

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Операционные системы и среды»

на тему: **«Изучение методов синхронизации в многопоточных приложениях»**

Исполнитель: студент гр. ИП-31
В.Г.Земченок

Руководитель: преподаватель
Г. П. Косинов

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____
Оценка работы: _____

Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2020

СОДЕРЖАНИЕ

Введение.....	3
1 Обзор существующих методов решения задачи.....	4
1.1 Понятие поверхности.....	4
1.2 Вычисление площади поверхности.....	6
1.3 Численное интегрирование методом правых прямоугольников	7
1.4 Используемые средства разработки.....	8
1.5 Распределение вычислений в <i>C# .NET</i>	11
1.6 Синхронизация потоков в <i>C# .NET</i>	16
1.7 Взаимодействие с пакетом <i>MS Excel</i>	18
2 Алгоритмический анализ.....	20
2.1 Постановка задачи, анализ входных и выходных данных.....	20
2.2 Построение функциональной модели.....	20
2.3 Применение метода численного интегрирования функции одной переменной при интегрировании функции двух переменных.....	21
2.4 Абсолютная и относительная погрешности.....	22
3 Разработка программного кода.....	23
3.1 Реализация графического интерфейса.....	23
3.2 Реализация вычислительных алгоритмов.....	25
3.3 Реализация построения поверхности	27
3.4 Реализация экспорта статистики	28
3.5 Реализация подсчёта погрешности вычислений.....	30
3.6 Реализация дополнительных структур	31
3.7 Описание параметров используемых методов и функций	31
4 Тестирование и верификация.....	33
4.1 Тестирование <i>GUI</i>	33
4.2 Негативное тестирование.....	34
4.3 Анализ факторов, влияющих на время вычислений	36
Заключение	38
Список использованных источников	39
Приложение А Исходный код приложения.....	40
Приложение Б Блок-схема алгоритма метода правых прямоугольников	55
Приложение В Блок-схема алгоритма интегрирования.....	56

ВВЕДЕНИЕ

Реалии современного мира таковы, что разработка практически любого программного обеспечения требует хороших знаний параллельного и распределённого программирования. Обе эти области объединяет то, что и параллельное, и распределённое программное обеспечение состоит из нескольких процессов, которые вместе решают одну общую задачу.

Под многопоточностью понимают возможность параллельно выполнять несколько видов операций в одной прикладной программе. Программа, разработанная с использованием механизма потоков, представляемая как некоторое множество задач в рамках одного процесса, может быть выполнена быстрее за счет параллельного функционирования её отдельных частей. Особенно это выгодно при наличии нескольких процессоров, ибо каждая задача может выполняться на отдельном процессоре. Также, особенно эффективно можно использовать многопоточность для выполнения распределенных приложений. Стоит отметить, что для совместной работы процессов им необходим некоторый механизм взаимодействия.

Целью данного курсового проекта является разработка компонентного приложения для обработки и отрисовки поверхности с использованием многопоточности.

Задачи данного курсового проекта:

- изучить теоретический материал о методах решения задачи;
- разработать функциональную модель;
- разработать необходимые программные компоненты;
- выполнить алгоритмический анализ поставленной задачи;
- исследовать эффективность организации многопоточности.

1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ

1.1 Понятие поверхности

Поверхность в пространстве, как правило, можно рассматривать как геометрическое место точек, удовлетворяющих какому-либо условию. Например, сфера радиуса R с центром в точке O есть геометрическое место всех точек пространства, находящихся от точки O на расстоянии R . Прямоугольная система координат $Oxyz$ в пространстве позволяет установить взаимно однозначное соответствие между точками пространства и тройками чисел x, y и z – их координатами. Свойство, общее всем точкам поверхности, можно записать в виде уравнения, связывающего координаты всех точек поверхности. Согласно [1], уравнением данной поверхности в прямоугольной системе координат $Oxyz$ называется уравнение вида

$$F(x, y, z) = 0 \quad (1)$$

с тремя переменными x, y и z , которому удовлетворяют координаты каждой точки, лежащей на поверхности, и не удовлетворяют координаты точек, не лежащих на этой поверхности.

Если функция $F(x, y, z)$ непрерывна в некоторой точке и имеет в ней непрерывные частные производные, по крайней мере одна из которых не обращается в нуль, то в окрестности этой точки поверхность, заданная уравнением (1), будет правильной поверхностью, заданной неявно.

Если одну из переменных, например, z , выразить через остальные, то способ задания будет явным:

$$z = f(x, y) \quad (2)$$

Случай неявного задания. Поверхность, заданная уравнением $F(x, y, z) = 0, F: \Omega \rightarrow R^3$, является гладкой регулярной поверхностью, если $\exists P_0(x_0, y_0, z_0) : F(x_0, y_0, z_0) = 0$, функция F непрерывно дифференцируема в своей области определения Ω , а её частные производные одновременно не обращаются в нуль (условие правильности) на всём множестве Ω :

$$\left(\frac{\partial F}{\partial x}\right)^2 + \left(\frac{\partial F}{\partial y}\right)^2 + \left(\frac{\partial F}{\partial z}\right)^2 > 0 \quad (3)$$

					КП 1-40 04 01.02	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Случай параметрического задания. Зададим поверхность векторным уравнением $r = r(u, v)$, или, что то же самое, тремя уравнениями в координатах:

$$\begin{cases} x = x(u, v) \\ y = y(u, v) \\ z = z(u, v) \end{cases} \quad (u, v) \in \Omega \quad (4)$$

Эта система уравнений задаёт гладкую регулярную поверхность, если выполнены условия:

- система устанавливает взаимно однозначное соответствие между образом и прообразом Ω ;
- функции $x(u, v), y(u, v), z(u, v)$ непрерывно дифференцируемы в Ω ;
- выполнено условие не вырожденности:

$$\begin{vmatrix} x'_u & x'_v \\ y'_u & y'_v \end{vmatrix}^2 + \begin{vmatrix} y'_u & y'_v \\ z'_u & z'_v \end{vmatrix}^2 + \begin{vmatrix} z'_u & z'_v \\ x'_u & x'_v \end{vmatrix}^2 > 0 \quad (5)$$

Геометрически последнее условие означает, что векторы $\frac{\partial r}{\partial u}, \frac{\partial r}{\partial v}$ нигде не параллельны. Параметры u, v можно рассматривать как внутренние координаты точек поверхности. Фиксируя одну из координат, мы получаем два семейства координатных кривых, покрывающих поверхность координатной сеткой.

Случай явного задания. Поверхность S может быть определена как график функции $z = f(x, y)$; тогда S является гладкой регулярной поверхностью, если функция f дифференцируема. Этот вариант можно рассматривать как частный случай параметрического задания:

$$x = u; y = v; z = f(u, v) [1].$$

Например, поверхность, заданная уравнением вида

$$z = \frac{x^2}{a^2} - \frac{y^2}{b^2}$$

с коэффициентами равными $a = 1$, $b = 1$ имеет отображение, представленное на рисунке 1.1.

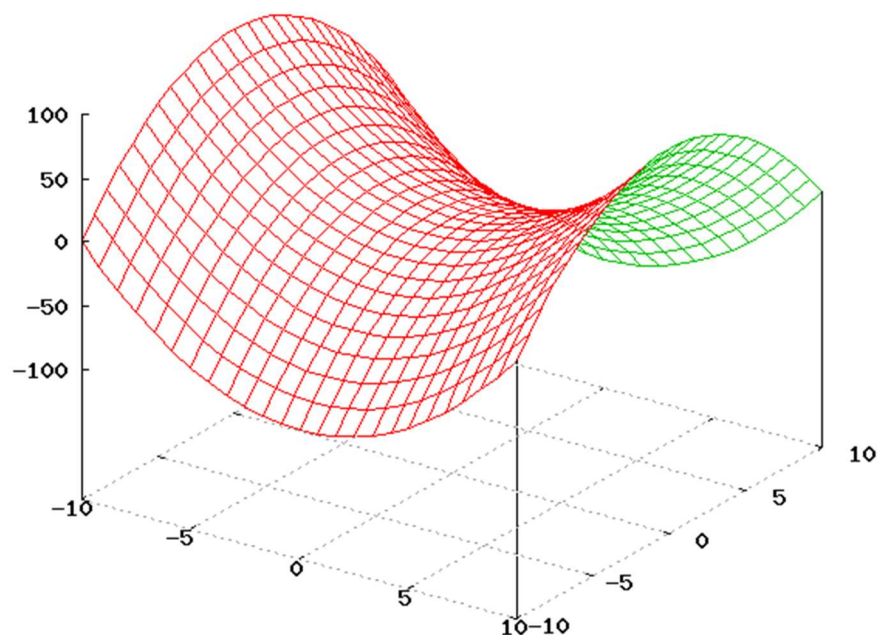


Рисунок 1.1 – Отображение гиперболического параболоида

1.2 Вычисление площади поверхности

Согласно [2], если везде в области D на координатной плоскости для формулы

$$I = \iint_D f(x, y) dx dy \quad (6)$$

положить $f(x, y) \equiv 1$, то, в соответствии со своим геометрическим смыслом, двойной интеграл будет численно равен площади S области интегрирования D , то есть

$$S = \iint_D dx dy \quad (7)$$

1.3 Численное интегрирование методом правых прямоугольников

Согласно [3], если функция $f(x)$ непрерывна на отрезке $[a; b]$ и требуется вычислить интеграл $I = \int_a^b f(x)dx$, численно равный площади соответствующей криволинейной трапеции. Основание этой трапеции, т.е. отрезок $[a; b]$ разбивается на n равных частей (отрезков): $[a = x_0; x_1], [x_1; x_2], [x_2; x_3], \dots, [x_{n-1}; x_n = b]$ длины $hx = \frac{b-a}{n}$ (шаг разбиения) с помощью точек $x_0 = a, x_1, x_2, \dots, x_n = b$. Можно записать, что $x_i = x_0 + h * i$, где $i = 1, 2, \dots, n$.

В конце каждого отрезка построим ординату $y_i = f(x_i)$. Приняв эту ординату за высоту, построим прямоугольник с площадью $hx * y_i$. Тогда сумма площадей всех n прямоугольников даёт площадь ступенчатой фигуры, представляющую собой приближённое значение искомого определённого интеграла:

$$I = \int_a^b f(x)dx \approx hx \sum_{i=1}^n f(x_i) \approx \frac{b-a}{n} \sum_{i=1}^n f(x_i) \quad (8)$$

Графически это имеет вид, изображённый на рисунке 1.2

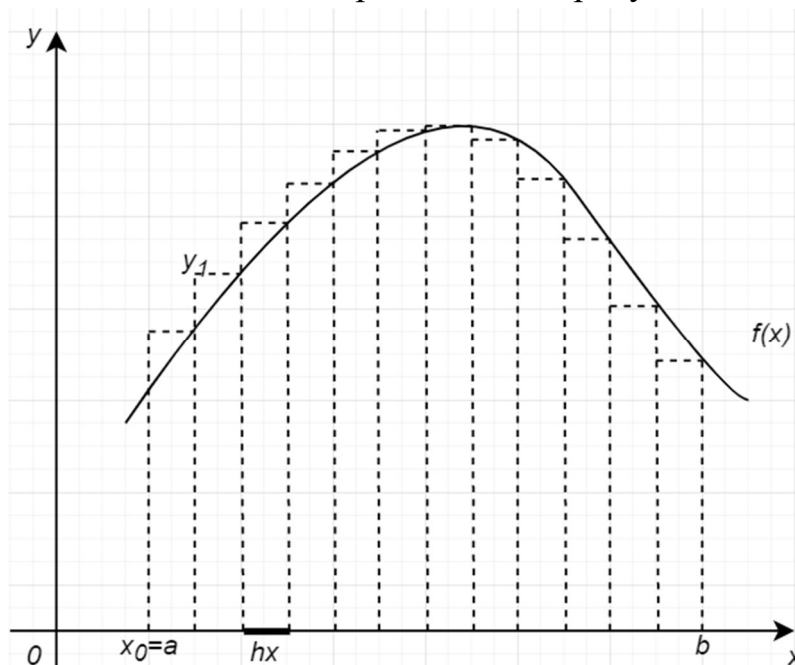


Рисунок 1.2 – Графическая интерпретация метода правых прямоугольников

Графическая схема метода правых прямоугольников приведена в Приложении Б на рисунке Б.1.

1.4 Используемые средства разработки

1.4.1 Выбор языка для написания приложения с главным окном проводился из двух языков: *C#* и *Java*. Графическая составляющая, в которой проводилось вычисление точного значения, реализована на языке программирования Python

1.4.2 Согласно [4], язык *Java*, разработанный компанией *Sun Microsystems*, создателем которого был Джеймс Гослинг, был выпущен в 1995 году в качестве основных компонентов компании *Sun Microsystems – Java* платформ (*Java 1.0 [J2SE]*).

По состоянию на 2020 год последней версией *Java Standard Edition* является 14.1 (*Java SE*) – *Sun Microsystems* переименовала прежнюю версию *J2* и ввела новые: *Java SE*, *Java EE* и *Java ME*. Введение в программирование *Java* различных версий подтверждало знаменитый слоган компании «Напиши один раз, запускай везде».

Язык программирования *Java* объектно–ориентированный – в нём все является объектом. Дополнение может быть легко расширено, так как он основан на объектной модели. Он также платформонезависимый: в отличие от многих других языков, включая *C* и *C++*, *Java*, когда был создан, он не компилировался в платформе конкретной машины, а в независимом от платформы байт–коде. Этот байт код распространяется через интернет и интерпретируется в *Java Virtual Machine (JVM)*, на которой он в настоящее время работает.

Стоит также отметить его простоту: процессы изучения и введение в язык программирования *Java* остаются простыми. Достаточно понимать основные концепции объектно-ориентированного программирования – это сильно упростит освоение данного языка. *Java* – подходящий для разработки язык с точки зрения безопасности: методы проверки подлинности основаны на шифровании с открытым ключом.

Компилятор данного языка, написанный на *ANSI C* с чистой переносимостью, который является подмножеством *POSIX*, генерирует архитектурно-нейтральные объекты формата файла, что делает скомпилированный код исполняемым на многих процессорах, в том числе с наличием системы *Java Runtime* – это подчёркивает, что *Java* – архитектурно-нейтральный язык программирования.

Достойна внимания «прочность» упомянутого языка: он выполняет усилия, чтобы устранить ошибки в различных ситуациях, делая упор в основном на время компиляции, проверку ошибок и проверку во время выполнения. В нём также доступно многопоточное программирование – написание программ, которые могут выполнять множество задач одновременно.

Java байт-код переводится на лету в машинные инструкции и нигде не сохраняется, что делает процесс более быстрым и аналитическим, поскольку связывание происходит как дополнительное с небольшого веса процесса.

Нельзя не упомянуть *Just-In-Time* компилятор, позволяет получить высокую производительность, а также тот факт, что данный язык является динамическим: программирование на *Java* считается более динамичным, чем на *C* или *C++*, так как он предназначен для адаптации к меняющимся условиям. Программы могут выполнять обширное количество во время обработки информации, которая может быть использована для проверки и разрешения доступа к объектам на время выполнения.

1.4.3 Согласно [5], *C#* – элегантный, типобезопасный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде *.NET Framework*. С помощью языка *C#* можно создавать обычные приложения *Windows*, приложения "клиент-сервер", приложения баз данных и т. д. *Visual C#* предоставляет развитый редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и множество других средств, упрощающих разработку приложений на базе языка *C#* и *.NET Framework*.

Синтаксис *C#* очень выразителен, но прост в изучении. Все, кто знаком с языками *C*, *C++* или *Java* с легкостью узнают синтаксис с фигурными скобками, характерный для языка *C#*. Разработчики, знающие любой из этих языков, как правило, смогут добиться эффективной работы с языком *C#* за очень короткое время. Синтаксис *C#* делает проще то, что было сложно в *C++*, и обеспечивает мощные возможности, такие как типы значений *Nullable*, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти, чего нет в *Java*. *C#* поддерживает универсальные методы и типы, обеспечивая более высокий уровень безопасности и производительности, а также итераторы, позволяющие при реализации коллекций классов определять собственное поведение итерации, которое может легко использоваться в клиентском коде. Выражения *LINQ* делают строго типизированный запрос очень удобной языковой конструкцией [4].

Как объектно-ориентированный язык, C# поддерживает понятия инкапсуляции, наследования и полиморфизма. Все переменные и методы, включая метод *Main* – точку входа приложения – инкапсулируются в определения классов. Класс может наследовать непосредственно из одного родительского класса, но может реализовывать любое число интерфейсов. Для методов, которые переопределяют виртуальные методы в родительском классе, необходимо ключевое слово *override*, чтобы исключить случайное повторное определение. В языке C# структура похожа на облегченный класс: это тип, распределяемый в стеке, реализующий интерфейсы, но не поддерживающий наследование.

В дополнение к основным описанным объектно-ориентированным принципам, язык C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

- инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях;
- свойства, выступающие в роли методов доступа для закрытых переменных-членов;
- *LINQ*, предлагающий встроенные возможности запросов в различных источниках данных.

Если потребуется обеспечить взаимодействие с другим программным обеспечением *Windows*, таким как объекты *COM* или собственные библиотеки *DLL Win32*, в языке C# можно использовать процесс, который называется "*Interop*". Процесс *Interop* позволяет программам на C# выполнять практически любые действия, которые может выполнять исходное приложение на C++. Посредством него, можно взаимодействовать с *MS Excel*, создавая книги и листы отдельными объектами для их изменения. Язык C# поддерживает даже указатели и понятие "небезопасного" кода для тех случаев, когда прямой доступ к памяти имеет крайне важное значение.

Процесс построения C# по сравнению с C и C++ прост и является более гибким, чем в *Java*. Нет отдельных файлов заголовка, а методы и типы не требуется объявлять в определенном порядке. В исходном файле C# может быть определено любое число классов, структур, интерфейсов и событий.

Язык программирования C# упрощает параллельное выполнение кода. Будучи частью фреймворка *.NET*, он может работать с библиотекой параллельных задач *TPL (Task Parallel Library)*, основной функционал которой располагается в пространстве имен *System.Threading.Tasks*. Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер. Кроме того, упрощается сама работа по созданию новых потоков.

Поэтому начиная с *.NET* 4.0. рекомендуется использовать именно *TPL* и ее классы для создания многопоточных приложений, хотя стандартные средства и класс *Thread* по-прежнему находят широкое применение.

Нельзя не упомянуть, что в *Microsoft .NET Framework* 4.7.2 существует технология *Windows Presentation Foundation*, позволяющая разрабатывать оконные приложения для операционной системы *Windows*. Одним из главных её преимуществ – наличие широкого набора удобных инструментов для разработки *Windows* приложений, в том числе визуального редактора окон [5][6].

Таким образом остановимся на языке *C#*.

Для построения поверхности был выбран язык *Python*.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра *Python* минималистичен.

В то же время стандартная библиотека включает большой объём полезных функций [7][8].

Основными архитектурными чертами языка являются: динамическая типизация, автоматическое управление памятью, механизм обработки исключений, поддержка многопоточных вычислений и высокоуровневые структуры данных, разбиения программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Построения поверхности в этом языке программирования самая тривиальная задача, которая может быть. Для ее выполнения необходимо наличие:

- библиотеки *Matplotlib* версии не ниже 0.99;
- математической библиотеки *numpy*.

1.5 Распределение вычислений в *C# .NET*

1.5.1 В эпоху многоядерных машин, которые позволяют параллельно выполнять сразу несколько процессов, стандартных средств работы с потоками в *.NET* уже оказалось недостаточно. Поэтому во фреймворк *.NET* была добавлена библиотека параллельных задач *TPL* (*Task Parallel Library*), основной функционал которой располагается в пространстве имен *System.Threading.Tasks* [9]. Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер. Кроме того, упрощается сама работа по созданию новых потоков. Поэтому начиная с *.NET* 4.0. рекомендуется использовать именно *TPL* и ее классы для создания многопоточных

приложений, хотя стандартные средства и класс *Thread* по-прежнему находят широкое применение.

Данный класс в пространстве имён *System.Threading* определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Основные свойства класса:

- статическое свойство *CurrentContext*, позволяющее получить контекст, в котором выполняется поток;
- статическое свойство *CurrentThread*, возвращающее ссылку на выполняемый поток;
- свойство *IsAlive* указывает, работает ли поток в текущий момент;
- свойство *IsBackground* указывает, является ли поток фоновым;
- свойство *Name* содержит имя потока;
- свойство *Priority* хранит приоритет потока - значение перечисления *ThreadPriority*;
- свойство *ThreadState* возвращает состояние потока - одно из значений перечисления *ThreadState*.

Некоторые методы класса *Thread*:

- статический метод *GetDomain*, возвращающий ссылку на домен приложения;
- статический метод *GetDomainID*, возвращающий id домена приложения, в котором выполняется текущий поток;
- статический метод *Sleep*, останавливающий поток на определенное количество миллисекунд;
- метод *Abort*, уведомляющий среду CLR о том, что надо прекратить поток (однако прекращение работы потока происходит не сразу, а только тогда, когда это становится возможно);
- метод *Interrupt* прерывает поток на некоторое время;
- метод *Join* блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод;
- метод *Start* запускает поток.

Применение потоков *Thread* показано на рисунке 1.3.

```

class Program
{
    class myThread
    {
        Thread thread;

        //Конструктор получает имя функции и номер до которого ведется счет
        public myThread(string name, int num)
        {
            thread = new Thread(this.func);
            thread.Name = name;
            thread.Start(num); //передача параметра в поток
        }

        void func(object num) //функция потока, передаем параметр
        {
            for (int i = 0; i < (int)num; i++)
            {
                Console.WriteLine(Thread.CurrentThread.Name + " выводит " + i);
                Thread.Sleep(0);
            }
            Console.WriteLine(Thread.CurrentThread.Name + " завершился");
        }
    }

    static void Main(string[] args)
    {
        myThread t1 = new myThread("Thread 1", 6);
        myThread t2 = new myThread("Thread 2", 3);
        myThread t3 = new myThread("Thread 3", 2);
        Console.Read();
    }
}

```

```

Thread 1 выводит 0
Thread 1 выводит 1
Thread 2 выводит 0
Thread 2 выводит 1
Thread 2 выводит 2
Thread 2 завершился
<...>

```

Рисунок 1.3 – Применение потоков *Thread*

1.5.2 В основе библиотеки *TPL* лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов *.NET* задача представлена специальным классом - классом *Task*, который находится в пространстве имен *System.Threading.Tasks*. Данный класс описывает отдельную задачу, которая запускается асинхронно в одном из потоков из пула потоков. Хотя ее также можно запускать синхронно в текущем потоке.

Для определения и запуска задачи можно использовать различные способы. Первый способ создание объекта *Task* и вызов у него метода *Start*.

В качестве параметра объект *Task* принимает делегат, то есть мы можем передать любое действие, которое соответствует данному делегату, например, лямбда-выражение или ссылку на какой-либо метод. Метод *Start()*, собственно, запускает задачу.

Второй способ заключается в использовании статического метода *Task.Factory.StartNew()*. Этот метод также в качестве параметра принимает делегат, который указывает, какое действие будет выполняться. При этом этот метод сразу же запускает задачу. В качестве результата метод возвращает запущенную задачу.

Третий способ определения и запуска задач представляет использование статического метода *Task.Run()*. Он также в качестве параметра может принимать делегат - выполняемое действие, и возвращает объект *Task*.

Применение *Task* при запуске задач показано на рисунке 1.5

```
class Program
{
    static void Main(string[] args)
    {
        Task task = new Task(Display);
        task.Start();
        Console.WriteLine("Завершение метода Main");
        Console.ReadLine();
    }

    static void Display()
    {
        Console.WriteLine("Начало работы метода Display");
        Console.WriteLine("Завершение работы метода Display");
    }
}
```

Завершение метода Main
Начало работы метода Display
Завершение работы метода Display

Рисунок 1.5 – Применение *Task*

1.5.3 Класс *Parallel* также является частью *TPL* и предназначен для упрощения параллельного выполнения кода. Класс *Parallel* имеет ряд методов, которые позволяют распараллелить задачу.

Метод *Parallel.For* позволяет выполнять итерации цикла параллельно. В параметрах принимает начальный индекс элемента в цикле, а второй параметр - конечный индекс, который будет пройден включительно. Третий параметр – делегат, указывает на метод, который будет выполняться один раз за итерацию. Пример использования *Parallel.For* приведён на рисунке 1.6.

```
ParallelLoopResult result =
    Parallel.For(0, 10, i =>
    {
        Console.WriteLine("{0}, task: {1}, thread: {2}", i,
            Task.CurrentId, Thread.CurrentThread.ManagedThreadId);
        Thread.Sleep(10);
    });
Console.WriteLine(result.IsCompleted);
```

```
№ 0, task: 1, thread: 1
№ 1, task: 2, thread: 3
№ 2, task: 3, thread: 4
№ 3, task: 2, thread: 1
№ 4, task: 1, thread: 4
№ 5, task: 2, thread: 3
№ 6, task: 3, thread: 3
№ 7, task: 1, thread: 1
№ 8, task: 2, thread: 4
№ 9, task: 3, thread: 4
True
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.6 – Применение *Parallel.For*

Метод *Parallel.ForEach* осуществляет итерацию по коллекции, реализующей интерфейс *IEnumerable*, подобно циклу *foreach*, только осуществляет параллельное выполнение перебора. В качестве параметров

принимает перебираемую коллекцию и делегат, выполняющийся один раз за итерацию для каждого перебираемого элемента коллекции.

1.5.4 По умолчанию все элементы коллекции в *LINQ* обрабатываются последовательно, но начиная с *.NET 4.0* в пространство имен *System.Linq* был добавлен класс *ParallelEnumerable*, который инкапсулирует функциональность *PLINQ* (*Parallel LINQ*) и позволяет выполнять обращения к коллекции в параллельном режиме.

При обработке коллекции *PLINQ* использует возможности всех процессоров в системе. Источник данных разделяется на сегменты, и каждый сегмент обрабатывается в отдельном потоке. Это позволяет произвести запрос на многоядерных машинах намного быстрее.

Пример программы с применением *PLINQ* показан на рисунке 1.7

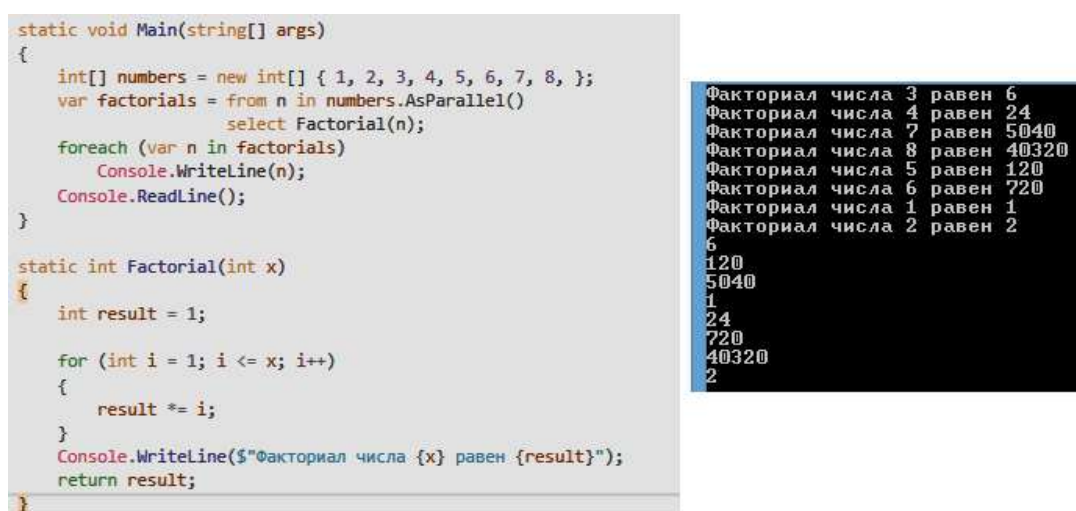


Рисунок 1.7 – Пример применения *PLINQ*

В то же время по умолчанию *PLINQ* выбирает последовательную обработку данных. Переход к параллельной обработке осуществляется в том случае, если это приведет к ускорению работы. Однако, как правило, при параллельных операциях возрастают дополнительные издержки. Поэтому если параллельная обработка потенциально требует больших затрат ресурсов, то *PLINQ* в этом случае может выбрать последовательную обработку, если она не требует больших затрат ресурсов.

Поэтому смысл применения *PLINQ* имеется преимущественно на больших коллекциях или при сложных операциях, где действительно выгода от распараллеливания запросов может перекрыть возникающие при этом издержки.

Также следует учитывать, что при доступе к общему разделяемому состоянию в параллельных операциях будет неявно использоваться синхронизация, чтобы избежать взаимоблокировки доступа к этим общим ресурсам. Затраты на синхронизацию ведут к снижению производительности, поэтому желательно избегать или ограничивать применения в параллельных операциях разделяемых ресурсов.

1.6 Синхронизация потоков в C# .NET

1.6.1 При построении многопоточного приложения необходимо гарантировать, что любая часть разделяемых данных защищена от возможности изменения их значений множеством потоков. Учитывая, что все потоки в *AppDomain* имеют параллельный доступ к разделяемым данным приложения, представьте, что может случиться, если несколько потоков одновременно обратятся к одному и тому же элементу данных. Поскольку планировщик потоков случайным образом будет приостанавливать их работу, что если поток А будет прерван до того, как завершит свою работу? А вот что: поток В после этого прочтет нестабильные данные.

Решение проблемы состоит в том, чтобы синхронизировать потоки и ограничить доступ к разделяемым ресурсам на время их использования каким-нибудь потоком. В языке программирования C# существует несколько способов достижения этой цели.

1.6.2 Одним из способов является использование ключевого слова *lock* [5]. Оператор *lock* определяет блок кода, внутри которого весь код блокируется и становится недоступным для других потоков до завершения работы текущего потока. Для блокировки с ключевым словом *lock* используется некоторый объект-заглушка, созданный пользователем. Когда выполнение доходит до оператора *lock*, объект-заглушка блокируется, и на время его блокировки монопольный доступ к блоку кода имеет только один поток. После окончания работы блока кода, объект-заглушка освобождается и становится доступным для других потоков. Применение *lock* представлено на рисунке 1.8


```

public class MyThread
{
    private object threadLock = new object();

    public void ThreadNumbers()
    {
        // Используем маркер блокировки
        lock (threadLock)
        {
            // Информация о потоке
            Console.WriteLine("{0} поток использует метод ThreadNumbers", Thread.CurrentThread.Name);
            // Выводим числа
            Console.WriteLine("Числа: ");
            for (int i = 0; i < 10; i++)
            {
                Random rand = new Random();
                Thread.Sleep(1000 * rand.Next(5));
                Console.Write(i + ", ");
            }
            Console.WriteLine();
        }
    }
}

```

```

Работает поток: #0 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #1 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #7 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #3 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #2 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #4 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #5 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #9 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #6 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Работает поток: #8 поток использует метод ThreadNumbers
Числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,

```

Рисунок 1.8 – Применение *lock*

1.6.3 Еще один инструмент управления синхронизацией потоков представляет класс *Mutex*, также находящийся в пространстве имен *System.Threading* [5]. Данный класс является классом-оболочкой над соответствующим объектом ОС Windows "мьютекс". Перепишем пример из прошлой темы, используя мьютексы:

Основную работу по синхронизации выполняют методы *WaitOne()* и *ReleaseMutex()*, первый приостанавливает выполнение потока до тех пор, пока не будет получен мьютекс, второй используется потоком освобождает мьютекс после выполнения всех действий, когда данный объект синхронизации больше не нужен. Таким образом, когда выполнение дойдет до вызова *WaitOne()* мьютекса, поток будет ожидать его освобождения. И после его получения продолжит выполнять свою работу. Пример применения мьютекса приведён на рисунке 1.8

```

class Example
{
    private static Mutex mut = new Mutex();
    private const int numIterations = 1;
    private const int numThreads = 3;
    static void Main()
    {
        for (int i = 0; i < numThreads; i++)
        {
            Thread newThread = new Thread(new ThreadStart(ThreadProc));
            newThread.Name = String.Format("Thread{0}", i + 1);
            newThread.Start();
        }
    }

    private static void ThreadProc()
    {
        for (int i = 0; i < numIterations; i++)
        {
            UseResource();
        }
    }

    private static void UseResource()
    {
        Console.WriteLine("{0} is requesting the mutex",
            Thread.CurrentThread.Name);
        mut.WaitOne();
        Console.WriteLine("{0} has entered the protected area",
            Thread.CurrentThread.Name);
        Thread.Sleep(500);
        Console.WriteLine("{0} is leaving the protected area",
            Thread.CurrentThread.Name);
        // Release the Mutex.
        mut.ReleaseMutex();
        Console.WriteLine("{0} has released the mutex",
            Thread.CurrentThread.Name);
    }
}

```

```

Thread1 is requesting the mutex
Thread2 is requesting the mutex
Thread1 has entered the protected area
Thread3 is requesting the mutex
Thread1 is leaving the protected area
Thread1 has released the mutex
Thread3 has entered the protected area
Thread3 is leaving the protected area
Thread3 has released the mutex
Thread2 has entered the protected area
Thread2 is leaving the protected area
Thread2 has released the mutex

```

Рисунок 1.9 – Применение мьютекса

1.6.4 Одним из самых эффективных объектов синхронизации является семафор. Для его создания используется класс *Semaphore* [5]. Его конструктор принимает два параметра: первый указывает, какому числу объектов изначально будет доступен семафор, а второй параметр указывает, какой максимальное число объектов будет использовать данный семафор.

Основной функционал сосредоточен в методе *Read*, который и выполняется в потоке. В начале для ожидания получения семафора используется метод *sem.WaitOne()*. После того, как в семафоре освободится место, данный поток заполняет свободное место и начинает выполнять все дальнейшие действия. После окончания чтения мы высвобождаем семафор с помощью метода *sem.Release()*. После этого в семафоре освобождается одно место, которое заполняет другой поток.

Применение семафора показано на рисунке 1.10

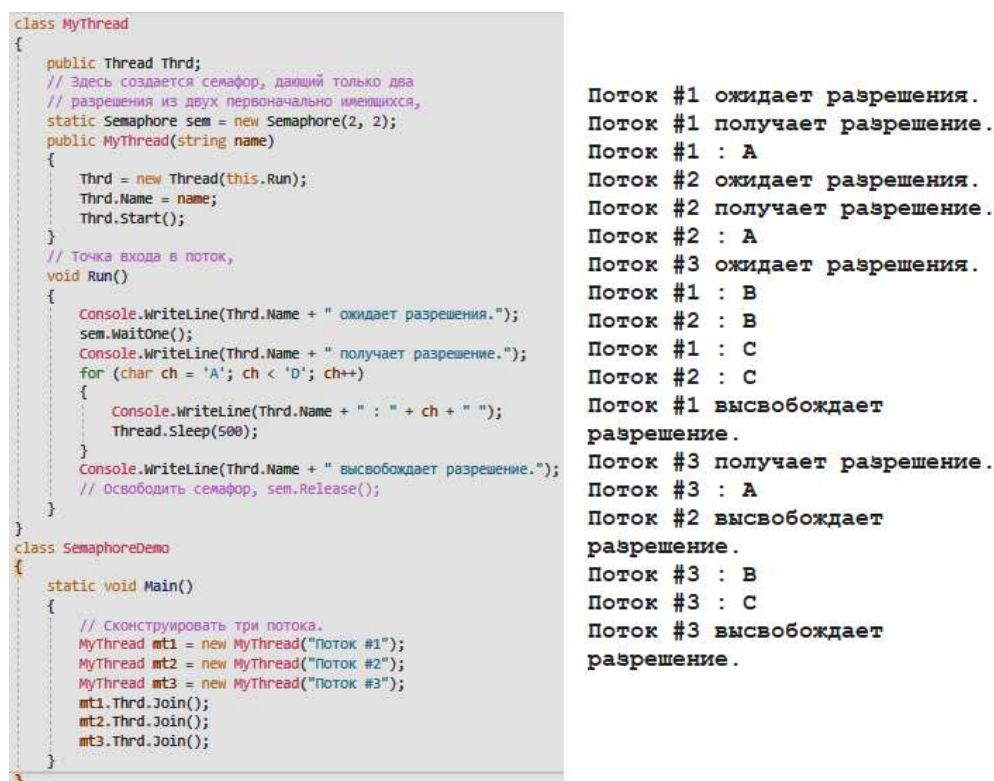


Рисунок 1.10 – Применение семафора

1.7 Взаимодействие с пакетом *MS Excel*

В языке программирования C# для взаимодействия с документами пакета *Microsoft Excel* можно использовать как сторонние *NuGet*-ресурсы, так и библиотеку *Microsoft.Interop.Office.Excel*. На примере *EPPlus* – *dll*-библиотеки с открытым исходным кодом, некоторые исследования показали,

что методы данной *NuGet*-библиотеки работают намного быстрее и проще, чем *API Microsoft.Interop.Office.Excel*, в том числе и из-за отсутствия *COM*-взаимодействия (как для скорости, так и для простоты использования). Она также имеет меньше требований, особенно при развертывании в серверной среде: без установки нежелательной программы Excel. *EPPlus* также может работать с книгами Excel как с потоками. Это может быть полезно при чтении Excel-книг с веб-сервера или когда вы хотите обрабатывать книгу, не имея «физического» файла. Среди его функционала также стоит отметить то, что в случае зашифрованной книги, в конструкторе *EPPlus* реализована вставка пароля, что очень полезно.

Microsoft.Interop.Office.Excel – библиотека которая позволяет представить книгу или лист приложения Excel в качестве объекта и производить над ним необходимые изменения. В данной библиотеке реализована поддержка таких действий, как изменение внешнего вида ячеек, в том числе установка рамок, цвета, настройка ширины и высоты ячеек; использование формул, взаимодействующих с диапазонами ячеек и т.д.

Использование *Microsoft.Interop.Office.Excel* (слева) и библиотеки *EPPlus* (справа) приведено на рисунке 1.11.

```
//Объявляем приложение
Excel.Application ex = new Microsoft.Office.Interop.Excel.Application();
//Отобразить Excel
ex.Visible = true;
//Количество листов в рабочей книге
ex.SheetsInNewWorkbook = 2;
//Добавить рабочую книгу
Excel.Workbook workbook = ex.Workbooks.Add(Type.Missing);
//Отключить отображение окон с сообщениями
ex.DisplayAlerts = false;
//Получаем первый лист документа (счет начинается с 1)
Excel.Worksheet sheet = (Excel.Worksheet)ex.Worksheets.get_Item(1);
//Название листа (вкладки снизу)
sheet.Name = "Отчет за 13.12.2017";
//Пример заполнения ячеек
for (int i = 1; i <= 9; i++)
{
    for (int j = 1; j < 9; j++)
    {
        sheet.Cells[i, j] = String.Format("Boom {0} {1}", i, j);
    }
}
//Захватываем диапазон ячеек
Excel.Range range1 = sheet.get_Range(sheet.Cells[1, 1], sheet.Cells[9, 9]);

using (ExcelPackage excelPackage = new ExcelPackage())
{
    //Set some properties of the Excel document
    excelPackage.Workbook.Properties.Author = "VDMND";
    excelPackage.Workbook.Properties.Title = "Title of Document";
    excelPackage.Workbook.Properties.Subject = "EPPlus demo export data";
    excelPackage.Workbook.Properties.Created = DateTime.Now;

    //Create the Worksheet
    ExcelWorksheet worksheet = excelPackage.Workbook.Worksheets.Add("Sheet 1");

    //Add some text to cell A1
    worksheet.Cells["A1"].Value = "My first EPPlus spreadsheet!";
    //You could also use [line, column] notation:
    worksheet.Cells[1, 2].Value = "This is cell B1!";

    //Save your file
    FileInfo fi = new FileInfo(@"Path\To\Your\File.xlsx");
    excelPackage.SaveAs(fi);
}
```

Рисунок 1.11 – Использование библиотек по работе с Excel

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ

2.1 Постановка задачи, анализ входных и выходных данных

Необходимо провести анализ заданной поверхности в математическом пакете. Анализ включает: построение графика поверхности, вычисления площади поверхности, используя аналитическую формулу

$$S = \iint_D \sqrt{1 + \left(\frac{dz}{dx}\right)^2 + \left(\frac{dz}{dy}\right)^2} dx dy \quad (9)$$

Вид поверхности: $z = Ax + By + C \sin(x) \cos(y)$, где значения коэффициентов $A = -3, B = 1, C = -5$.

Метод численного интегрирования – метод правых прямоугольников, в пределах от $X_n = -40, X_k = 40, Y_n = -80, Y_k = 80$.

Необходимо провести численный анализ заданной поверхности, используя формулы численного интегрирования. Программный код оформить в виде компонента, позволяющий принимать в качестве параметров границы области интегрирования, вычислять площадь заданной поверхности в пределах области интегрирования, готовить данные для построения графика в математическом пакете.

Также необходимо провести анализ скорости обработки заданной поверхности и погрешности численных методов, в зависимости от количества потоков, используемых для расчетов.

2.2 Построение функциональной модели

Для выполнения поставленной задачи, программа должна реализовывать следующий функционал:

- расчет площади заданной поверхности заданным численным методом (методом правых прямоугольников);
- построение графика поверхности;
- анализ влияния распараллеливания вычисления площади поверхности заданным численным методом.

Структурная схема программы представлена на рисунке 2.1.

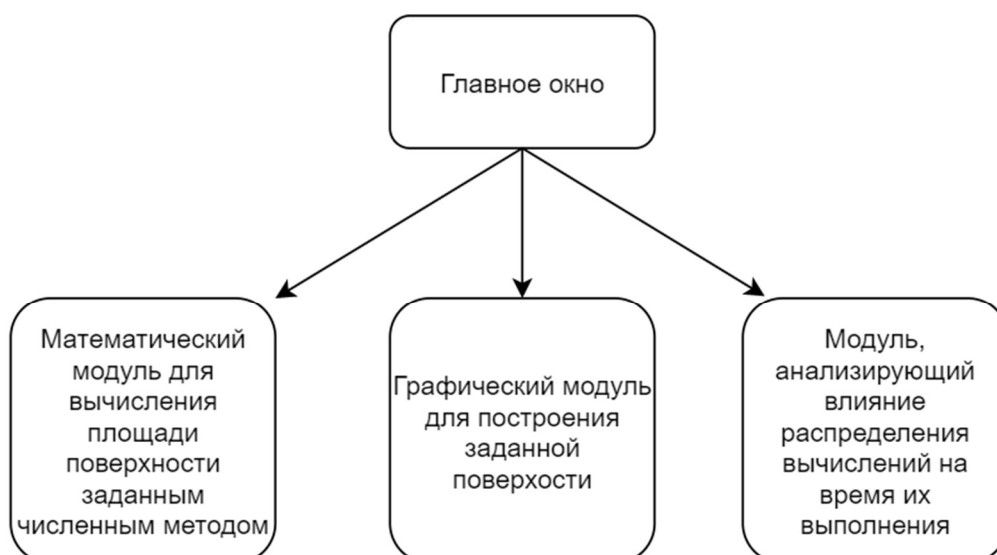


Рисунок 2.1 – Структурная схема программы

Приложение представляет собой комплекс модулей-классов для расчета площади, отображения поверхности и анализа влияния распараллеливания подсчета площади. При этом под анализом будет пониматься наглядное сравнение погрешности вычисления и затраченного времени на выполнение задачи параллельным и линейным методами.

Исполнительные файлы, входящие в состав указанных выше модулей, располагаются в одной директории. Библиотеки языка программирования Python должны быть установлены заблаговременно в стандартную директорию на системном диске.

Реализацию программного кода можно разделить на несколько логических частей:

- разработка графического интерфейса;
- разработка кода главного окна программы;
- разработка кода математического модуля, осуществляющего линейное и распределённое интегрирование;
- разработка кода модуля построения поверхности;
- разработка кода взаимодействия с *MS Excel*.

2.3 Применение метода численного интегрирования функции одной переменной при интегрировании функции двух переменных

Для вышеупомянутых формул (8, 9), а также заданного вида поверхности имеют место следующие формулы:

$$\frac{dz}{dx} = A + C \cos(y) \cos(x) \quad (10)$$

$$\frac{dz}{dy} = B - C \sin(y) \sin(x) \quad (11)$$

$$\begin{aligned} S &= \iint_D \sqrt{1 + \left(\frac{dz}{dx}\right)^2 + \left(\frac{dz}{dy}\right)^2} dx dy = \\ &= \int_{X_H}^{X_K} \left(\int_{Y_H}^{Y_K} \sqrt{1 + (A + C \cos(y) \cos(x))^2 + (B - C \sin(y) \sin(x))^2} dy \right) dx = \\ &= \int_{X_H}^{X_K} \left(h y \sum_{j=1}^N \sqrt{1 + (A + C \cos(y_j) \cos(x))^2 + (B - C \sin(y_j) \sin(x))^2} \right) dx = \\ &= h x \sum_{i=1}^N \left(h y \sum_{j=1}^N \sqrt{1 + (A + C \cos(y_j) \cos(x_i))^2 + (B - C \sin(y_j) \sin(x_i))^2} \right) \\ S &= h x h y \sum_{i=1}^N \sum_{j=1}^N \sqrt{1 + (A + C \cos(y_j) \cos(x_i))^2 + (B - C \sin(y_j) \sin(x_i))^2} \quad (12) \end{aligned}$$

Графическая схема алгоритма интегрирования функции двух переменных указанным методом численного интегрирования, формула (12) для которого представлена выше, приведена в приложении В на рисунке В.1.

2.4 Абсолютная и относительная погрешности

Под погрешностью понимают отклонение измеренного значения некоторой величины от её истинного (действительного) значения.

Абсолютной погрешностью называют разницу между подсчитанным и точным значениями. Если некоторое a – приближённое значение x , то справедлива формула (13):

$$\Delta x = |a - x| \quad (13)$$

Относительная погрешность – отношение абсолютной погрешности к приближённому значению, формула (14) представлена ниже.

$$\delta x = \frac{\Delta x}{a} * 100\% \quad (14)$$

3 РАЗРАБОТКА ПРОГРАММНОГО КОДА

3.1 Реализация графического интерфейса

Графический интерфейс реализован с использованием технологии *Windows Presentation Foundation* платформы *Microsoft .NET Framework 4.7.2* на объектно-ориентированном языке *C#*, руководствуясь [5][6]. Данная технология была выбрана мной, так как она имеет достаточно удобный инструмент для разработки *Windows* приложений.

Окно главной программы, представленное на рисунке 3.1, реализовано окном *WPF* – классом *MainWindow*, являющимся дочерним по отношению к классу *System.Windows.Window*. Описание членов класса *MainWindow* приведено в таблице 3.1.

Таблица 3.1 – Члены класса *MainWindow*

Ключевые слова	Тип	Наименование	Комментарий
1	2	3	4
public	–	MainWindow	Конструктор
private	List <Table-View-Object>	tableViewObjects	Для строк таблицы со статистикой по распределённым вычислениям
private	void	AcceptParameters	Валидация и изменение коэффициентов поверхности и пределов интегрирования
		UpdateInfo	Обновление отображения вида поверхности
		ErrorMsg	Сообщение об ошибке
		PythonBuild_Click	Построение поверхности в Python
		CalculateThreads_Click	Проведение многопоточных вычислений

Продолжение таблицы 3.1

1	2	3	4
private	void	ExportExcel_Click	Экспорт данных в MS Excel с построением графика

Графический интерфейс окна *MainWindow* представлен на рисунке 3.1.

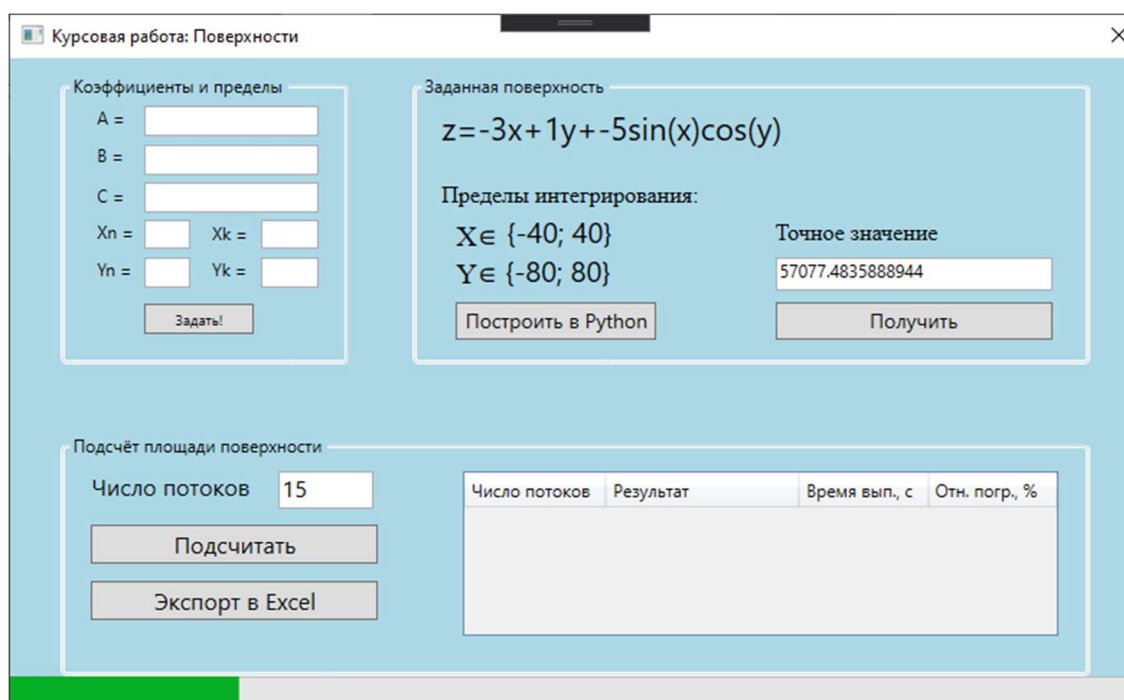


Рисунок 3.1 – Окно *MainWindow*

При запуске процесса подсчёта на определённом числе потоков, программа передаёт управление математическому модулю с выводом сообщения о том, что процедура подсчёта для n потоков запущена, и необходимо подождать, чтобы она завершилась.

Внизу окна виден графический элемент *WPF ProgressBar*, который заполняется ровно в той степени, насколько задача выполнена на данный момент (рисунок 3.2).

По завершению вычислений данные отобразятся в таблице в главном окне.

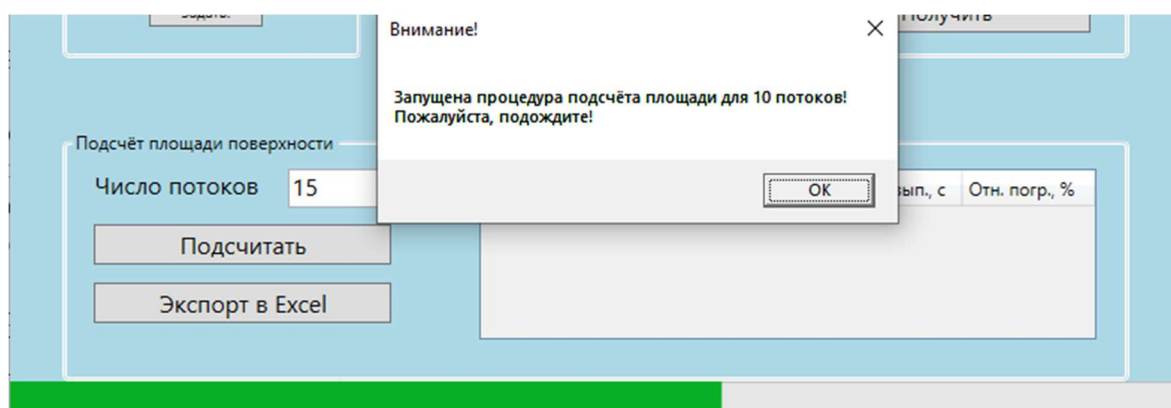


Рисунок 3.2 – Программа в процессе подсчёта площади на 10 потоках

3.2 Реализация вычислительных алгоритмов

Математический модуль программы представлен интерфейсом, который описывает функционал калькулятора площади поверхности и одним классом, реализующим этот интерфейс.

Описание членов интерфейса *IAreaCalculator* приведено в таблице 3.2

Таблица 3.2 – Члены интерфейса *IAreaCalculator*

Тип	Наименование	Комментарий
1	2	3
double	A, B, C, Xn, Xk, Yn, Yk, N	Коэффициенты поверхности
double	CalculateSingleThread	Линейное выполнение подсчёта площади поверхности
double	CalculateMultiThread	Распределённое выполнение подсчёта площади поверхности

Класс *SurfaceAreaCalculator* реализует интерфейс *IAreaCalculator*. Члены данного класса описаны в таблице 3.3

Таблица 3.3 – Члены класса *SurfaceAreaCalculator*

Ключевые слова	Тип	Наименование	Комментарий
1	2	3	4
public	double	A, B, C, Xn, Xk, Yn, Yk, N	Коэффициенты поверхности

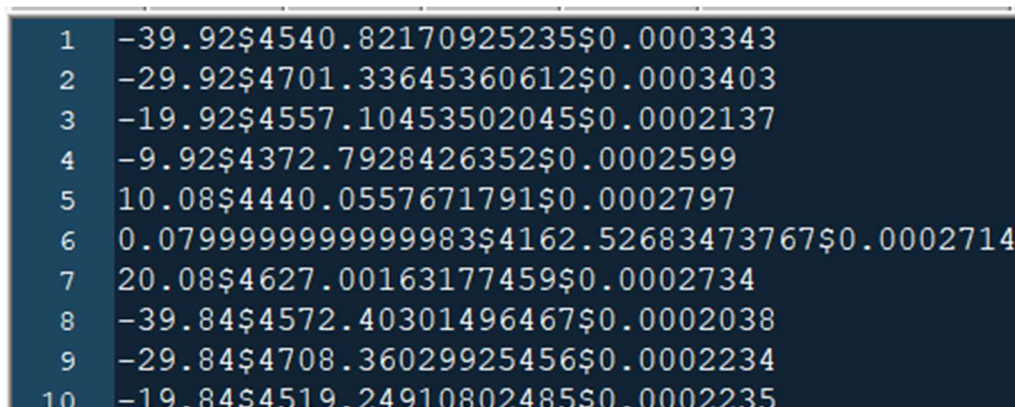
Продолжение таблицы 3.3

1	2	3	4
public	double	CalculateSingleThread	Линейное выполнение подсчёта площади поверхности
	double	CalculateMultiThread	Распределённое выполнение подсчёта площади поверхности
	—	SurfaceAreaCalculator	2 конструктора
	double	hx	Шаг изменения абсцисс
		hy	Шаг изменения ординат
		Xderivativefunction	Формула производной (10)
		Yderivativefunction	Формула производной (11)
		Integrand	Подынтегральная функция (9)

Линейный вариант вычисления площади поверхности, согласно формуле (12), реализован в методе *CalculateSingleThread*.

Распределённый вариант вычисления площади поверхности реализован в методе *CalculateMultiThread*, распределённость достигнута с применением методов класса *Parallel* библиотеки *TPL*, а именно *Parallel.For*, который позволяет выполнять итерации цикла параллельно. Так как в формуле (12) идёт сперва подсчёт суммы по $i = \overline{1, N}$ циклом, *Parallel.For* исполняется на данном цикле. Число потоков задаётся объектом класса *ParallelOptions* в поле *MaxDegreeOfParallelism*, распределение элементов цикла между потоками происходит автоматически. Время, которое потребовалось для вычислений внутреннего цикла в каждом потоке, вычисляется как промежуток времени непосредственно перед и после вычислениями. Далее идёт синхронизация потоков при доступе к файлу для записи результатов – достигается при помощи ключевого слова *lock*, который блокирует доступ к участку кода для других потоков, пока его не выполнит текущий поток. В файл записывается

ровно столько же строк, сколько итераций было пройдено в цикле *Parallel.For*, строка имеет вид « x_i ;площадь\$время_подсчёта» (рисунок 3.3), где x_i – значение абсциссы для правой стороны рассматриваемого прямоугольника, площадь – результат подсчёта внутреннего цикла конкретного потока, время_подсчёта – время, затраченное на вычисление.



```

1 -39.92$4540.82170925235$0.0003343
2 -29.92$4701.33645360612$0.0003403
3 -19.92$4557.10453502045$0.0002137
4 -9.92$4372.7928426352$0.0002599
5 10.08$4440.0557671791$0.0002797
6 0.0799999999999983$4162.52683473767$0.0002714
7 20.08$4627.00163177459$0.0002734
8 -39.84$4572.40301496467$0.0002038
9 -29.84$4708.36029925456$0.0002234
10 -19.84$4519.24910802485$0.0002235

```

Рисунок 3.3 – Содержимое файла вывода с синхронизацией потоков

В ответе получим информацию о значении площади и времени вычисления при использовании определённого числа потоков – сумма всех площадей и сумма всех значений времени в данном файле соответственно.

3.3 Реализация построения поверхности

Для построения поверхности было реализовано приложение на языке программирования *Python* с помощью модуля *pyplot* библиотеки *matplotlib*. С помощью функции *figure* создаётся окно, в нём, используя функцию *add_subplot*, добавляется график. Функциями *set_xlabel*, *set_ylabel*, *set_zlabel* устанавливаются подписи к координатным осям и далее с помощью функции *show* происходит отображение окна с графиком.

Коэффициенты в уравнении вида поверхности, а также пределы интегрирования в данном приложении передаются через параметры запуска, например, запуск приложения “*PythonSurfaceBuilder.exe -A 6*” запустит отображение поверхности с коэффициентом *A* равным 6, незадаанным параметрам будет присвоено значение по умолчанию согласно поставленной задаче.

Данное приложение вызывается из главного приложения при помощи объектов класса *ProcessStartupInfo* и *Process*, далее получает управление, и

отображает поверхность. По завершению работы оно возвращает управление главному приложению.

Результат отображения поверхности приведён на рисунке 3.4.

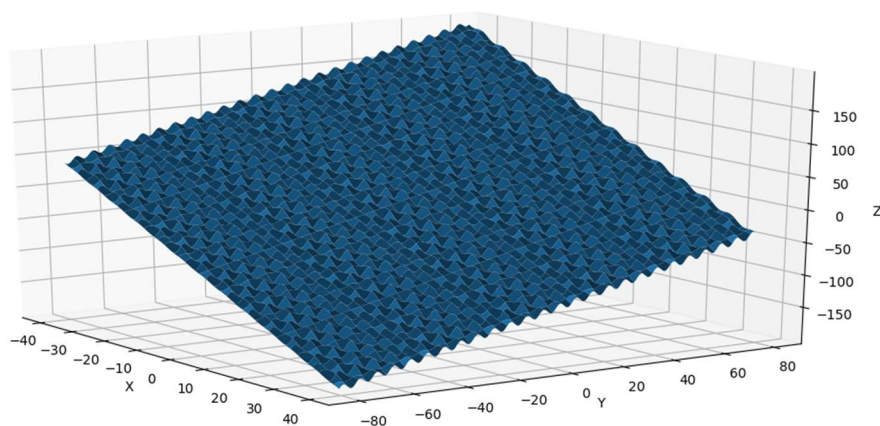


Рисунок 3.4 – Отображение поверхности

3.4 Реализация экспорта статистики

После проведения вычислений и отображения их результатов в таблице в главном окне, как показано на рисунке 3.5, статистику можно экспортировать в *MS Excel*.

Подсчёт площади поверхности

Число потоков

Число потоков	Результат	Время вып., с	Отн. погр., %
1	57074.3721370475	0.3824329	0.00545157437
2	57074.3721370518	0.2774687	0.00545157437
3	57074.3721370517	0.3551754	0.00545157437
4	57074.3721370518	0.3646055	0.00545157437
5	57074.3721370517	0.2228293	0.00545157437

Рисунок 3.5 – Неэкспортированные результаты вычислений

Экспорт данных достигается при помощи библиотеки *Microsoft.Interop.Office.Excel*.

Основной класс библиотеки – *Excel.Application* – создаёт пустой экземпляр приложения. Далее посредством вызовов метода *Add* поля *Workbooks* в него добавляется книга, после чего мы можем обращаться к её листам. В первые два столбца выводится информация о числе потоков (рисунок 3.6) и суммарном времени выполнения вычислений при их использовании.

	А	В
1	Потоки	Время
2	1	0.3824329
3	2	0.2774687
4	3	0.3551754
5	4	0.3646055
6	5	0.2228293
7	6	0.3864714

Рисунок 3.6 – Текстовая информация в *Excel*

Взаимодействие со строками и столбцами возможно как напрямую через коллекцию *Cells* конкретного листа *sheet*, так и выделением группы ячеек/столбцов методом *get_Range()*. Ячейки можно объединять в одну, вызывая метод *Merge()* у выделенной группы, также доступно форматирование конкретного столбца/строки (изменение ширины или высоты и т.д.)

График зависимости времени вычисления от числа потоков представляет собой двумерный график функции одной переменной, значения которой описаны в определённых ячейках. В библиотеке *Microsoft.Interop.Office.Excel* все интерактивные элементы, в том числе графики, рассматриваются как объекты, поэтому на листе создаётся коллекция графиков и в неё добавляется новый *ChartObject* на определённых координатах относительно левого верхнего угла. Для отображения данных на графике необходимо добавить в него коллекцию значений – для этого служат поля *XValues*, *Values* объекта *Series*. Далее задаётся тип графика как элемент перечисления *XlChartType*, содержащего множество видов, и приложение необходимо отобразить, изменив значение его поля *Visible*. Общий вид полученного окна *Excel* представлен на рисунке 3.7.

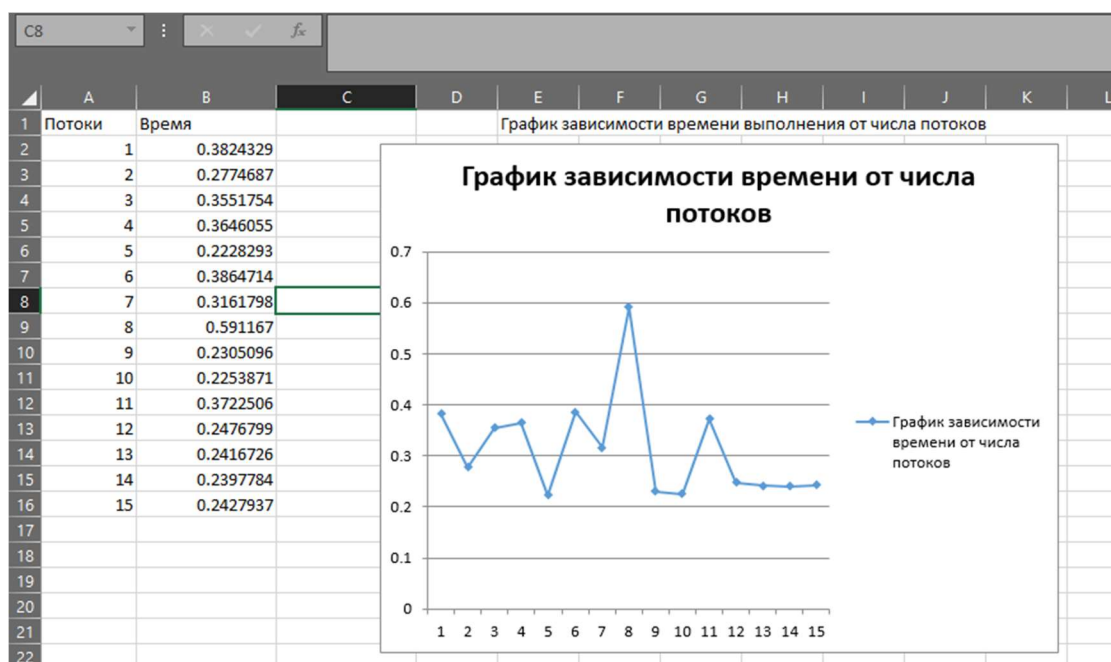


Рисунок 3.7 – Общий вид экспортированной в Excel информации

3.5 Реализация подсчёта погрешности вычислений

Согласно формулам (13, 14) необходимо найти точное значение величины площади заданной поверхности. В данной курсовой работе было реализовано дополнительное приложение *DoubleIntegration*, использующее модуль *integrate* библиотеки *scipy*, для подсчёта значения двойного интеграла по указанному выражению, с последующим выводом значения в файл. Данное значение и было использовано в качестве точного.

Коэффициенты уравнения вида поверхности передаются в данное приложение через параметры командной строки, в случае отсутствия им присваиваются значения по умолчанию согласно поставленной задаче. Для этого используется модуль *ArgumentParser* библиотеки *argparse*

Полученное точное значение выводится в специальную форму в главном окне приложения (рисунок 3.8), и используется далее при подсчёте и выводе относительной погрешности для каждого вычисления. В случае его отсутствия или некорректного отображения погрешность считаться не будет, о чём будет выведено соответствующее сообщение (рисунок 3.9).

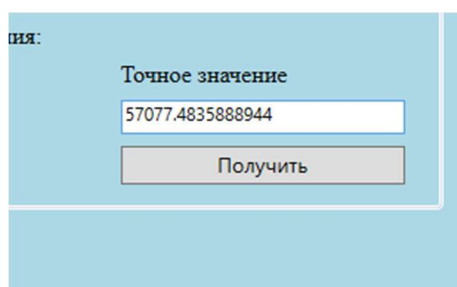


Рисунок 3.6 – Форма, содержащая точное значение

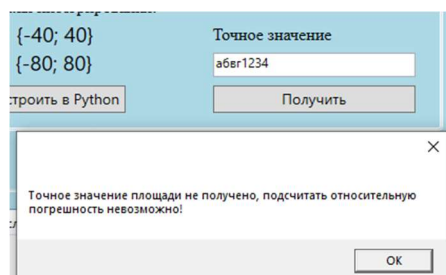


Рисунок 3.7 – Ошибка в точном значении

3.6 Реализация дополнительных структур

Для корректного отображения данных в главном окне был использован элемент *WPF DataGrid* с привязкой к коллекции объектов класса *TableViewObject*. Члены класса представлены в таблице 3.4

Таблица 3.4 – Члены класса *TableViewObject*

Ключевые слова	Тип	Имя	Комментарий
public	int	threadsAmount	Число потоков
	string	answer	Значение площади
		time	Время выполнения вычислений
		err	Абсолютная погрешность

3.7 Описание параметров используемых методов и функций

Параметры используемых методов классов и функций приведены в таблице 3.5 без учёта методов-обработчиков событий (нажатий на кнопки в окнах и т.п.).

Таблица 3.5 – Описание параметров методов и функций

Класс	Функция/ метод	Тип	Имя параметра	Комментарий
Surface Area Calculator	XDerivativefunction, Yderivativefunction, Integrand	double	x, y	значения абсцисс, ординат
Main Window	ErrorMsg	string	template	шаблон сообщения об ошибке , содержащий поля подстановки {0}, {1} и т.п.
			coefficient	название поля ввода коэффициента, содержащее ошибку
	getErr	string	exactValue	точное значение интеграла
			approximate Value	приближённое подсчитанное значение интеграла

4 ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ

4.1 Тестирование *GUI*

В соответствии с [10], *GUI* (англ. *Graphical user interface*, рус. графический интерфейс пользователя) – это то, что видит пользователь; совокупность элементов интерфейса (меню, кнопок, список и т.п.), представленные пользователю на дисплее, исполненные в виде графических изображений. Тестирование *GUI* – процесс тестирования графического пользовательского интерфейса системы из тестируемого приложения. Данный вид тестирования включает в себя проверку окон с элементами управления, такими как кнопки меню, иконка и все виды «баров» – панели инструментов, панели меню, диалоговым окном и т.д. Особенно акцент делается на дизайн структуры, образы, которые они работают должным образом или нет. Кроме того, все кнопки, если являются доступными, должна работать при нажатии; если пользователь изменил размер экрана, ни изображения, ни содержание должно сокращаться, кадрироваться или перекрываться.

Составленный чек-лист тестирования *GUI* приведен в таблице 4.1.

Таблица 4.1 –Чек-лист тестирования GUI

Индекс	Описание	Ожидаемый результат	Соответствие ожидаемому результату
1	2	3	4
1.1	Проверка всех элементов <i>GUI</i> на корректность размера, положения, ширины, длины и «принимаемости» символов и цифр	Корректное отображение вводимых данных в соответствующие поля ввода	Полное
1.2	Правильное отображение сообщений об ошибках	При вводе некорректных данных или отсутствии выбранных в таблице элементов выведется сообщение об ошибке.	Полное

Продолжение таблицы 4.1

1	2	3	4
1.3	Чёткое разграничение функциональных участков окон	Элементы окна не перекрывают друг друга и отображаются полностью по всей своей длине и ширине	Полное
1.4	Шрифт, используемый в приложении, и его отображение, отсутствие орфографических, пунктуационных ошибок	Шрифт читабелен, информация воспринимается корректно, ошибки отсутствуют	Полное
1.5	Адаптация элементов окна под разные размеры экрана	Элементы изменяют свои размеры в соответствии с новым размером окна	Частичное
1.6	Изменение курсора при наведение на различные элементы окна	При наведении курсора на кнопки, они изменяют свой цвет/подсвечиваются	Полное

4.2 Негативное тестирование

Согласно [10], негативное тестирование – вид тестирования, при котором в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам (классический случай – деление на ноль).

При вводе некорректных данных ранее отображалось «не число», в соответствие с чем в программе была введена проверка на наличие нулевых строк во входных данных и корректности выходных данных. Если в результате работы программы производятся попытки сохранить некорректные данные, операция не производится. Под некорректными данными можно понимать текст, который не представляется возможным перевести в нецелое число, начальный предел, превышающий или равный конечному пределу. Пример данного тестирования приведён на рисунке 4.1.

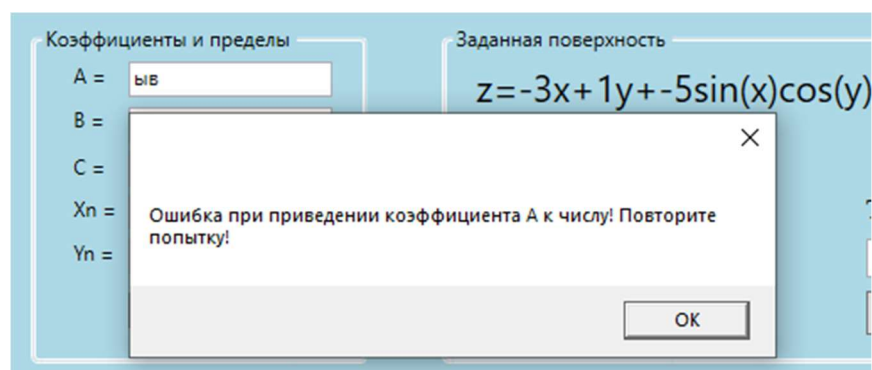


Рисунок 4.1 – Некорректная запись в поле задания коэффициента

От нежелательного ввода и изменения защищены поля «Точное значение» и редактирования строк или столбцов – таблица с данными о подсчёте площади при разном числе потоков.

При отсутствии доступа главного приложения к модулям, а именно модулю отображения поверхности или модулю подсчёта точного значения интеграла, оно выдаст соответствующую ошибку, примеры которой приведены на рисунке 4.2

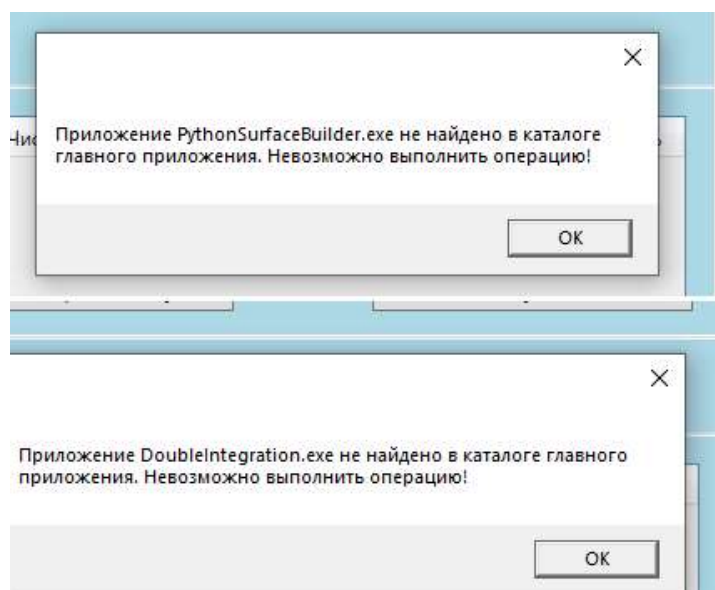


Рисунок 4.2 – Ошибки при отсутствии модулей

При неверно заданном числе потоков, программа выдаст соответствующее сообщение (рисунок 4.3), вычисления не будут начаты и экспорт статистики в *Excel* станет недоступен до момента исправления данной критической ситуации.

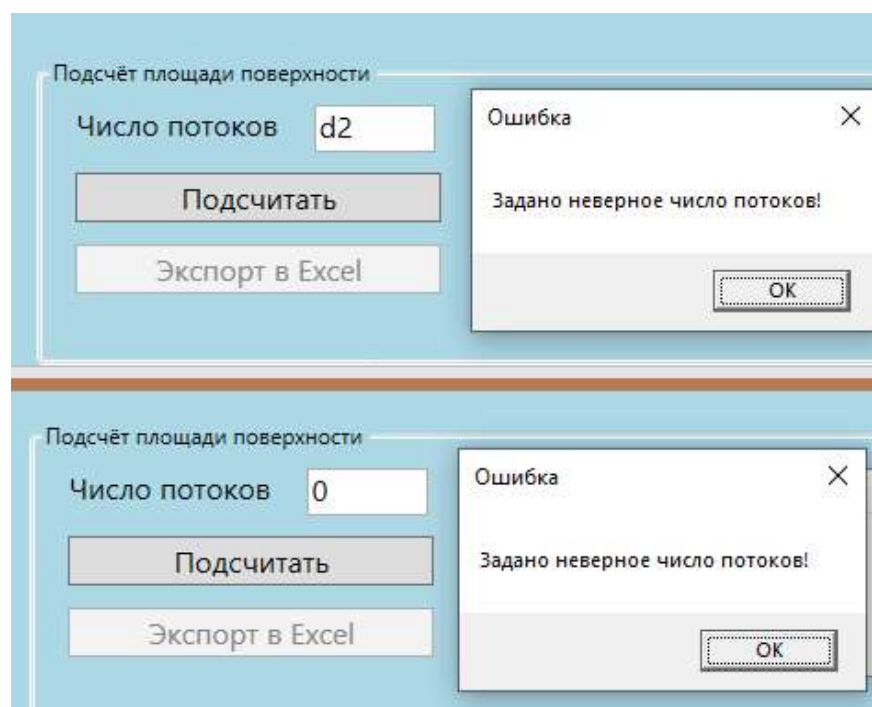


Рисунок 4.3 – Число потоков задано некорректно

4.3 Анализ факторов, влияющих на время вычисления

Для проведения анализа влияния распределения вычислений на время выполнения вычислений значения интеграла, был реализован экспорт информации в MS Excel с построением графика зависимости времени выполнения от числа потоков.

Было проведено сравнение значений в случаях с использованием нескольких потоков, а также в обычной линейной реализации. В данной курсовой работе рассматривается от 1 до 15 потоков для вычисления значения интеграла, но пользователь может задать любое число потоков, до которого необходимо получать данные. Результаты (график *Excel*), полученные при сравнении, приведены на рисунке 4.4. На основании данного графика можно сделать вывод, что скорость выполнения распределённых вариантов методов в несколько раз превышает скорость выполнения обычных линейных методов при достаточной большой размерности матрицы входных данных.



Рисунок 4.4 – График зависимости времени от числа потоков

На основании данного графика можно сделать вывод, что скорость выполнения решения системы алгебраических уравнений наименьшая при совпадении числа потоков и числа ядер процессора компьютера, где запущена система. В случае автора курсовой работы, программа была запущена на компьютере с четырёхядерным процессором AMD A12-9720P.

На основании вышеизложенного можно сделать следующие выводы:

- распределённые методы при достаточно большом числе неизвестных работают в несколько раз быстрее, чем их линейные аналоги. При малой размерности входных данных, время выполнения линейных методов начинает в несколько раз превышать время распределённых аналогов в связи с тем, что будут иметь место дополнительные затраты ресурсов на организацию параллельных вычислений помимо малых затрат ресурсов непосредственно на вычислительные операции;

- при разном количестве процессов наименьшее время выполнения наблюдается при числе, приближающемся к числу процессоров вычислительной машины, на которой запущена программа;

- с увеличением размерности матрицы входных данных растёт и время выполнения решения системы линейных алгебраических уравнений. Это связано с тем, что у каждого процесса увеличивается объём производимых вычислений.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта было разработано комплексное приложение, предназначенное для вычисления площади заданной поверхности в заданных пределах, построения графика данной поверхности и анализа влияния распределения вычисления площади на время выполнения и точность результатов. Были проведены такие этапы разработки, как изучение теоретического материала, алгоритмический анализ задачи и составление алгоритма работы приложения.

По мере выполнения данного курсового проекта был изучен теоретический материал о численном методе интегрирования – методе правых прямоугольников для вычисления значения интеграла, на основе которого в дальнейшем были разработаны алгоритмы решения поставленной задачи.

Разработка программы производилась в интегрированной среде разработки *Microsoft Visual Studio Community 2019* с использованием платформы *.NET Framework* версии 4.7.2, интерфейс пользователя был разработан средствами *Windows Presentation Foundation* языка программирования высокого уровня *C#*. Отображение поверхности было реализовано при помощи инструментов языка *Python*.

Приложение позволяет вычислить площадь поверхности, заданной уравнением при постановке задачи в указываемых пределах интегрирования, выполнить построение интерактивного графика этой поверхности и проанализировать влияние распределения вычисления площади поверхности на время выполнения.

Продукт был разработан, отлажен и протестирован. Поставленные в курсовой работе задачи были решены полностью.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ильин В. А., Позняк Э. Г. Аналитическая геометрия. — М.: ФИЗМАТЛИТ, 2002. — 240 с.
2. Письменный, Д. Т. Конспект лекций по высшей математике: полный курс / Д. Т. Письменный. — 4-е изд. — М.: Айрис-пресс, 2006. — 608 с.
3. Основные понятия. Численные методы линейной алгебры и анализа [Электронный ресурс]: Свободная энциклопедия. — Электронные данные. — Режим доступа: https://slemeshevsky.github.io/python-num-pde/term1/build/html/_computing-integrals/double-triple.html — Дата доступа: -03.05.2020.
4. Java — обзор языка: введение, история и преимущества [Электронный ресурс]: PROGLAND.SU. — Режим доступа: <http://proglang.su/java/introduction-to-programming>. — Дата доступа: 01.05.2020.
5. Джосеф Альбахари и Бен Альбахари. C# 5.0 in a Nutshell, Fifth Edition. — «O'Reilly Media, Inc», 2012. — 1062 с.
6. Полное руководство по WPF [Электронный ресурс]: METANIT.COM. — Режим доступа: <https://metanit.com/sharp/wpf/>. — Дата доступа: 07.04.2020.
7. Matplotlib — Python plotting — Matplotlib 3.2.1 documentation [Электронный ресурс]. — Режим доступа: <https://matplotlib.org/>. — Дата доступа: 03.05.2020.
8. Самоучитель Python [Электронный ресурс]. — Режим доступа: <https://pythonworld.ru/samouch-itel-python>. — Дата доступа: 07.04.2020.
9. Параллельное программирование и библиотека TPL [Электронный ресурс]. — Режим доступа <https://metanit.com/sharp/tutorial/12.1.php>
10. Тестирование графического интерфейса (GUI Testing) [Электронный ресурс]: Статья в журнале «Техника. Технологии. Инженерия». — Режим доступа: <https://moluch.ru/th/8/archive/62/2630/>. — Дата доступа: 07.03.2020.
11. Куликов С.С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. - Минск: Четыре четверти, 2017. - 312 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код приложения

MainWindow.xaml.cs

```
using MathModule;
using Microsoft.Office.Interop.Excel;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Windows;
using Excel = Microsoft.Office.Interop.Excel;

namespace mainApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : System.Windows.Window
    {
        const string errorTemplateParse = "Ошибка при приведении коэффициента {0} к числу! Повторите попытку!";
        const string errorTemplateBounds = "Ошибка! Неверно заданы коэффициенты {0}!";

        const string equationTemplate = "z={0}x+{1}y+{2}sin(x)cos(y)";
        const string resultsPathsTemplate = "Stats\\{0}threadsResult.txt";

        private List<TableViewObject> tableViewObjects { get; set; }

        private IAreaCalculator calculator;

        public MainWindow()
        {
            calculator = new SurfaceAreaCalculator();
            tableViewObjects = new List<TableViewObject>();
            InitializeComponent();
            UpdateInfo();
        }

        private void AcceptParametersClick(object sender, RoutedEventArgs e)
        {
            double A, B, C, Xn, Xk, Yn, Yk;
            if (!double.TryParse(TBox_A.Text, out A))
                ErrorMsg(errorTemplateParse, "A");
            else
                calculator.A = A;

            if (!double.TryParse(TBox_B.Text, out B))
                ErrorMsg(errorTemplateParse, "B");
            else
                calculator.B = B;
```



```

if (!double.TryParse(TBox_C.Text, out C))
    ErrorMsg(errorTemplateParse, "C");
else
    calculator.C = C;

if (!double.TryParse(TBox_Xn.Text, out Xn) || !double.TryParse(TBox_Xk.Text, out Xk) )
{
    ErrorMsg(errorTemplateParse, "Xn или Xk");
}
else if(Xk <= Xn)
{
    ErrorMsg(errorTemplateBounds, "Xn и Xk");
}
else
{
    calculator.Xn = Xn;
    calculator.Xk = Xk;
}

if (!double.TryParse(TBox_Yn.Text, out Yn) || !double.TryParse(TBox_Yk.Text, out Yk))
{
    ErrorMsg(errorTemplateParse, "Yn или Yk");
}
else if (Yk <= Yn)
{
    ErrorMsg(errorTemplateBounds, "Yn и Yk");
}
else
{
    calculator.Yn = Yn;
    calculator.Yk = Yk;
}

UpdateInfo();
MessageBox.Show("Успешно приняты!");
}

private void UpdateInfo()
{
    absc.Text = "";
    ordin.Text = "";
    equationText.Text = String.Format(equationTemplate, calculator.A, calculator.B, calcula-
tor.C);
    absc.Text += "{";
    absc.Text += calculator.Xn;
    absc.Text += "; ";
    absc.Text += calculator.Xk;
    absc.Text += "}";
    ordin.Text += "{";
    ordin.Text += calculator.Yn;
    ordin.Text += "; ";
}

```

```

        ordin.Text += calculator.Yk;
        ordin.Text += "}";
    }

    private void ErrorMsg(string template, string coefficient)
    {
        MessageBox.Show(string.Format(template, coefficient));
    }

    private void PythonBuild_Click(object sender, RoutedEventArgs e)
    {
        if (File.Exists("PythonSurfaceBuilder.exe"))
        {
            ProcessStartInfo process = new ProcessStartInfo("PythonSurfaceBuilder", "-A " + calculator.A
                                                                    + "-B " + calculator.B
                                                                    + "-C " + calculator.C
                                                                    + "-Xn " + calculator.Xn
                                                                    + "-Xk " + calculator.Xk
                                                                    + "-Yn " + calculator.Yn
                                                                    + "-Yk " + calculator.Yk);

            process.UseShellExecute = false;
            using (Process one = Process.Start(process))
            {
                one.WaitForExit();
            }
        }
        else
        {
            MessageBox.Show("Приложение PythonSurfaceBuilder.exe не найдено в каталоге главного приложения. Невозможно выполнить операцию!");
        }
    }

    private void CalculateThreads_Click(object sender, RoutedEventArgs e)
    {
        if (exactValue.Text == "" || !double.TryParse(exactValue.Text, out double res))
        {
            MessageBox.Show("Точное значение площади не получено, подсчитать относительную погрешность невозможно!");
        }

        InfoGrid.ItemsSource = null;
        InfoGrid.Items.Clear();
        tableViewObjects.Clear();
        string[] buf;

        ProgressBar.Visibility = Visibility.Visible;
        ProgressBar.Value = 0;
        if (int.TryParse(ThreadsBox.Text, out int threads) && threads > 0)
        {
            calculator.CalculateSingleThread();
            int i = 1;

```

```

        using (StreamReader sr = new StreamReader(string.Format(resultsPathsTemplate, i), System.Text.Encoding.Default))
        {
            buf = sr.ReadLine().Split('$');
            tableViewObjects.Add(new TableViewObject { answer = buf[0], threadsAmount = i, time = buf[1], err = getErr(exactValue.Text, buf[0]) });
        }

        if (threads > 1)
        {
            ProgressBar.Value = 100 / threads;
            for (i = 2; i <= threads; i++)
            {
                MessageBox.Show($"Запущена процедура подсчёта площади для {i} потоков! Пожалуйста, подождите!", "Внимание!");
                calculator.CalculateMultiThread(i);

                using (var sr = new StreamReader(string.Format(resultsPathsTemplate, i), System.Text.Encoding.Default))
                {
                    buf = sr.ReadLine().Split('$');
                    tableViewObjects.Add(new TableViewObject { answer = buf[0], threadsAmount = i, time = buf[1], err = getErr(exactValue.Text, buf[0]) });
                }
                ProgressBar.Value = (i + 0.0) / threads * 100;
            }
        }
        MessageBox.Show($"Завершено!");
        InfoGrid.ItemsSource = tableViewObjects;
        ExcelBtn.IsEnabled = true;
    }
    else
    {
        ExcelBtn.IsEnabled = false;
        MessageBox.Show("Задано неверное число потоков!", "Ошибка");
    }
}

private string getErr(string exactValue, string approximateValue)
{
    if (exactValue == "" || !double.TryParse(exactValue, out double res))
        return "";
    else
        return (Math.Abs(res - double.Parse(approximateValue)) / double.Parse(approximateValue) * 100.0).ToString();
}

private void ExportExcel_Click(object sender, RoutedEventArgs e)
{
    if (int.TryParse(ThreadsBox.Text, out int threads))
    {
        Excel.Application application = new Microsoft.Office.Interop.Excel.Application();
    }
}

```

```

application.Workbooks.Add(Type.Missing);
Worksheet sheet = (Worksheet)application.Sheets[1];

for (int i = 2; i <= threads + 1; i++)
{
    sheet.Cells[i, 1] = i - 1;
    sheet.Cells[i, 2] = tableViewObjects[i-2].time;
}

sheet.get_Range("E1:M1").Merge();
sheet.Cells[1, 5] = "График зависимости времени выполнения от числа потоков";
sheet.Cells[1, 1] = "Потоки";
sheet.Cells[1, 2] = "Время";
sheet.Columns[1].ColumnWidth = 10;
sheet.Columns[2].ColumnWidth = 15;
sheet.Columns[3].ColumnWidth = 15;

ChartObjects xlCharts = (ChartObjects)sheet.ChartObjects(Type.Missing);
ChartObject myChart = (ChartObject)xlCharts.Add(200, 20, 400, 300);
Chart chart = myChart.Chart;
SeriesCollection seriesCollection = (SeriesCollection)chart.SeriesCollection(Type.Miss-
ing);

Series series = seriesCollection.NewSeries();
series.Name = "График зависимости времени от числа потоков";
series.XValues = sheet.get_Range("A2", "A" + (threads + 1));
series.Values = sheet.get_Range("B2", "B" + (threads + 1));
chart.ChartType = XlChartType.xlLineMarkers;
application.Visible = true;
}
else
    MessageBox.Show("Не удалось прочитать число потоков для экспорта данных.
Отмена операции!");
}

private void getExactValue_Click(object sender, RoutedEventArgs e)
{
    const string resultsErrorTemplate = "Приложение было запущено, но файл value.txt с
результатом в каталоге главного приложения {0}. Невозможно выполнить операцию!";
    const string filename = "DoubleIntegration.exe";

    if (File.Exists(filename))
    {
        ProcessStartInfo process = new ProcessStartInfo(filename, "-A " + calculator.A
                                                    + "-B " + calculator.B
                                                    + "-C " + calculator.C
                                                    + "-Xn " + calculator.Xn
                                                    + "-Xk " + calculator.Xk
                                                    + "-Yn " + calculator.Yn
                                                    + "-Yk " + calculator.Yk);

        process.UseShellExecute = false;
        using (Process one = Process.Start(process))
        {

```

```

one.WaitForExit();
if (File.Exists(filename))
{
    using (StreamReader sr = new StreamReader(filename))
    {
        string res = sr.ReadLine();
        if(double.TryParse(res, out double exactErrorValue))
        {
            exactValue.Text = exactErrorValue.ToString();
            MessageBox.Show("Точное значение успешно подсчитано!");
        }
        else
        {
            MessageBox.Show(string.Format(resultsErrorTemplate, "повреждён"));
        }
    }
}
else
    MessageBox.Show(string.Format(resultsErrorTemplate, "не существует"));
}
}
else
    MessageBox.Show("Приложение DoubleIntegration.exe не найдено в каталоге
главного приложения. Невозможно выполнить операцию!");
}
}
}

```

MainWindow.xaml

```

<Window x:Class="mainApp.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:mainApp"
    mc:Ignorable="d"
    Title="Курсовая работа: Поверхности" ResizeMode="NoResize"
    Height="486.5" Width="800">
    <Grid Background="LightBlue" Height="460" VerticalAlignment="Top" Margin="0,0,0,-3">
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition Height="222"/>
            <RowDefinition Height="27"/>
            <RowDefinition Height="249"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="25*"/>
            <ColumnDefinition Width="247*"/>
            <ColumnDefinition Width="497*"/>
            <ColumnDefinition Width="25*"/>
        </Grid.ColumnDefinitions>
    </Grid>

```

```

<GroupBox Grid.Column="1" Header="Коэффициенты и пределы" Margin="10,10,34,10"
Grid.Row="1">
  <Grid Margin="0,0,-2,0">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.3*"/>
      <ColumnDefinition Width="0.5*"/>
      <ColumnDefinition Width="2*"/>
      <ColumnDefinition Width="0.1*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="0.1*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.1*"/>
      <RowDefinition Height="0.4*"/>
      <RowDefinition Height="0.1*"/>
      <RowDefinition Height="0.05*"/>
    </Grid.RowDefinitions>
    <TextBox Name="TBox_A" Grid.Column="2" Height="21" TextWrapping="Wrap" Ver-
ticalAlignment="Top" Grid.Row="1" Margin="0,0,10,0"/>
    <TextBox Name="TBox_B" Grid.Column="2" Height="21" TextWrapping="Wrap" Ver-
ticalAlignment="Top" Grid.Row="2" Margin="0,0,10,0"/>
    <TextBox Name="TBox_C" Grid.Column="2" Height="21" TextWrapping="Wrap" Ver-
ticalAlignment="Top" Grid.Row="3" Margin="0,0,10,0"/>
    <TextBox Name="TBox_Xn" Grid.Column="2" HorizontalAlignment="Left"
Height="20" TextWrapping="Wrap" VerticalAlignment="Top" Width="32" Grid.Row="4"/>
    <TextBox Name="TBox_Yn" Grid.Column="2" HorizontalAlignment="Left"
Height="21" TextWrapping="Wrap" VerticalAlignment="Top" Width="32" Grid.Row="5"/>
    <TextBox Name="TBox_Xk" Grid.Column="2" Height="20" TextWrapping="Wrap"
VerticalAlignment="Top" Grid.Row="4" Margin="81,0,10,0" RenderTrans-
formOrigin="0.533,0.706"/>
    <TextBox Name="TBox_Yk" Grid.Column="2" Height="21" TextWrapping="Wrap"
VerticalAlignment="Top" Grid.Row="5" Margin="81,0,10,0"/>

    <Button Content="Задать!" Click="AcceptParametersClick" Grid.Column="2" Horizon-
talAlignment="Left" Grid.Row="7" VerticalAlignment="Top" Width="76" FontSize="10"
Height="21"/>
    <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="1" TextWrap-
ping="Wrap" Text="A =" VerticalAlignment="Top" Width="24"/>
    <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="2" TextWrap-
ping="Wrap" Text="B =" VerticalAlignment="Top" Width="24"/>
    <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="3" TextWrap-
ping="Wrap" Text="C =" VerticalAlignment="Top" Width="24" Margin="0,1,0,0"/>
    <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="4" TextWrap-
ping="Wrap" Text="Xn =" VerticalAlignment="Top" Width="29" Height="21"
Grid.RowSpan="2"/>

```

```

    <TextBlock Grid.Column="2" HorizontalAlignment="Left" Grid.Row="4" TextWrap-
ping="Wrap" Text="Xk =" VerticalAlignment="Top" Width="29" Height="20" Mar-
gin="47,1,0,0"/>
    <TextBlock Grid.Column="2" HorizontalAlignment="Left" Grid.Row="5" TextWrap-
ping="Wrap" Text="Yk =" VerticalAlignment="Top" Width="29" Height="20" Margin="47,0,0,0"
RenderTransformOrigin="0.172,0.5"/>
    <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="5" TextWrap-
ping="Wrap" Text="Yn =" VerticalAlignment="Top" Width="29" Height="21"/>
    </Grid>
</GroupBox>

<GroupBox Grid.Column="2" Header="Заданная поверхность" Margin="10"
Grid.Row="1">
    <Grid Margin="0,0,-2,0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="15*"/>
            <ColumnDefinition Width="69*"/>
            <ColumnDefinition Width="66*"/>
            <ColumnDefinition Width="68*"/>
            <ColumnDefinition Width="233*"/>
            <ColumnDefinition Width="16*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="0.1*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.1*"/>
            <RowDefinition Height="0.4*"/>
            <RowDefinition Height="0.1*"/>
            <RowDefinition Height="0.05*"/>
        </Grid.RowDefinitions>
        <Button Content="Построить в Python" Click="PythonBuild_Click" Grid.Column="1"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="139" Height="26" Grid.Row="6"
Grid.ColumnSpan="3" FontSize="14" Margin="10,5,0,0" Grid.RowSpan="2"/>
        <Button Content="Получить" Click="getExactValue_Click" Grid.Column="4" Horizontal-
Alignment="Left" VerticalAlignment="Top" Width="192" Height="26" Grid.Row="6" Font-
Size="14" Margin="31,5,0,0" Grid.RowSpan="2"/>
        <TextBlock x:Name="equationText" Grid.Column="1" HorizontalAlignment="Left"
Grid.Row="1" TextWrapping="Wrap" VerticalAlignment="Top" Width="426" Height="33" Font-
Size="22" Grid.ColumnSpan="4" Grid.RowSpan="2"><Run/><LineBreak/><Run/></TextBlock>

        <TextBlock HorizontalAlignment="Left" Grid.Row="3" TextWrapping="Wrap" Verti-
calAlignment="Top" Width="416" Height="26" FontSize="16" Grid.ColumnSpan="4" Grid.Col-
umn="1"><Run FontFamily="Times New Roman" Text="Пределы интегрирования:"/></Text-
Block>

        <TextBlock HorizontalAlignment="Left" Grid.Row="4" TextWrapping="Wrap" Verti-
calAlignment="Top" Width="192" Height="26" FontSize="16" Grid.Column="4" Mar-
gin="31,0,0,0"><Run FontFamily="Times New Roman" Text="Точное значение"/></TextBlock>

```

```

        <TextBlock HorizontalAlignment="Left" Grid.Row="4" TextWrapping="Wrap" VerticalAlignment="Top" Width="35" Height="26" Margin="10,0,0,0" FontSize="20" Grid.Column="1"><Run FontFamily="Times New Roman" Text="X∈"/></TextBlock>
        <TextBlock x:Name="absc" HorizontalAlignment="Left" Grid.Row="3" TextWrapping="Wrap" VerticalAlignment="Top" Width="158" Height="26" Margin="45,21,0,0" FontSize="20" Grid.Column="1" Grid.ColumnSpan="3" Grid.RowSpan="2"/>
        <TextBlock x:Name="ordin" HorizontalAlignment="Left" Grid.Row="4" TextWrapping="Wrap" VerticalAlignment="Top" Width="173" Height="26" Margin="44,22,0,0" FontSize="20" Grid.Column="1" Grid.ColumnSpan="4" Grid.RowSpan="2"/>
        <TextBlock HorizontalAlignment="Left" Grid.Row="5" TextWrapping="Wrap" VerticalAlignment="Top" Width="35" Height="26" Margin="10,0,0,0" FontSize="20" Grid.Column="1"><Run FontFamily="Times New Roman" Text="Y∈"/></TextBlock>
        <TextBox Name="exactValue" Grid.Column="4" HorizontalAlignment="Left" Height="23" Margin="31,0,0,0" Grid.Row="5" TextWrapping="Wrap" VerticalAlignment="Top" Width="192"/>
    </Grid>
</GroupBox>
<GroupBox Grid.Column="1" Header="Подсчёт площади поверхности" Margin="10,10,10,70" Grid.Row="3" Grid.ColumnSpan="2">
    <Grid Margin="0,0,16,12">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="16*"/>
            <ColumnDefinition Width="114*"/>
            <ColumnDefinition Width="17*"/>
            <ColumnDefinition Width="130*"/>
            <ColumnDefinition Width="395*"/>
            <ColumnDefinition Width="24*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="10*"/>
            <RowDefinition Height="37"/>
            <RowDefinition Height="39*"/>
            <RowDefinition Height="48*"/>
        </Grid.RowDefinitions>
        <Button Content="Подсчитать" Click="CalculateThreads_Click" Grid.Column="1" HorizontalAlignment="Left" VerticalAlignment="Top" Width="199" FontSize="16" Height="27" Grid.Row="2" Grid.ColumnSpan="3"/>
        <Button Name="ExcelBtn" IsEnabled="True" Content="Экспорт в Excel" Click="ExportExcel_Click" Grid.Column="1" HorizontalAlignment="Left" VerticalAlignment="Top" Width="199" FontSize="16" Height="27" Grid.Row="3" Grid.ColumnSpan="3"/>
        <TextBox Name="ThreadsBox" Grid.Column="3" Height="26" Grid.Row="1" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="16" Margin="0,0,62,0"/>
        <TextBlock Grid.Column="1" HorizontalAlignment="Left" Grid.Row="1" TextWrapping="Wrap" Text="Число потоков" VerticalAlignment="Top" Height="26" Width="125" FontSize="16" Grid.ColumnSpan="2"/>
        <DataGrid Name="InfoGrid" Grid.Column="4" HorizontalAlignment="Left" Height="114" Grid.Row="1" Grid.RowSpan="3" VerticalAlignment="Top" Width="413" Grid.ColumnSpan="2" AutoGenerateColumns="False" CanUserAddRows="False" CanUserDeleteRows="False" CanUserReorderColumns="False" CanUserResizeRows="False" CanUserSortColumns="False">
            <DataGrid.Columns>

```



```

        <DataGridTextColumn Width="2.2*" Header="Число потоков" Binding="{Binding Path=threadsAmount}"/>
        <DataGridTextColumn Width="3*" Header="Результат" Binding="{Binding Path=answer}"/>
        <DataGridTextColumn Width="2*" Header="Время вып., с" Binding="{Binding Path=time}"/>
        <DataGridTextColumn Width="2*" Header="Отн. погр., %" Binding="{Binding Path=err}"/>
    </DataGrid.Columns>
</DataGrid>
</Grid>
</GroupBox>
<ProgressBar Name="ProgressBar" Visibility="Hidden" HorizontalAlignment="Left"
Height="32" Margin="0,179,0,0" Grid.Row="3" VerticalAlignment="Top" Width="794" Grid.ColumnSpan="4"
    Minimum="0" Maximum="100" Value="0"/>
</Grid>
</Window>

```

TableViewObject

```

namespace mainApp
{
    class TableViewObject
    {
        public int threadsAmount { get; set; }
        public string answer { get; set; }
        public string time { get; set; }
        public string err { get; set; }
    }
}

```

IAreaCalculator

```

namespace MathModule
{
    public interface IAreaCalculator
    {
        double A { get; set; }
        double B { get; set; }
        double C { get; set; }
        double Xn { get; set; }
        double Xk { get; set; }
        double Yn { get; set; }
        double Yk { get; set; }
        double N { get; set; }

        double CalculateSingleThread();
        double CalculateMultiThread(int threadsAmount);
    }
}

```

SurfaceAreaCalculator

```

using System;

```

```

using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathModule
{
    public class SurfaceAreaCalculator : IAreaCalculator
    {
        public double A { get; set; }
        public double B { get; set; }
        public double C { get; set; }
        public double Xn { get; set; }
        public double Xk { get; set; }
        public double Yn { get; set; }
        public double Yk { get; set; }
        public double N { get; set; }

        public SurfaceAreaCalculator()
        {
            A = -3;
            B = 1;
            C = -5;
            Xn = -40;
            Xk = 40;
            Yn = -80;
            Yk = 80;
            N = 1000;
        }

        public SurfaceAreaCalculator(double paramA, double paramB, double paramC, double
paramXn, double paramXk, double paramYn, double paramYk, double paramN)
        {
            A = paramA;
            B = paramB;
            C = paramC;
            Xn = paramXn;
            Xk = paramXk;
            Yn = paramYn;
            Yk = paramYk;
            N = paramN;
        }

        public double hx => (Xk - Xn) / N;
        public double hy => (Yk - Yn) / N;

        public double XDerivativefunction(double x, double y)
            => A + C * Math.Cos(y) * Math.Cos(x);

        public double YDerivativefunction(double x, double y)

```

```

=> B - C * Math.Sin(x) * Math.Sin(y);

public double Integrand(double x, double y)
{
    double dx = XDerivativefunction(x, y), dy = YDerivativefunction(x, y);
    return Math.Sqrt(1 + dx * dx + dy * dy);
}

public double CalculateSingleThread()
{
    double x, y;
    double sum = 0;
    List<double> xlist = new List<double>();
    List<double> values = new List<double>();

    Stopwatch sw = new Stopwatch();
    sw.Start();

    for (x = Xn + hx; x <= Xk; x += hx)
    {
        for (y = Yn + hy; y <= Yk; y += hy)
        {
            sum += Integrand(x, y);
        }
    }
    sw.Stop();

    sum *= hx;
    sum *= hy;

    using (StreamWriter sWriter = new StreamWriter("Stats\\1threadsResult.txt", false))
    {
        sWriter.Flush();
        sWriter.WriteLine(sum + "$" + sw.Elapsed.TotalSeconds);
    }

    return sum;
}

public double CalculateMultiThread(int threadsAmount)
{
    string filePath = "Stats\\" + threadsAmount + "stats.txt";

    object lockObject = new object();
    int j;

    Stopwatch sw = new Stopwatch();

    if (File.Exists(filePath))
    {
        File.WriteAllText(filePath, string.Empty);
    }
}

```

```

Parallel.For(1, (int) N + 1, new ParallelOptions { MaxDegreeOfParallelism = thread-
sAmount }, i =>
{
    double x, y, sum = 0;
    sw.Reset();
    sw.Start();
    sum = 0;
    x = Xn + i * hx;
    for (y = Yn + hy; y <= Yk; y += hy)
    {
        sum += Integrand(x, y);
    }
    sw.Stop();

    lock (lockObject)
    {
        using (StreamWriter sWriter = new StreamWriter(filePath, true))
        {
            sWriter.WriteLine(Xn + i * hx + "$" + sum + "$" + sw.Elapsed.TotalSeconds);
        }
    }
});

string line;
double time = 0;
double answer = 0;

using (StreamReader sr = new StreamReader(filePath))
{
    while ((line = sr.ReadLine()) != null)
    {
        answer += double.Parse((line.Split('$'))[1]);
        time += double.Parse((line.Split('$'))[2]);
    }
}
answer *= hx;
answer *= hy;

using (StreamWriter sWriter = new StreamWriter("Stats\\" + threadsAmount + "threadsRe-
sult.txt"))
{
    sWriter.WriteLine(answer + "$" + time);
}

return (answer);

}
}
}

```

PythonSurfaceBuilder.py

```

import sys
import argparse
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import math

def createParser ():
    parser = argparse.ArgumentParser()
    parser.add_argument ('-A', '--A', default=-3)
    parser.add_argument ('-B', '--B', default=1)
    parser.add_argument ('-C', '--C', default=-5)
    parser.add_argument ('-Xn', '--Xn', default=-40)
    parser.add_argument ('-Xk', '--Xk', default=40)
    parser.add_argument ('-Yn', '--Yn', default=-80)
    parser.add_argument ('-Yk', '--Yk', default=80)
    parser.add_argument ('-step', '--step', default=0.5)
    return parser

if __name__ == '__main__':
    parser = createParser()
    namespace = parser.parse_args(sys.argv[1:])

    x = np.arange(namespace.Xn, namespace.Xk, namespace.step)
    y = np.arange(namespace.Yn, namespace.Yk, namespace.step)
    X, Y = np.meshgrid(x, y)

    Z = namespace.A * X + namespace.B * Y + namespace.C * np.sin(X) * np.cos(Y)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    ax.plot_surface(X, Y, Z)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()

```

DoubleIntegration

```

import sys
import argparse
import math
from scipy import integrate

def createParser ():
    parser = argparse.ArgumentParser()
    parser.add_argument ('-A', '--A', default=-3)
    parser.add_argument ('-B', '--B', default=1)
    parser.add_argument ('-C', '--C', default=-5)
    parser.add_argument ('-Xn', '--Xn', default=-40)

```

```

parser.add_argument('-Xk', '--Xk', default=40)
parser.add_argument('-Yn', '--Yn', default=-80)
parser.add_argument('-Yk', '--Yk', default=80)
return parser

if __name__ == '__main__':
    parser = createParser()
    namespace = parser.parse_args(sys.argv[1:])
    A = namespace.A
    B = namespace.B
    C = namespace.C
    Xn = namespace.Xn
    Xk = namespace.Xk
    Yn = namespace.Yn
    Yk = namespace.Yk

    f = lambda y, x: math.sqrt(1 + (A + C * math.cos(y) * math.cos(x))**2 + (B - C * math.sin(x) *
    math.sin(y))**2)

    value = integrate.dblquad(f, Xn, Xk, lambda x: Yn, lambda x: Yk)

    with open("value.txt", "w") as file:
        file.write(value)

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Блок-схема алгоритма метода правых прямоугольников

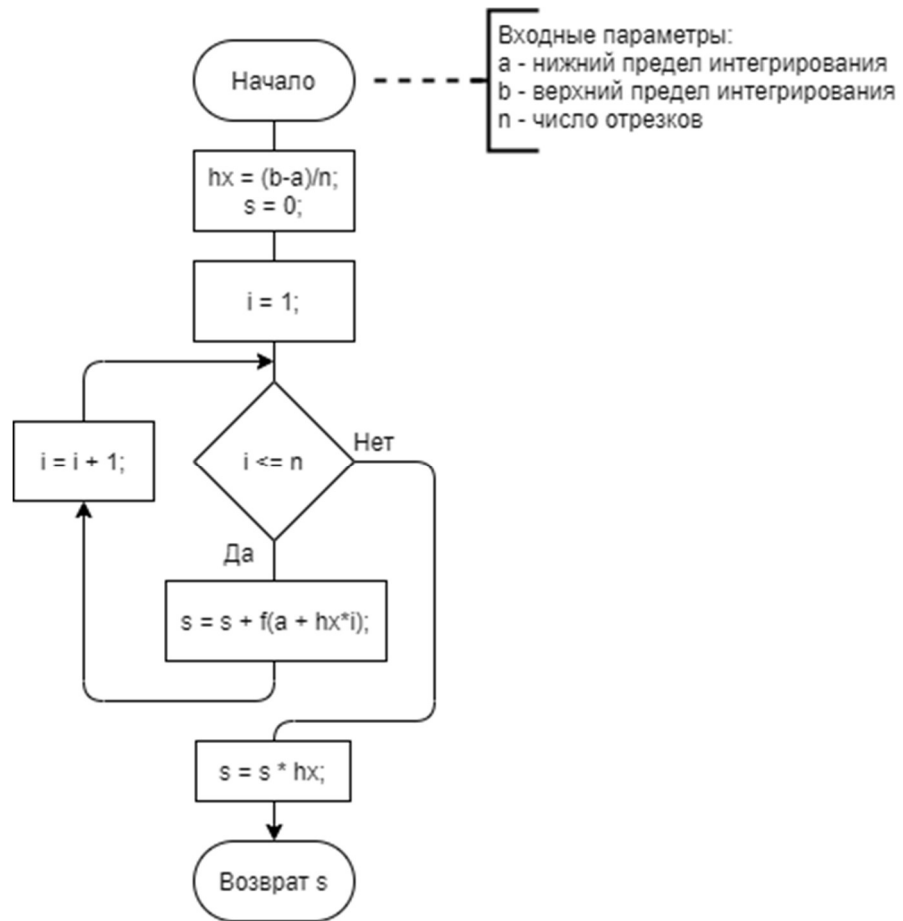


Рисунок Б.1 – Блок-схема алгоритма метода правых прямоугольников

ПРИЛОЖЕНИЕ В (обязательное) **Блок-схема алгоритма интегрирования**

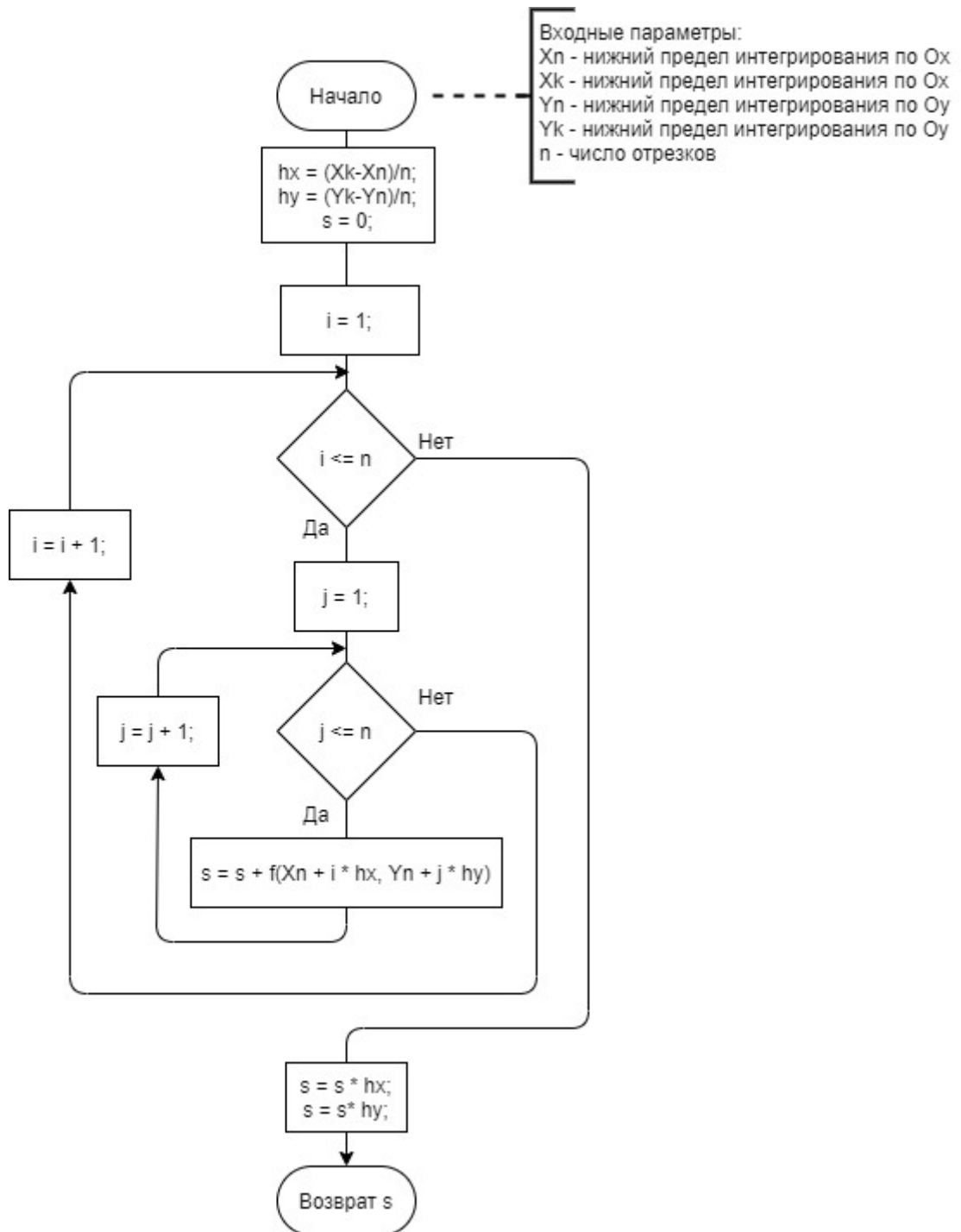


Рисунок В.1 – Блок-схема алгоритма интегрирования