPoints();

double monteCarloSquare = Calculator.CalculateMonteCarloArea(

circle.radius,

currentPoints.Points.Count,

currentPoints.CuttedPoints.Count);

var roundedMonteCarloSquare = Math.Round(monteCarloSquare, 6);

ShowAnswereMessage(realSquare, monteCarloSquare, currentPoints);

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

WriteResultOnLabels(roundedRealSquare, roundedMonteCarloSquare);

\_databaseHelper.SaveResults(

circle,

currentPoints,

realSquare,

monteCarloSquare);

}

catch (OperationCanceledException)

{

// Игнорируем отмену задачи

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Отображение исключения в виде окна сообщения

private void ShowException(Exception ex)

{

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

\_logger.LogException(ex);

}

// Вывод результатов расчётов в элементы управления

private void WriteResultOnLabels(double? realSquare, double monteCarloSquare)

{

if (realSquare.HasValue)

realSquareLabel.Text = $"Аналитически: {realSquare:F6}";

monteCarloSquareLabel.Text = $"Методом Монте-Карло: {monteCarloSquare:F6}";

}

// Показать сообщение с результатами расчёта, если включено отображение

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, 6);

double roundRelativeError = Math.Round(relativeError, 6);

double maxAccuracy = 1 / (double)pointsCount;

string message = $"""

Всего точек: {currentPoints.Points.Count}

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

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

---------------------------------------------------------------------

Площадь круга: {Calculator.CircleSuare(circle.radius):F8}

Площадь секции аналитически: {realSquare:F8}

Площадь секции методом Монте-Карло: {monteCarloSquare:F8}

---------------------------------------------------------------------

Абсолютаня погрешность вычислений: {roundAbsoluteError}

Относительная погрешность вычислений: {roundRelativeError}%

Максимальная точность при заданном количестве точек: {maxAccuracy}

""";

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

}

// Обработка закрытия формы

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

{

\_generationCts?.Cancel();

\_logger.Log("Приложение закрыто");

Application.Exit();

}

// Открытие файла справки

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

{

try

{

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

// Открываем справку в браузере по умолчанию

Process.Start(new ProcessStartInfo

{

FileName = helpFile,

UseShellExecute = true

});

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString(), $"Не удалось открыть справку");

}

}

// Открытие окна "О программе"

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

{

try

{

var form = new AboutProgramForm();

form.ShowDialog();

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Очистка всех точек и графика

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

{

try

{

\_pointsGenerator.ClearPoints();

paintPanel.Invalidate();

WriteResultOnLabels(null, 0);

\_logger.Log("Очищение точек");

}

catch (Exception ex)

{

ShowException(ex);

}

}

// Выход из приложения через меню

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

// Открытие формы анализа результатов

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);

}

}

// Обработка изменения размеров панели отрисовки

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

{

paintPanel.Invalidate();

}

// Открытие окна управления экспериментами

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

{

try

{

var form = new DataManagementForm();

form.ShowDialog();

}

catch (Exception ex)

{

ShowException(ex);

}

}

}

}

Код класса MonteCarloView.cs

using Monte\_Karlo.Models;

namespace Monte\_Karlo.Utilites.View

{

public class MonteCarloView

{

// Свойство для установки шага сетки.

// При изменении GridStep также пересчитывается \_step — масштаб для отрисовки

public int GridStep

{

get => \_gridStep;

set

{

\_gridStep = value;

\_step = \_gridStep \* 2; // Масштаб равен двойному шагу сетки

}

}

private int \_gridStep = 100; // Начальный шаг сетки в пикселях

private int \_step = 200; // Масштабный шаг (используется для позиционирования элементов)

private static readonly int \_viewPointsLimit = 1\_000\_000; // Максимальное количество точек для отрисовки

// Цвет фона панели (темно-серый)

private static readonly Color \_backgroundColor = Color.FromArgb(20, 20, 20); //Black

// Карандаши для рисования различных элементов

private static readonly Pen \_gridPen = new(Color.Gray, 1); // Сетка

private static readonly Pen \_axisPen = new(Color.FromArgb(200, 200, 200), 2); // Оси (светло-серый)

private static readonly Pen \_cutterPen = new(Color.Red, 4); // "Cutter" — красная вертикальная линия

private static readonly Pen \_circlePen = new(Color.Yellow, 2); // Окружность — желтая

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

private static readonly Pen \_cuttedPointsBrush = new(Color.LawnGreen, 1); // Точки (цвет газонно-зеленый)

private static readonly Color \_textColor = Color.FromArgb(200, 200, 200); // Цвет текста — светло-серый

private static readonly Brush \_textBrush = new SolidBrush(\_textColor); // Кисть для текста

private static readonly Font \_textFont = new("Arial", 8); // Шрифт для координат

// Основной метод отрисовки: очищает фон и вызывает внутренний метод OnPaint

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

{

e.Graphics.Clear(\_backgroundColor);

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

}

// Основная отрисовка всех элементов на панели

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

{

var g = e.Graphics;

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighSpeed; // Быстрая отрисовка без сглаживания

// Центр экрана (панели)

float centerX = panel.Size.Width / 2;

float centerY = panel.Size.Height / 2;

var centerScreen = new PointF(centerX, centerY);

// Верхний левый угол квадрата, который вписывает круг с данным радиусом

float squareX = centerX - radius \* \_step;

float squareY = centerY - radius \* \_step;

var squarePoint = new PointF(squareX, squareY);

// Проекция центра круга на экран (с учетом масштаба и направления осей)

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

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

var origin = new PointF(originX, originY);

// Отрисовка точек, сетки, осей, подписей, квадрата, окружности и линии "cutter"

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, C);

}

// Метод рисует сетку по заданному "origin" — точке с координатами (0,0) на экране

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

{

// Вертикальные линии сетки: слева и справа от origin.X

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 сетки: выше и ниже origin.Y

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 X и Y через центр (origin)

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

{

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

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

}

// Рисует числа координат по осям вокруг origin (точки (0,0))

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

{

// Числа по оси X слева от origin

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

{

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

if (digit == 0) // Не рисуем ноль здесь, чтобы не дублировать

continue;

string text = digit.ToString();

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

float textX = x - textSize.Width / 2; // Центрируем по горизонтали

float textY = origin.Y + 5; // Немного ниже оси X

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

{

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

}

}

// Числа по оси X справа от origin

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 по оси Y вверх от origin

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; // Немного правее оси Y

float textY = y - textSize.Height / 2; // Центрируем по вертикали

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

{

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

}

}

// Числа по оси Y вниз от origin

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 "0" в начале координат с отступом

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

}

// Проверка, что текст помещается полностью в пределах 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 квадрат по заданной точке и размеру

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

{

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

}

// Рисует окружность (эллипс с равными сторонами) по заданной точке и размеру

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

{

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

}

// Рисует вертикальную красную линию "cutter" по параметру C относительно origin

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

{

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

}

// Рисует точки из списка pointsData.CuttedPoints

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

{

if (pointsData.CuttedPoints.Count == 0)

return;

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

// Создаем массив маленьких прямоугольников размером 1x1 пиксель для каждой точки

var rectangles = new RectangleF[pointsToDraw];

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

{

var point = pointsData.CuttedPoints[i];

float screenX = center.X + point.X \* gridStep; // Перевод координаты X в экранную систему

float screenY = center.Y - point.Y \* gridStep; // Перевод координаты Y (обратите внимание на минус — экранная система Y вниз)

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

}

// Рисуем все точки одним вызовом

g.DrawRectangles(\_cuttedPointsBrush, rectangles);

}

}

}

Код класса PointsData.cs

namespace Monte\_Karlo.Models

{

// Класс для хранения и классификации наборов точек в формате PointF

public class PointsData

{

// Список всех точек (координаты с плавающей точкой)

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

// Список точек, которые были включены (например, удовлетворяют некоторому условию)

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

// Список точек, которые были исключены или отсечены (например, не подходят по условию)

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

}

}

Код класса PointsDataTests.cs

namespace TestProject.Models

{

[TestClass]

public class PointsDataTests

{

// Проверка инициализации коллекций

[TestMethod]

public void PointsData\_DefaultInitialization\_CollectionsAreEmpty()

{

// Arrange & Act

var data = new PointsData();

// Assert

Assert.AreEqual(0, data.Points.Count);

Assert.AreEqual(0, data.IncludedPoints.Count);

Assert.AreEqual(0, data.CuttedPoints.Count);

}

// Корректность добавления элементов в коллекции

[TestMethod]

public void PointsData\_AddPoints\_CollectionsContainItems()

{

// Arrange & Act

var data = new PointsData

{

Points = new List<PointF> { new PointF(1, 2) },

IncludedPoints = new List<PointF> { new PointF(3, 4) },

CuttedPoints = new List<PointF> { new PointF(5, 6) }

};

// Assert

Assert.AreEqual(1, data.Points.Count);

Assert.AreEqual(1, data.IncludedPoints.Count);

Assert.AreEqual(1, data.CuttedPoints.Count);

}

}

}

Код класса PointsGenerator.cs

using Monte\_Karlo.Models;

using System.Collections.Concurrent;

namespace Monte\_Karlo.Utilites

{

public class PointsGenerator

{

// Мьютекс для синхронизации доступа к \_currentPoints из разных потоков

private readonly Mutex \_mutex = new();

// Хранит текущий набор точек

private PointsData \_currentPoints = new();

/// <summary>

/// Асинхронно генерирует случайные точки внутри квадрата, ограниченного радиусом круга,

/// а затем вычисляет включённые и отсеянные точки.

/// </summary>

/// <param name="circle">Объект круга с параметрами</param>

/// <param name="count">Количество точек для генерации</param>

/// <param name="token">Токен отмены для прерывания операции</param>

/// <returns></returns>

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

{

try

{

// Захватываем мьютекс для исключительного доступа

\_mutex.WaitOne();

// Проверяем, не отменена ли операция

token.ThrowIfCancellationRequested();

// Создаем новый контейнер для точек

var newPoints = new PointsData();

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

float radius = circle.radius;

// Запускаем тяжелую работу в отдельном потоке

await Task.Run(() =>

{

var parallelOptions = new ParallelOptions

{

CancellationToken = token,

MaxDegreeOfParallelism = Environment.ProcessorCount

};

// Генерируем точки параллельно

GeneratePoints(newPoints, count, radius, parallelOptions);

token.ThrowIfCancellationRequested();

// Вычисляем точки, лежащие внутри круга

CalculateIncludedPoints(newPoints, radius, parallelOptions);

token.ThrowIfCancellationRequested();

// Вычисляем точки, отсеянные по дополнительному условию (с учётом параметра C круга)

CalculateCuttedPoints(newPoints, circle, parallelOptions);

}, token);

// Обновляем текущее состояние точек

\_currentPoints = newPoints;

}

finally

{

// Освобождаем мьютекс независимо от результата

\_mutex.ReleaseMutex();

}

}

/// <summary>

/// Асинхронно пересчитывает отсеянные точки для текущего набора,

/// либо вызывает генерацию точек, если их нет.

/// </summary>

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

{

try

{

\_mutex.WaitOne();

token.ThrowIfCancellationRequested();

// Если нет точек, генерируем их заново

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

{

await GenerateRandomPointsAsync(circle, count, token);

return;

}

// Иначе пересчитываем отсеянные точки на основе текущих данных

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;

}

/// <summary>

/// Генерирует заданное количество случайных точек внутри квадрата со стороной 2\*radius,

/// центрированного в начале координат.

/// </summary>

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

{

// Для каждого потока создаём свой экземпляр Random для избежания конфликтов

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

var points = new PointF[count];

// Параллельный цикл генерации точек

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

{

// Координаты по X и Y случайны в диапазоне [-radius, radius]

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();

}

/// <summary>

/// Вычисляет и сохраняет точки, лежащие внутри круга с данным радиусом.

/// </summary>

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

{

float radiusSquared = radius \* radius;

var includedPoints = new ConcurrentBag<PointF>();

// Параллельно проверяем расстояние от начала координат до каждой точки

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

{

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

// Если точка внутри круга (расстояние меньше радиуса), добавляем её в результат

if (distanceSquared < radiusSquared)

{

includedPoints.Add(point);

}

});

pointsData.IncludedPoints = includedPoints.ToList();

}

/// <summary>

/// Вычисляет "отсеянные" точки среди включённых, по дополнительному условию,

/// зависящему от параметра C круга.

/// </summary>

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

{

// Если нет включённых точек — выходим

if (pointsData.IncludedPoints.Count == 0)

return;

var cuttedPoints = new ConcurrentBag<PointF>();

Point center = circle.circleCenter;

float C = circle.C;

// Определяем направление отсечения в зависимости от C и центра круга

bool lefter = C < center.X;

float centerX = center.X;

// Параллельно фильтруем точки по условию отсечения

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

{

bool condition = lefter

? point.X + centerX >= C

: point.X + centerX <= C;

if (condition)

{

cuttedPoints.Add(point);

}

});

// Обновляем список отсечённых точек

pointsData.CuttedPoints.Clear();

pointsData.CuttedPoints = cuttedPoints.ToList();

}

}

}

Код класса Program.cs

using System.Diagnostics;

using System.Runtime.InteropServices;

namespace Monte\_Karlo

{

internal static class Program

{

// user32.dll

[DllImport("user32.dll")]

private static extern bool SetForegroundWindow(IntPtr hWnd);

// , STA (Single Thread Apartment)

[STAThread]

static void Main()

{

// ( , DPI . .)

ApplicationConfiguration.Initialize();

// , ( )

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

{

// ,

Process target = Process.GetProcessesByName(Process.GetCurrentProcess().ProcessName)[0];

IntPtr handle = target.MainWindowHandle;

// ,

// ( )

if (handle != IntPtr.Zero)

{

SetForegroundWindow(handle);

}

// ,

return;

}

// , SplashScreenForm

Application.Run(new SplashScreenForm());

}

}

}

Код класса SimulationResult.cs

using System.ComponentModel.DataAnnotations;

namespace Monte\_Karlo.Models

{

// Класс, представляющий результат одной симуляции Монте-Карло

public class SimulationResult

{

[Key]

// Уникальный идентификатор результата симуляции

public int Id { get; set; }

// Общее количество точек, использованных в симуляции

public int Points { get; set; }

// Количество точек, попавших в интересующий сегмент (например, в круг)

public int PointsInSegment { get; set; }

// Результат оценки методом Монте-Карло (например, отношение PointsInSegment к Points)

public double MonteCarloResult { get; set; }

// Внешний ключ на параметры круга, к которым относится данный результат симуляции

public int CircleParamsId { get; set; }

// Навигационное свойство для связи с параметрами круга

public CircleParams CircleParams { get; set; }

// Переопределение метода ToString для удобного вывода информации о результате симуляции

public override string ToString()

{

return $"""

Id: {Id}, Points: {Points}, PointsInSegment: {PointsInSegment},

MonteCarloResult: {MonteCarloResult}, CircleParamsId: {CircleParamsId}

""";

}

}

}

Код класса SimulationResultTests.cs

namespace TestProject.Models

{

[TestClass]

public class SimulationResultTests

{

// Проверяет корректность установки и получения значений всех свойств класса

[TestMethod]

public void SimulationResult\_Properties\_CanBeSetAndGet()

{

// Arrange

var result = new SimulationResult

{

Id = 1,

Points = 100,

PointsInSegment = 50,

MonteCarloResult = 3.14,

CircleParamsId = 2

};

// Act & Assert

Assert.AreEqual(1, result.Id);

Assert.AreEqual(100, result.Points);

Assert.AreEqual(50, result.PointsInSegment);

Assert.AreEqual(3.14, result.MonteCarloResult);

Assert.AreEqual(2, result.CircleParamsId);

}

// Проверяет возможность установки и получения связанного объекта CircleParams

[TestMethod]

public void SimulationResult\_CircleParams\_CanBeSet()

{

// Arrange

var result = new SimulationResult();

var circleParams = new CircleParams { Id = 1 };

// Act

result.CircleParams = circleParams;

// Assert

Assert.IsNotNull(result.CircleParams);

Assert.AreEqual(1, result.CircleParams.Id);

}

// Проверяет, что метод ToString возвращает строку с ожидаемым форматом и данными

[TestMethod]

public void SimulationResult\_ToString\_ReturnsCorrectFormat()

{

// Arrange

var result = new SimulationResult

{

Id = 1,

Points = 100,

PointsInSegment = 50,

MonteCarloResult = 3.14,

CircleParamsId = 2

};

// Act

var str = result.ToString();

// Assert

StringAssert.Contains(str, "Id: 1");

StringAssert.Contains(str, "Points: 100");

StringAssert.Contains(str, "PointsInSegment: 50");

StringAssert.Contains(str, "MonteCarloResult: 3,14");

StringAssert.Contains(str, "CircleParamsId: 2");

}

}

}

Код класса SplashScreenForm.cs

namespace Monte\_Karlo

{

public partial class SplashScreenForm : Form

{

private int \_time = 0; // ,

private int \_timeout = 3; // ( ),

public SplashScreenForm()

{

InitializeComponent(); //

}

//

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

{

var form = new MainForm(); //

form.Show(); //

this.Hide(); // ( )

}

//

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

{

timer1.Start(); //

}

//

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

{

if (++\_time >= \_timeout) //

{

timer1.Stop(); //

startButton.PerformClick(); //

}

}

}

}

Код класса StatisticCalculator.cs

namespace Monte\_Karlo.Utilites.Calculators

{

// Статический класс для базовых статистических вычислений

public static class StatisticCalculator

{

// Метод для вычисления медианы списка чисел (центрального значения)

public static double CalculateMedian(List<double> values)

{

var sorted = values.OrderBy(x => x).ToList(); // Сортируем входной список по возрастанию

int count = sorted.Count;

if (count % 2 == 0) // Если количество элементов чётное

// Возвращаем среднее арифметическое двух центральных элементов

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

else

// Если нечётное — возвращаем центральный элемент

return sorted[count / 2];

}

// Метод для вычисления размаха (range) — разницы между максимальным и минимальным значением

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

}

}

Код класса StatisticCalculatorTests.cs

namespace TestProject

{

[TestClass]

public class StatisticCalculatorTests

{

// Проверяет правильность вычисления медианы при нечётном количестве элементов

[TestMethod]

public void CalculateMedian\_ReturnsCorrectForOddCount()

{

// Arrange

var values = new List<double> { 1, 3, 5, 7, 9 };

double expected = 5;

// Act

double result = StatisticCalculator.CalculateMedian(values);

// Assert

Assert.AreEqual(expected, result);

}

// Проверяет правильность вычисления медианы при чётном количестве элементов

[TestMethod]

public void CalculateMedian\_ReturnsCorrectForEvenCount()

{

// Arrange

var values = new List<double> { 1, 3, 5, 7, 9, 11 };

double expected = 6;

// Act

double result = StatisticCalculator.CalculateMedian(values);

// Assert

Assert.AreEqual(expected, result);

}

// Проверяет правильность вычисления размаха

[TestMethod]

public void CalculateRange\_ReturnsDifferenceBetweenMaxAndMin()

{

// Arrange

var values = new List<double> { 1, 5, 3, 9, 2 };

double expected = 8;

// Act

double result = StatisticCalculator.CalculateRange(values);

// Assert

Assert.AreEqual(expected, result);

}

}

}

# ПРИЛОЖЕНИЕ В

**ОКНА ПРОГРАММЫ**

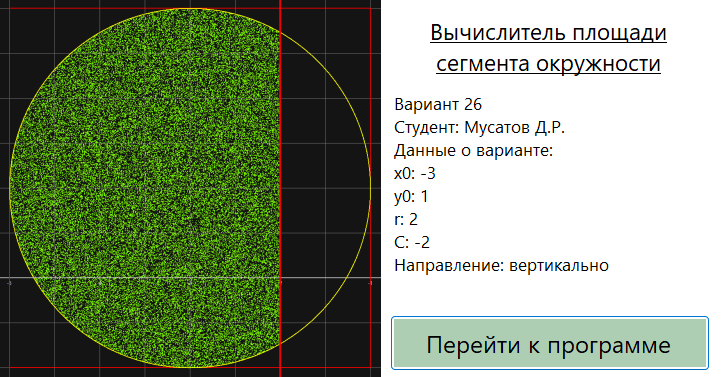


Рисунок В1 – Заставка программы

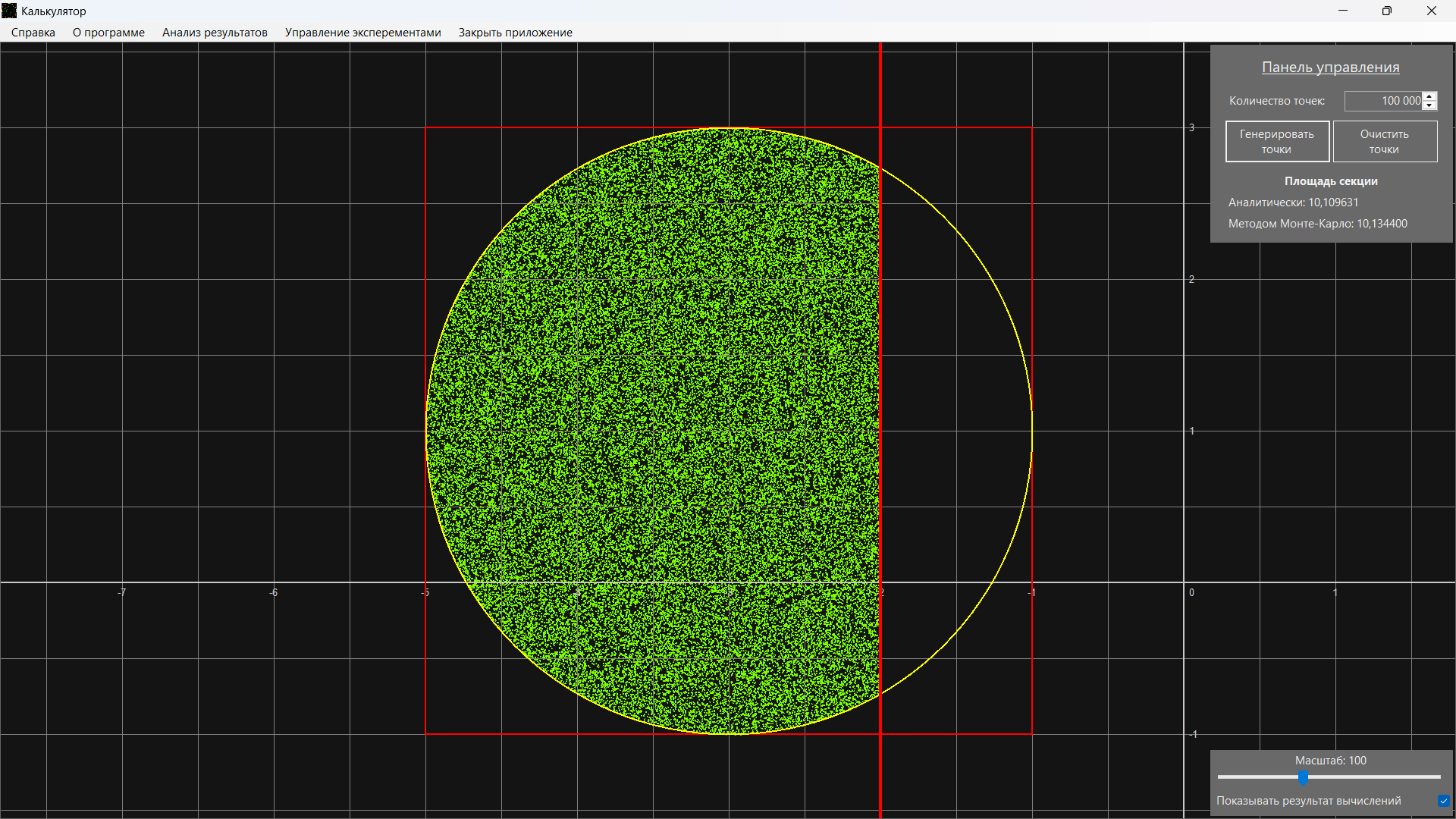


Рисунок В2 – Основное окно программы

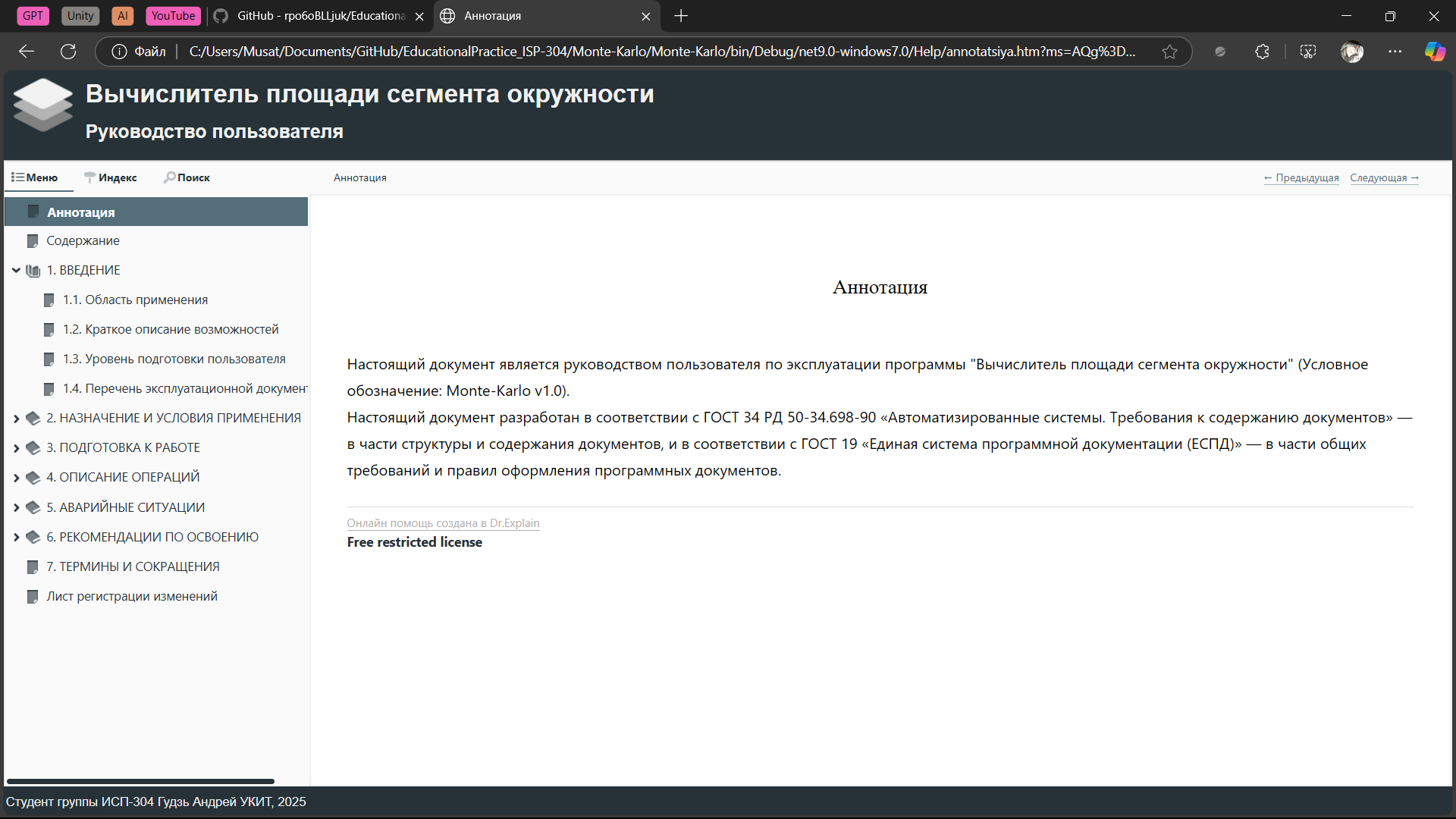


Рисунок В3 – справка, открываемая в браузере



Рисунок В4 – окно «О программе»

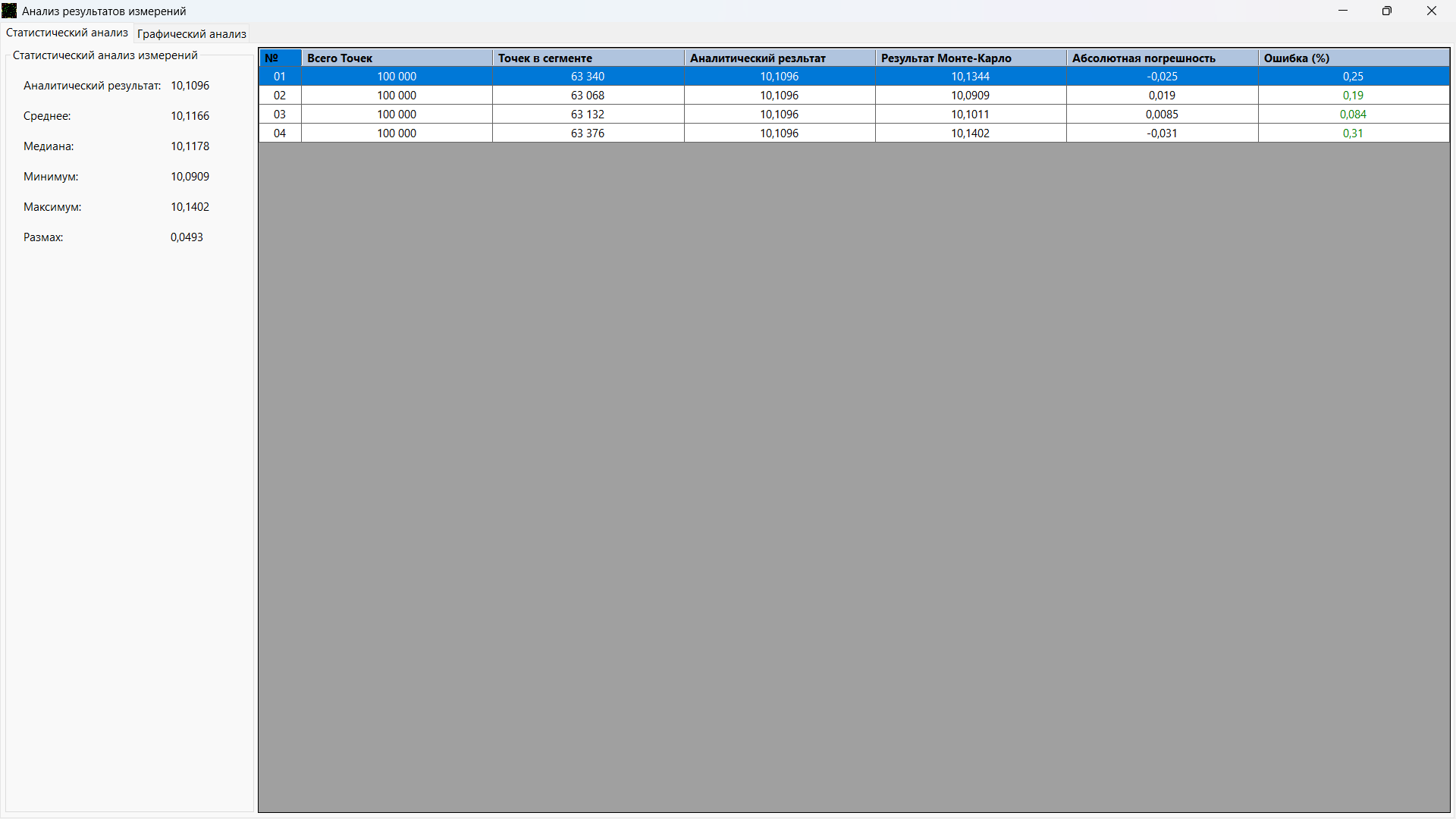


Рисунок В5 – вкладка статистического анализа в окне анализа результатов измерений

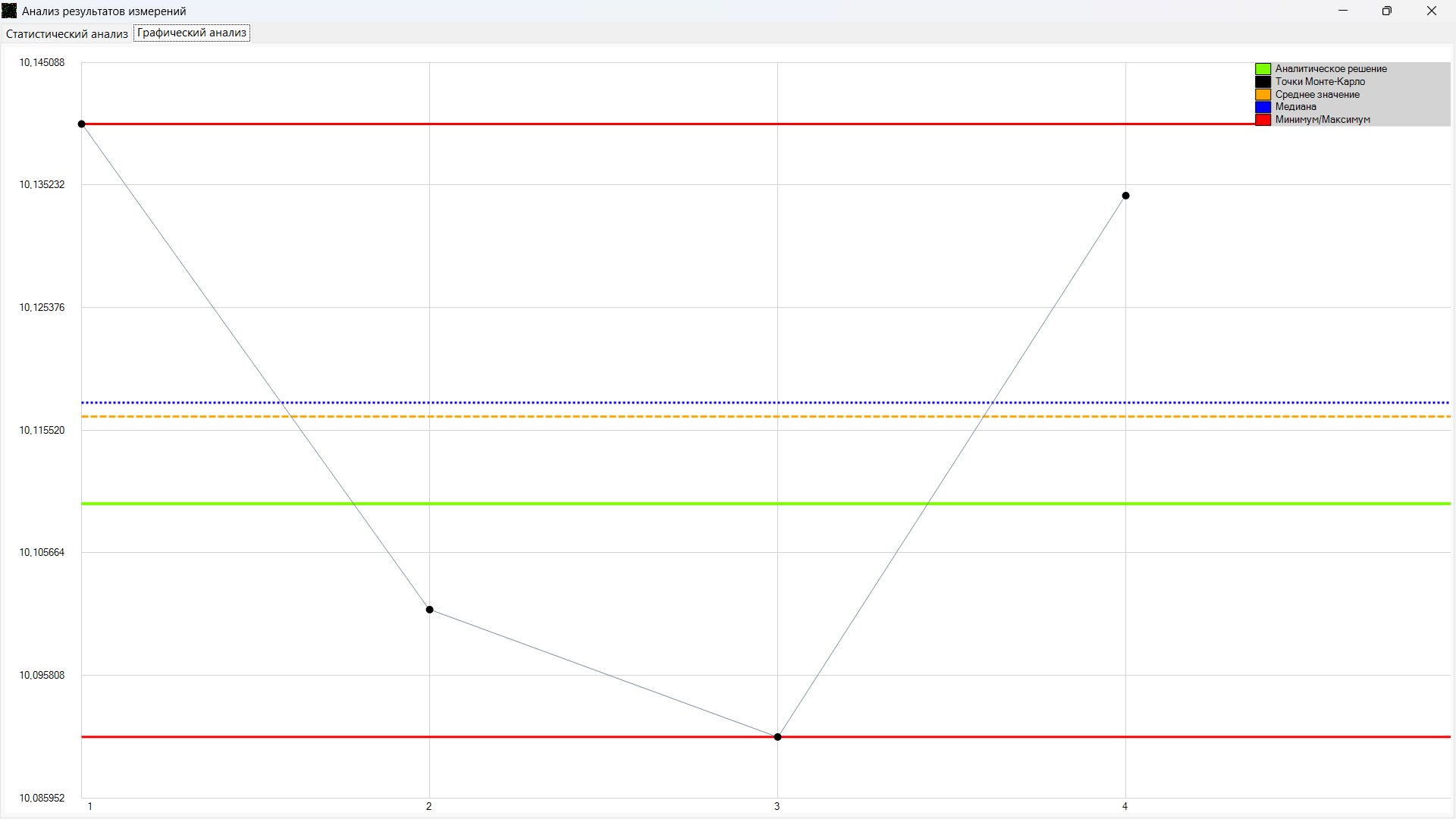


Рисунок В6 – вкладка графического анализа в окне анализа результатов измерений

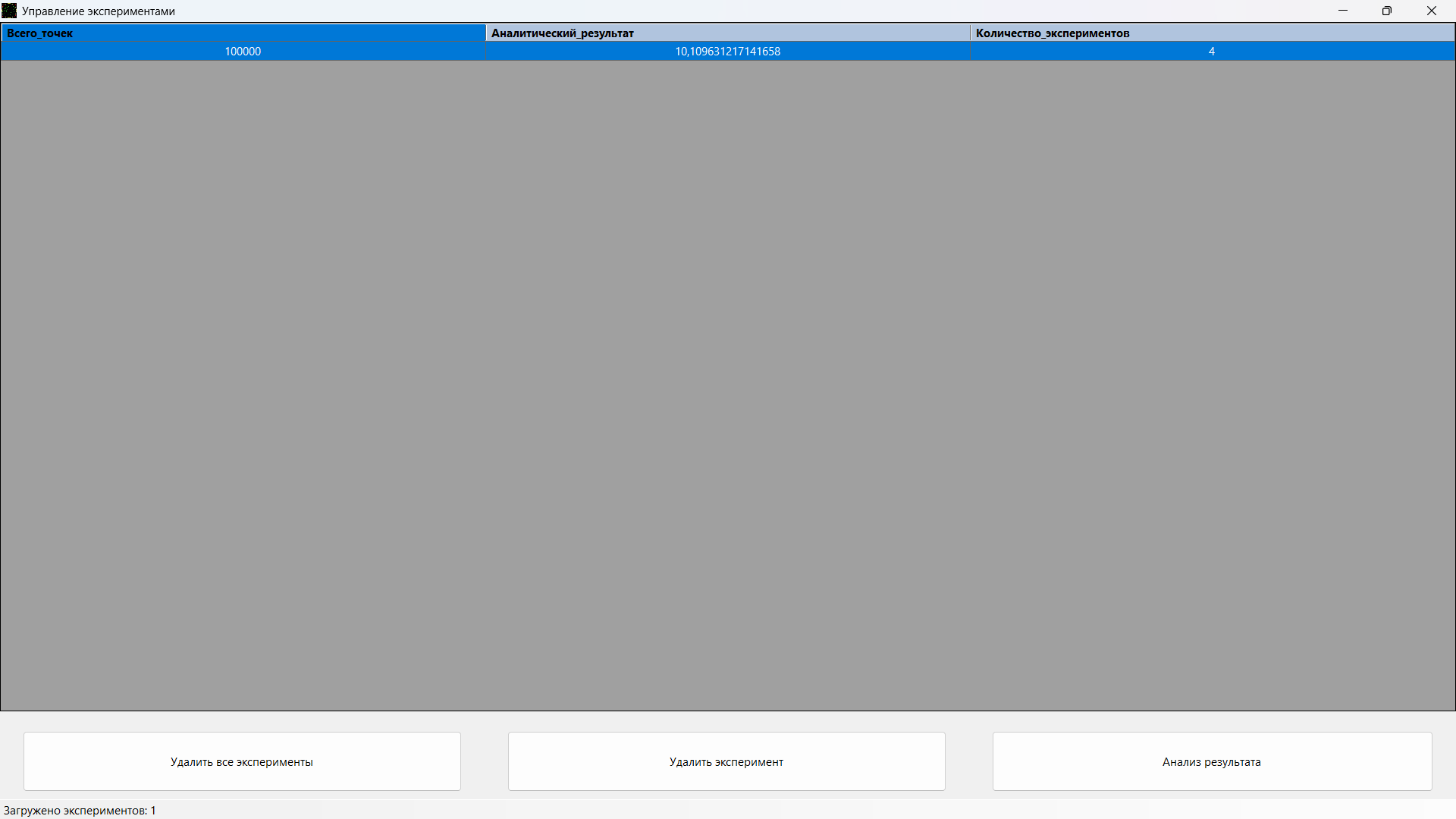


Рисунок В7 – окно управления экспериментами