

Содержание

Введение	4
1. Анализ технического задания	6
2. Программная реализация.....	15
3. Тестирование.....	23
Заключение.....	27
Список литературы.....	28
Приложение	29

					МИВУ 01.03.02-13			
Изм.	Лист	№ докум	Подпись	Дата				
Разраб		Савельев М.Ю.			Разработка приложения визуализации параболы, являющейся сглаживающей асимптотой заданного количества частных сумм натурального ряда чисел, с определением точки пересечения оси ординат	Литера	Лист	Листов
Пров		Соколов М.С.					3	37
						МИВЛГУ ПМИ-122		
Н. Контр.								
Утв								

Введение

Задача данной курсового проекта – разработать приложения для визуализации параболы, которая будет является сглаживающей асимптотой известного количества частичных сумм натурального ряда чисел и определение точки пересечения оси ординат. В ходе выполнения работы планируется исследовать и изучить на практике возможные математические методы и алгоритмы в языке программирования C++.

Целью проекта является создание удобного, понятного и оптимизированного приложения, которое поможет визуализировать параболу, заданными частичными суммами натурального ряда чисел и определить точку пересечения параболы с осью ординат.

Для выполнения этой цели необходимо использовать различные математические методы и алгоритмы STL. Необходимо изучить свойства параболы, определить ее уравнение и найти точку пересечения с осью ординат. Для удобной и понятной визуализации графика так же обязательна разработка графического интерфейса, который позволит пользователям вводить количество частных сумм для наглядной визуализации графика сглаживаемой параболы.

Для справки стоит добавить, что натуральные числа – это положительные целые числа, начинающиеся с единицы и продолжающиеся бесконечно. Частичные суммы натурального ряда представляют собой сумму первых n членов ряда, где n – указанное, в нашем случае, количество членов. Главной задачей программы является разработка приложения, которое будет визуализировать параболу для пользователя, являющуюся сглаживающей асимптотой заданного количества частичных сумм натурального ряда чисел. Параллельно этой задачи необходимо определить точку пересечения оси ординат параболы. Точка пересечения оси ординат должна соответствовать условию $(0, y)$.

Для создания программы использован язык программирования C++ с использованием платформенного приложения MFC для разработки приложения визуализации и расчета сглаживающей асимптоты параболы. Важными аспектами ж этой работы будут анализ и обработка данных, визуализация результатов графика и создание понятного и удобного пользовательского интерфейса.

Результаты работы могут быть полезными для изучения визуализированных свойств ряда натуральных чисел, а также для изучения сглаживающих асимптот и их влияния на график функции. Интерфейс приложения предоставляет удобный и интуитивно понятный интерфейс, который никаким образом не мешает исследовать и анализировать параболу с разными количествами частичных сумм натурального ряда чисел.

В итоговом варианте этого проекта ожидается получение практических результатов, которые будут полезными и интересными для любого пользователя. Приложение будет иметь возможность сохранить результаты в виде скриншота, что не даст результату работы исчезнуть бесследно или быть доступным для пользователя в любой момент.

Анализ технического задания

Курсовая проект посвящен разработке приложения для визуализации параболы, которая является сглаживающей асимптотой определенного количества частичных сумм натурального ряда чисел. Важной частью работы над этим курсовым проектом является определение точки пересечения этой параболы с осью ординат. В этом разделе будет проведен подробный анализ технического задания проекта, который представляет собой список указаний, требований и ограничений, которые необходимы для реализации и правильного функционирования приложения.

Цель проекта заключается в разработке приложения, способного визуализировать параболу в соответствии с заданными пользователем параметрами. Построить параболу, которая будет служить сглаживаемой асимптотой заданного количества частичных сумм натурального ряда чисел, и найти точку ее пересечения с осью ординат.

Функциональные требования

Пользователю необходимо иметь возможность ввести достаточное для реализации программы количество частичных сумм натурального ряда чисел. Приложение должно рассчитать и наглядно визуализировать параболу, основываясь на заданном пользователем числе. Определить точку пересечения данной параболы с осью ординат и вывести координаты данной точки. Приложение необходима возможность предоставить возможность сохранения результатов в виде скриншота. По мере необходимости, пользователю может быть предложено ознакомиться с дополнительной информацией, доступной через кнопку "справочник".

Технические требования

Приложение должно быть разработано на языке программирования, с возможностью поддерживать графический интерфейс. Для визуализации графиков могут использоваться особые графические библиотеки. Расчеты должны быть оптимизированы для обеспечения короткого времени ожидания ответа. Приложение должно иметь удобный формат ввода и вывода данных.

Ограничения проекта

Анализ технического задания позволяет оценить основные требования и ограничения этого проекта. Благодаря нему, проект будет некой схемой для дальнейшей разработки приложения и обеспечения понимания того, что необходимо достичь в рамках курсового проекта.

Алгоритм работы с программой

Для реализации поставленной задачи необходимо реализовать алгоритм нахождения коэффициентов бедующей параболы. Для этого реализуем метод Гаусса для решения системы уравнения и нахождения коэффициентов.

Как известно, у параболы есть коэффициент «с» отвечающий за точку пересечения графика в координате 0 по «х» и значением у, которое будет равно коэффициенту «с».

Что бы решить такую задачу будем реализовывать следующий алгоритм для поиска этого числа:

- Для начала нужно найти набор координат. Так как числовой ряд — это выражение вида(1.1):

$$\sum_{n=1}^{\infty} u_n = u_1 + u_2 + \dots + u_n + \dots, \quad (1.1)$$

где члены ряда $u_1, u_2, \dots, u_n, \dots$ — действительные или комплексные числа, u_n общий член ряда. Ряд задан, если известен общий член ряда u_n , выраженный как функция его номера n .

n -я частичная сумма ряда — это сумма первых n членов ряда(1.2):

$$S_n = u_1 + u_2 + \dots + u_n. \quad (1.2)$$

Рассмотрим следующие суммы(1.3):

$$S_1 = u_1, \quad S_2 = u_1 + u_2, \quad S_3 = u_1 + u_2 + u_3, \dots \quad (1.3)$$

Ряд сходится, если существует конечный предел (1.4):

$$S = \lim_{n \rightarrow \infty} S_n \quad (1.4)$$

Ряд расходится, если (1.4) не существует или равен бесконечности[2,5с.]

В математике существует метод суммирования, по которому получается определенный результат даже у расходящихся рядов (2).

$$1 + 2 + 3 + 4 + \dots = -\frac{1}{12} \quad (2)$$

Набор координат можем найти по формуле (3). С помощью этой формулы строится основной график частных сумм натурального ряда чисел. Координаты рассчитанные с помощью этой формулы заносим в вектора `x_coordinates` и `y_coordinates` соответственно.

$$\sum_{k=1}^n k = \frac{n(n+1)}{2} \quad (3)$$

- Далее, переходим к реализации метода Гаусса. Задаем количество точек с помощью `int size = x_coordinates`.

- Создаем матрицы А и В, которые будут хранить коэффициенты системы уравнения. Затем необходимо заполнить эти матрицы. Заполнение проходит по методу наименьших квадратов. Определив частные производные, положив их равными нулю и собрав коэффициенты при неизвестных a_0, a_1, \dots, a_m в соответствующие суммы, получим следующую систему линейных алгебраических уравнений [3,83 с.] (4):

$$\begin{cases} (n+1)a_0 + a_1 \sum_{j=0}^n x_j + a_2 \sum_{j=0}^n x_j^2 + \dots + a_m \sum_{j=0}^n x_j^m = \sum_{j=0}^n y_j; \\ a_0 \sum_{j=0}^n x_j + a_1 \sum_{j=0}^n x_j^2 + a_2 \sum_{j=0}^n x_j^3 \dots + a_m \sum_{j=0}^n x_j^{m+1} = \sum_{j=0}^n y_j x_j; \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ a_0 \sum_{j=0}^n x_j^m + a_1 \sum_{j=0}^n x_j^{m+1} + a_2 \sum_{j=0}^n x_j^{m+2} \dots + a_m \sum_{j=0}^n x_j^{2m} = \sum_{j=0}^n y_j x_j^m \end{cases} \quad (4)$$

В коде это выглядит примерно так:

$A[0][0] += x*x*x*x$ $A[0][1] += x*x*x$ $A[0][2] += x*x$

$A[1][0] += x*x*x$ $A[1][1] += x*x$ $A[1][2] += x$

$A[2][0] += x * x$ $A[2][1] += x$ $A[2][2] += 1;$

$B[0] += x * x * y$ $B[1] += x * y$ $B[2] += y;$

Здесь x и y представляют собой координаты точек.

Таким образом для каждой точки (x,y) происходит вычисления соответствующих элементов матрицы А и В, основываясь на формулах метода наименьших квадратов. Это позволит получить систему уравнения, которую возможно будет решить методом Гаусса.

- Для решения уравнения методом Гаусса, необходимо начиная с первого столбца и первой строки найти максимальный элемент. По нахождению этого элемента нужно поменять местами текущую строку с строкой, содержащий этот элемент. Это происходит при использовании функции $\text{swap}(A[i], A[\text{maxIdx}])$ – где i номер столбца, а maxIdx максимальный элемент.

- Целью этих преобразований является приведение матрицы A к верхнетреугольной матрице (5), т.е. к системе, у которой равны нулю все элементы, расположенные ниже главной диагонали. Это необходимо для упрощения решения системы уравнения.

$$\begin{cases} x_1 + a_{12}^{(1)}x_2 + \dots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ x_2 + \dots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \dots\dots\dots \\ x_n = b_n^{(n)} \end{cases} \quad (5)$$

- После привода матрицы A к верхнетреугольному виду, происходит обратный ход, в котором значения и будущие коэффициенты будут вычисляется в обратном порядке [4,11с.]. Начиная с последней строки матрицы A , вычисляем значения последней неизвестной и так дальше двигаясь вверх.

- Наконец, получаем значения коэффициентов a , b и c , которые представляют собой решение системы уравнений. Значения этих коэффициентов будут использованы для построения сглаживающей асимптоты графика.

Результат работы будет выводиться в форму кнопки в виде двух графиков и в виде десятичного числа в поле ввода.

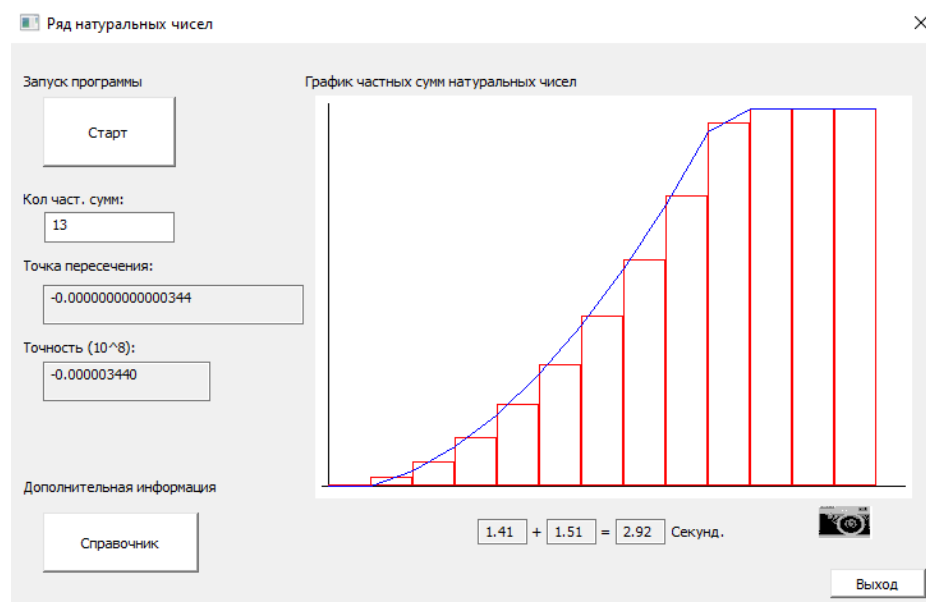


Рисунок 1 – Вид программы

- Пользователь вводит число частичных сумм натурального ряда, которое ему захочется.
- Проверяем корректность записи числа частичных сумм натурального ряда. Если запись этого числа некорректна график частных сумм не может быть построен. В таком случае будет выводиться предупреждения об ошибке.
- Нахождения коэффициентов для реализации параболы по построенному графику частных сумм.
- Вычисление точки пересечения параболой с осью ординат. Эта точка должна соответствовать условию $(0, y)$.
- Визуализация графика построенной параболы вместе с графиком частных чисел натурального ряда. Используемые графические функции и мастер-классов позволяют визуализировать эти графики в форме кнопки.
- Вывод точки пересечения будет проводится в форме поля ввода с отключенной возможности редактирования.

При разработке приложения для визуализации параболы, выбраны следующие инструментальные программные средства:

Язык программирования

C++ является доступным и эффективным языком программирования, который широко используется для разработки приложений с высокой производительностью. Он превосходно удовлетворяет требованиям разработки высокопроизводительного программного обеспечения. Его синтаксис стал стандартом для профессиональных языков программирования, а используемые в нем стратегические принципы проектирования проходят красной нитью через всю вычислительную технику [1,14с.].

MFC (Microsoft Foundation Classes)

MFC является набором классов и функций, предоставленных Microsoft для разработки Windows - приложений на C++. Он представляет удобный и мощный инструментарий для создания графического интерфейса пользователя, обработки событий, работы с окнами и элементами управления. MFC упрощает разработку приложений, особенно если требуется создание традиционного Windows - стиля интерфейса.

Исходя из рассмотрения поставленной задачи, можно выделить следующие факторы, определяющие её сложность:

- Необходимость отрисовки графика частных сумм натурального ряда чисел и сглаживающей асимптоты в виде параболы с помощью обработчика событий в мастер-классы.
- Чем больше количество частичных сумм, тем сложнее будет вычислить коэффициенты параболы и точку пересечения.
- Проверка корректности пользовательского ввода и обработка ошибок может представлять сложность, особенно если необходимо учесть все возможные варианты некорректного ввода.

Таблица 1. Контейнеры STL

Array	Vector	List
<p>Array – стандартный контейнер, который предоставляет собой последовательность фиксированного размера с возможностью случайного доступа к элементам.</p> <p>Плюсы: нет дополнительного расхода памяти для хранения элементов. Минусы: не всегда эффективно для очень больших объемов данных. Если зарезервированного места окажется недостаточно, то данные могут потеряться.</p>	<p>Vector - структура данных, которая является моделью динамического массив. Плюсы: удобная реализация произвольного доступа к элементам. Минусы: вставка и удаление элементов не очень эффективна, также всегда выделяется больше ёмкости, чем нужно.</p>	<p>List – структура данных, которая построена на двусвязных списках. Плюсы: вставка и удаление эффективно работают. Минусы: произвольный доступ плохо реализован.</p>

Так как в программе будет много расчётов и введённые значения данных может быть большим. То для корректной работы программы будем использовать контейнер Vector.

Для рисования графиков будут использоваться следующие графические компоненты:

CDC (Device Context): это объект, представляющий контекст устройства, на котором происходит рисование. CDC содержит методы и свойства для работы с устройством вывода, такими как экран, принтер или изображение.

CWnd (Window): это базовый класс, представляющий окно в MFC приложении.

CPaintDC: это класс-обертка над CDC, который предоставляет функциональность для рисования на окне. Когда окно получает сообщение WM_PAINT, можно создать экземпляр CPaintDC для выполнения рисования на окне.

GDI (Graphical Device Interface): это набор функций и структур, предоставляемых операционной системой Windows для рисования графических объектов. GDI позволяет рисовать линии, прямоугольники, текст, изображения и многое другое.

Обработка сообщения WM_PAINT: WM_PAINT - это сообщение, которое отправляется окну при необходимости его перерисовки. Для реализации рисования на форме в MFC приложении нужно обработать это сообщение и выполнить необходимые действия по отрисовке графических элементов. Рисование графика будет происходить на одном из элементов управления. Некоторые из них:

CButton: кнопка, на которой можно отобразить изображение, символы или просто изменить фон.

CEdit: поле ввода текста, которое также поддерживает рисование на своей поверхности

Программная реализация

Код для построения графика:

- После записи пользователем количества частных сумм натурального ряда чисел, в коде графика рассчитываются его координаты по которым он и строится.

```
// Рисуем ось y
dc1.MoveTo(x_start, y_start);
dc1.LineTo(x_start, y_start - h + 15);

// Задаем цвет пера - красный
CPen pnPenRed(PS_SOLID, 1, RGB(255, 0, 0));
dc1.SelectObject(&pnPenRed);

// Вычисляем ширину столбца
int column_width = (w - x_start - 20) / max_x;

// Цикл для рисования столбцов
for (int i = 1; i <= max_x; i++)
{
    // Вычисляем частичную сумму с использованием формулы
    int sum = (((i + 1) * i) / 2);
    // Вычисляем x-координату
    int x = x_start + column_width * (i);

    // Проверяем, чтобы x-координата находилась в границах
    if (x > x_start + w - 10)
        x = x_start + w - 10;
    if (x < x_start)
        x = x_start;

    // Вычисляем y-координату на основе частичной суммы
    int y = y_start - int((h * sum / (50)));

    // Проверяем, чтобы y-координата находилась в границах
    if (y < 10)
        y = 10;
    if (y > y_start)
        y = y_start;

    // Рисуем столбец
    int column_height = y_start - y;
    dc1.Rectangle(x, y, x + column_width, y_start);
}
```

Рисунок 2 – Код для построения графика частных сумм натурального ряда

Эти координаты заносятся в динамические массивы соответствующие своим осям. В дальнейшем эти данные понадобятся для построения сглаживающей асимптоты и нахождения точки пересечения оси ординат.

Для визуализации уникального графика, добавляем в основное диалоговое окно поле ввода количества частных сумм натурального ряда.

```

// Получаем количество частных сумм с клавиатуры
CString strNumSum;
GetDlgItemText(IDC_EDIT2, strNumSum);
int max_x = _ttoi(strNumSum); // Преобразуем строку в число

// Получаем количество частных сумм с клавиатуры
GetDlgItemText(IDC_EDIT2, strNumSum);

```

Рисунок 3 – Код для принятия количества частных сумм в поле ввода

Рисунок 4 – Внешний вид и ввод в поле ввода

Принятые меры для надежности:

- Для надежности кода в месте ввода частных сумм натурального ряда чисел, установлен объект `MessageBox` для предупреждения пользователя о правильности ввода символов.

Так при записи в поле ввода: букв, некорректных символов или отсутствие символов вообще. Будет выводиться сообщение об ошибке и будет предупреждать о некорректной записи данных.

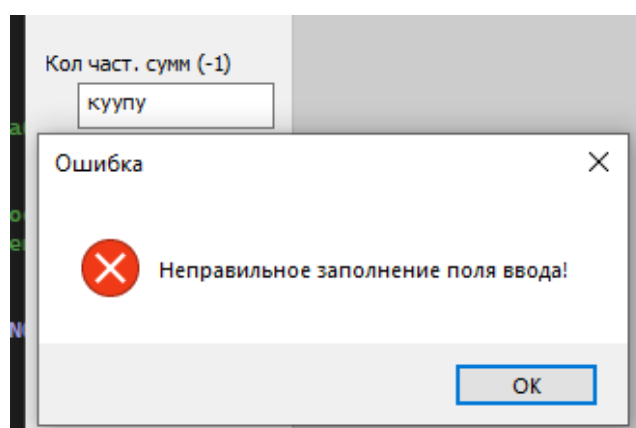


Рисунок 5 – Вывод ошибки при некорректном вводе

- В проекте существует возможность сохранения результатов работы в виде скриншота. В случае ввода конкретных данных в поле ввода будут подсчитаны и построены уникальные для этого значения результаты приложения.

По этому для надежности и доступности результатов в диалоговом окне существует кнопка для создания скриншота. При нажатие на кнопку создается скриншот, который можно сохранить в выбранный пользователям каталог. При этом вызывается второй MessageBox, подтверждающий о создание картинки и показывающий путь к скриншоту.

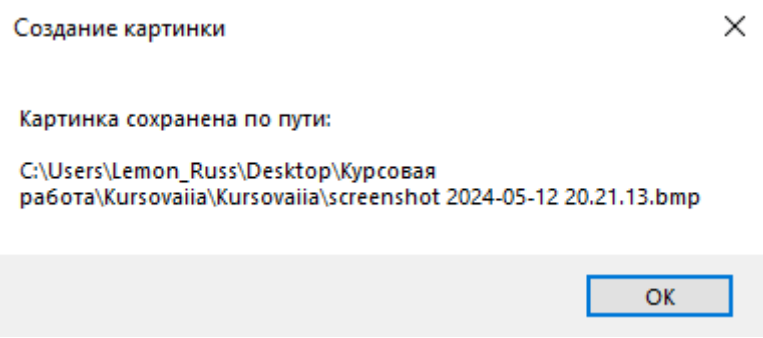


Рисунок 6 – Сообщения о скриншоте

- Чтобы улучшить надежность программы для каждого элемента приложения созданны переменные и классы. С помощью классов возможна построение график в диалоговм окне, а переменные позволяют выводить ответ посчитанный в алгоритмах кода.

Понятность кода

- Для понятности кода в проекте подписаны комментарии почти для каждого алгоритма и математического метода.

Эргономичности интерфейса

- Для удобного и понятного интерфейса в программе все кнопки и поля ввода установлены на равном расстоянии друг от друга, чтобы пользователь не запутался и четко видел, что делает каждый элемент программы.

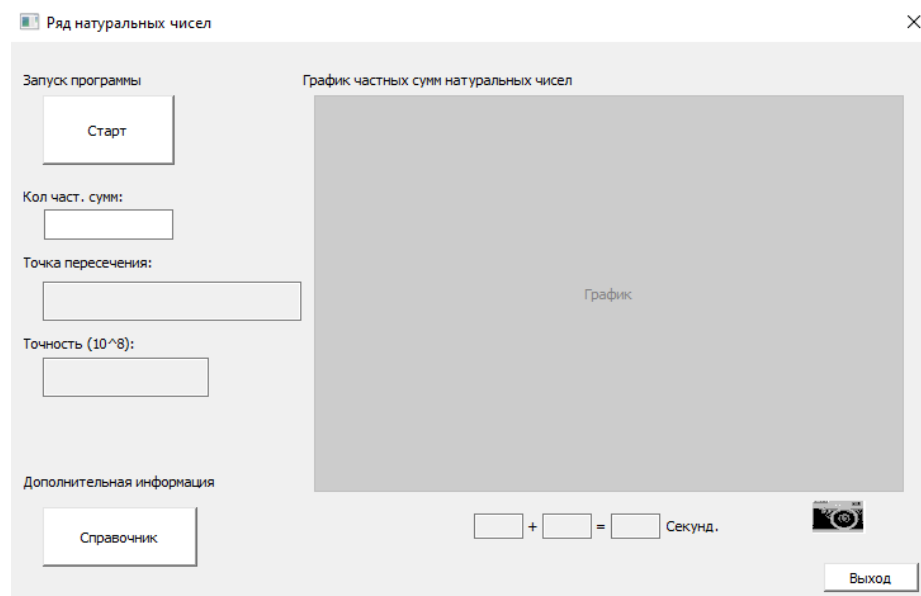


Рисунок 7 – Внешний вид интерфейса

- Для кнопки создания скриншота и сохранения его в корневой каталог проекта, создана картинка для такого чтобы пользователь знал как можно сохранить результаты работы. Чтобы это сделать необходимо создать Bitmap с конкретным размером картинки и добавления кода в функцию OnPaint с указанием индентификатора Bitmap.

```

HBITMAP hIcon = (HBITMAP)LoadImage(AfxGetApp()->m_hInstance,
    MAKEINTRESOURCE(IDB_BITMAP4), IMAGE_BITMAP, 0, 0, LR_DEFAULTCOLOR);
// Загружаем изображение из ресурсов приложения
// AfxGetApp()->m_hInstance получает инстанс приложения
// IDB_BITMAP4 - идентификатор ресурса изображения
// IMAGE_BITMAP указывает, что мы загружаем битовое изображение
// LR_DEFAULTCOLOR - флаг, указывающий на использование цветов по умолчанию
// Результат загрузки сохраняем в переменную hIcon

Button4.SetBitmap(hIcon);
// Устанавливаем загруженное изображение hIcon в качестве битмапа для кнопки Button4

```

Рисунок 8 – Код для внешнего вида кнорки

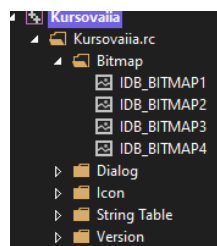


Рисунок 9 – Битовые изображения в ресурсах



Рисунок 10 – Внешний вид кнопки

- Для уникальности программы в инерфейсе при нажатие на кнопку старта, она будет менять свой цвет. Это также работает с другими кнопками в программе и их цвет зависит от тех действий, что они делают.

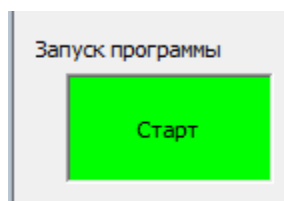


Рисунок 11 – Вид кнопки «Старт» при нажатие



Рисунок 12 – Вид кнопки «Выход» при нажатие

```
void CKursovaiaDlg::Grafika(NMHDR* pNMHDR, LRESULT* pResult)
{
    LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
    // TODO: добавьте свой код обработчика уведомлений
    CDC dc;
    dc.Attach(pNMCD->hdc);
    // Стиль
    UINT state = pNMCD->uItemState;
    bool isPressed = (state & ODS_SELECTED);
    bool isDisabled = (state & ODS_DISABLED);

    // Цвет
    COLORREF bgColor = isPressed ? RGB(0, 255, 0) : RGB(255, 255, 255);
    COLORREF textColor = isDisabled ? RGB(255, 0, 0) : RGB(0, 255, 0);

    // Заливка фона
    dc.FillSolidRect(&pNMCD->rc, bgColor);

    // Рисование текста
    CString buttonText;
    dc.SetTextColor(textColor);
    dc.DrawText(buttonText, &pNMCD->rc, DT_CENTER | DT_VCENTER | DT_SINGLELINE);

    // Отрисовка границ
    if (isPressed)
    {
        dc.DrawEdge(&pNMCD->rc, EDGE_SUNKEN, BF_RECT);
    }
    else
    {
        dc.DrawEdge(&pNMCD->rc, EDGE_RAISED, BF_RECT);
    }
    dc.Detach();
    *pResult = 0;
}
```

Рисунок 13– Функции раскрашивания кнопок «Старт» и «Справочник»
при нажатие

```

void CKursovaiaDlg::Exit1(NMHDR* pNMHDR, LRESULT* pResult)
{
    LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
    // TODO: добавьте свой код обработчика уведомлений
    CDC dc;
    dc.Attach(pNMCD->hdc);
    // Стил
    UINT state = pNMCD->uItemState;
    bool isPressed = (state & ODS_SELECTED);
    bool isDisabled = (state & ODS_DISABLED);

    // Цвет
    COLORREF bgColor = isPressed ? RGB(255, 0, 0) : RGB(255, 255, 255);
    COLORREF textColor = isDisabled ? RGB(255, 0, 0) : RGB(0, 255, 0);

    // Заливка фона
    dc.FillSolidRect(&pNMCD->rc, bgColor);

    // Рисование текста
    CString buttonText;
    //buttonText = "Текст кнопки"; // Закомментируйте эту строку, чтобы не отобра
    dc.SetTextColor(textColor);
    dc.DrawText(buttonText, &pNMCD->rc, DT_CENTER | DT_VCENTER | DT_SINGLELINE);

    // Отрисовка границ
    if (isPressed)
    {
        dc.DrawEdge(&pNMCD->rc, EDGE_SUNKEN, BF_RECT);
    }
    else
    {
        dc.DrawEdge(&pNMCD->rc, EDGE_RAISED, BF_RECT);
    }
    dc.Detach();
    *pResult = 0;
}

```

Рисунок 14– Функции раскрашивания кнопки «Выход» при нажатие

- При необходимости, пользователь может узнать дополнительную информацию если нажмет кнопку «Справочник». Откроется диалоговое окно где будет приведена информация, которая может быть интересна пользователю:

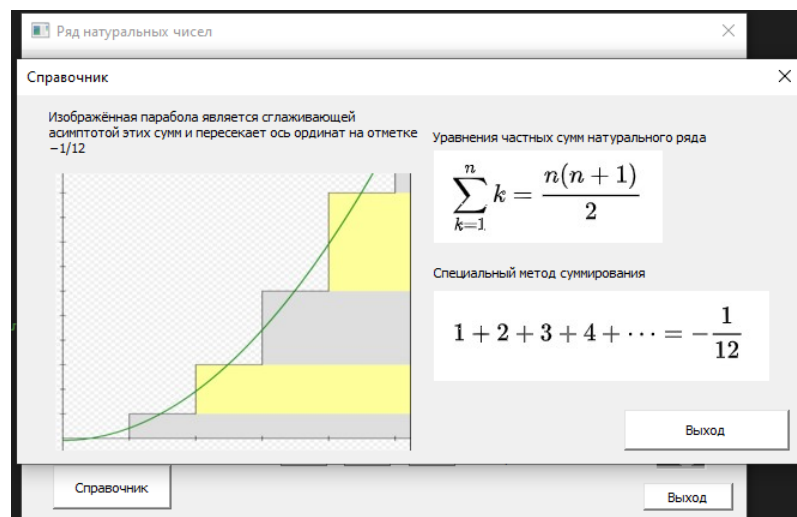


Рисунок 15 – Внешний вид нового диалогового окна «Справочник

```
void CKursovaiaDlg::OnBnClickedButton3()
{
    // TODO: добавьте свой код обработки
    NewSpravoch dlg1;
    dlg1.DoModal();
}
```

Рисунок 16 – Код вывода диалогового окна «Справочник»

- График этой программы строится в форме кнопки, чьи размеры рассчитываются внутри кода кнопки «Старт». И поэтому чтобы кнопка не была действительна и не мешала работе пользователю, форма после запуска программы становится сразу неактивной.

```
// добавление пункта в программу... в системное меню.
// Получаем указатель на кнопку
CButton* pButton = (CButton*)GetDlgItem(IDC_BUTTON2);

// Делаем кнопку неактивной
pButton->EnableWindow(FALSE);
```

Рисунок 17 – Код делающий форму неактивной

- Для нахождения точки пересечения оси ординат с помощью решения системы уравнения методом Гаусса используем такой код:

```
// Создание матрицы A и вектора b
std::vector<std::vector<double>> A(3, std::vector<double>(3));
std::vector<double> b1(3);
int size = x_coordinates.size();

// Заполнение матрицы A и вектора b
for (int i = 0; i < size; i++) {
    double x = x_coordinates[i];
    double y = y_coordinates[i];

    A[0][0] += x * x * x * x;
    A[0][1] += x * x * x;
    A[0][2] += x * x;

    A[1][0] += x * x * x;
    A[1][1] += x * x;
    A[1][2] += x;

    A[2][0] += x * x;
    A[2][1] += x;
    A[2][2] += 1;

    b1[0] += x * x * y;
    b1[1] += x * y;
    b1[2] += y;
}

// Решение системы уравнений методом Гаусса
int n = A.size();
std::vector<double> coefficients(n);

for (int i = 0; i < n; i++) {
    // Поиск максимального элемента в столбце i
    int maxIdx = i;
    for (int j = i + 1; j < n; j++) {
        if (abs(A[j][i]) > abs(A[maxIdx][i])) {
            maxIdx = j;
        }
    }
}
```

Рисунок 18 – Код для решения системы уравнения

```

// Обратный ход
for (int i = n - 1; i >= 0; i--) {
    double sum = 0;
    for (int j = i + 1; j < n; j++) {
        sum += A[i][j] * coefficients[j];
    }
    coefficients[i] = (b1[i] - sum) / A[i][i];
}

// Получение коэффициентов a, b и c
double c = coefficients[2];

```

Рисунок 19 – Получение коэффициентов

Заполняем матрицы методом наименьших квадратов и находим главный элемент перебирая строки и столбцы начиная с первых. Приводим матрицу к верхтреугольному виду и используем обратныйход. Обратный ход проходит по матрице в которой все известные элементы расположены на диагонали, другие элементы будут нулями.

Тестирование

Необходимо обработать возможные ошибки, которые могут произойти при работе программы. Обработаем возможные ошибки при некорректном вводе входных данных:

```
// Получаем количество частных сумм с клавиатуры
CString strNumSum;
GetDlgItemText(IDC_EDIT2, strNumSum);
int max_x = _ttoi(strNumSum); // Преобразуем строку в число

// Получаем количество частных сумм с клавиатуры
GetDlgItemText(IDC_EDIT2, strNumSum);

if (strNumSum.IsEmpty()) {
    MessageBox(L"Поле ввода пустое!", L"Ошибка", MB_OK | MB_ICONERROR);
    return; // Останавливаем выполнение кода здесь
}

// Проверяем, правильно ли заполнено поле ввода
if (max_x <= 2) {
    MessageBox(L"Неправильное заполнение поля ввода!", L"Ошибка", MB_OK | MB_ICONERROR);
    return; // Останавливаем выполнение кода здесь
}

// Проверяем, правильно ли заполнено поле ввода
if (max_x >= 438) {
    MessageBox(L"Слишком большое число для графиков!", L"Ошибка", MB_OK | MB_ICONERROR);
    return; // Останавливаем выполнение кода здесь
}
```

Рисунок 20 – Код для проверки поля ввода

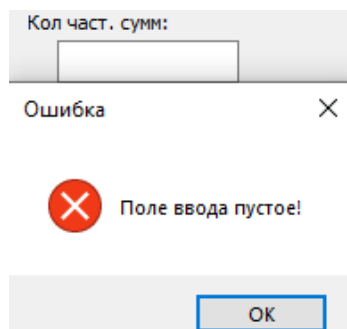


Рисунок 21 – Ошибка 1

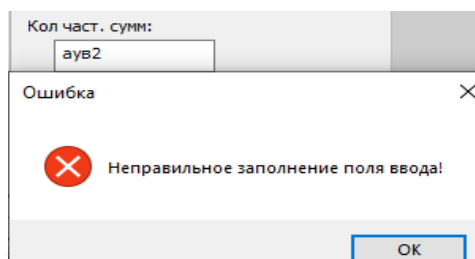


Рисунок 22 – Ошибка 2

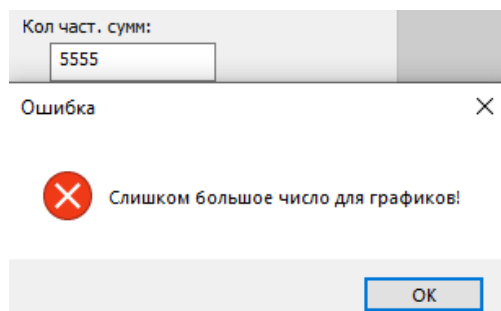


Рисунок 23 – Ошибка 3

Используя встроенные возможности языка C++ проведём замеры времени, необходимого для работы программы. Самым трудоёмким процессом в данной работе является постройка графиков. Это следует учитывать так как в работе используется функция Sleep.

В работе приведено два способа измерения времени. Для измерения времени построения графиков используется функция clock и библиотека ctime. Функция ctime возвращает строку, представляющую символьное представление даты или времени [5, 283 с; 419 с]. Для расчета всей работы используется библиотека chrono.

Важным аспектом работы является замер времени работы программы при различных вводных данных, что позволит говорить о диапазоне чисел, в котором программа работает гарантированно быстро и точно[5].

```
// Время через ctime
clock_t start1, end1;
double fintime1;
start1 = clock();
```

Рисунок 24 – Код для старта расчета времени через функцию clock

```
end1 = clock();
fintime1 = double(end1 - start1) / CLOCKS_PER_SEC;
CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT3);
CString strResult; // Преобразование числа в строку
strResult.Format(_T("%.2f"), fintime1); // Формати
pEdit1->SetWindowText(strResult); // Установка текс
```

Рисунок 25 – Код для конца расчета времени через clock и вывод результата


```
// Отображение сообщения с именем сохраненного файла
CString message;
message.Format(_T("Картинка сохранена в корневом каталоге проекта:\n\n%s"), fileName);
MessageBox(message, _T("Создание картинки"), MB_OK);
```

Рисунок 32 – Функция вывода MessageBox

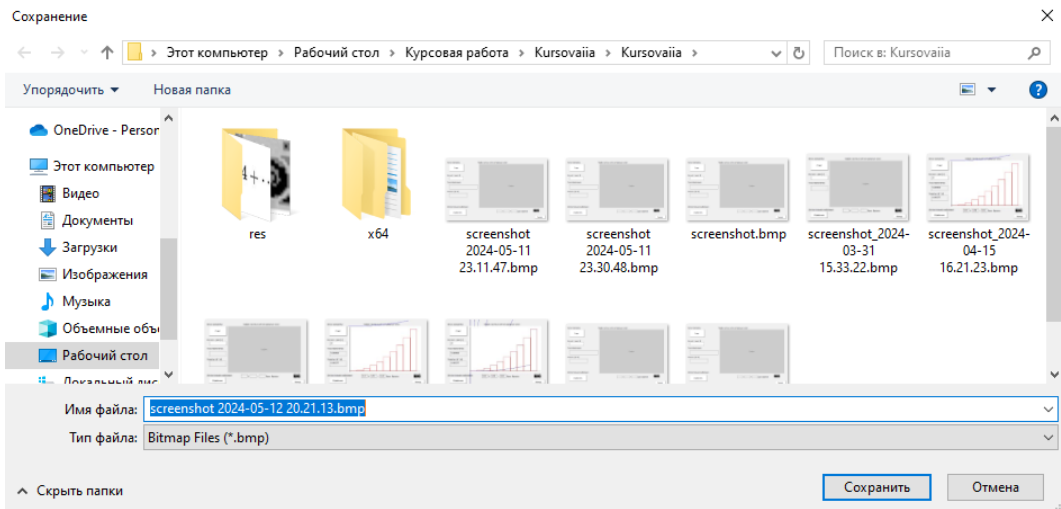


Рисунок 33 – Процесс сохранения снимка

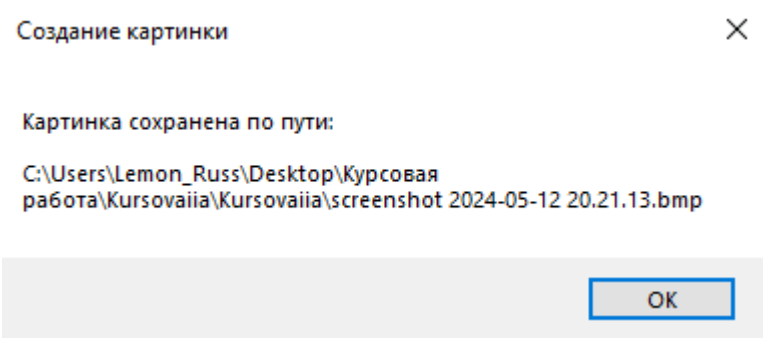


Рисунок 34 – Результат сохранения

Заключение

В результате выполнения курсовой работы была разработана программа в среде VisualStudio на языке программирования C++, которая позволяет визуализировать сглаживающую параболу частных сумм натурального ряда. Помимо этого, реализована возможность сохранения результатов работы программы в виде картинок с уникальным именем для удобства.

В процессе выполнения работы активно использовалась сторонняя литература, в основном, представленная в виде веб-ресурсов и электронных книг. Для проверки кода использовались консольные приложения. Исследованы различные математические методы, которые могут помочь реализовать график частных сумм. Были учтены такие параметры как количество частных сумм, размеры формы, время построения графика, шаги построения графика и.т.д.

Разработанная программа удовлетворяет всем поставленным требованиям, исключая в работе критических ошибок. Создавая понятные и удобные условия для работы для любого пользователя.

Список литературы

1. Г. Шилдт. С++ для начинающих. Шаг за шагом. – ЭКОМ Паблишерз, 2013. – 639 с.
2. Гредасова Н.В. Ряды : учебное пособие / Н.В. Гредасова Н. И. Желонкина, М.А. Корешникова, Е.Г. Полищук, И.Ю. Андреева.— Екатеринбург : Изд-во Урал. ун-та, 2016.— 116 с.
3. Кошев А.Н., Кузина В.В. Вычислительные методы. – ПГУАС – 2012. – 203 с.
4. Авхадиев Ф.Г. Численные методы алгебры и анализа. – Изда-во Казан. ун-та, 2019. – 200 с.
5. С/С++. Программирование на языке высокого уровня / Т. А. Павловская. — СПб.: Питер, 2003. —461 с.

Программа

```
CWnd* m_DrawArea1;
void FirstDrawGraff(CDC* pDC, int max_x, std::vector<int>& x_coordinates, std::vector<int>& y_coordinates)
{
    // Получаем окно области рисования
    CWnd* m_DrawArea1 = pDC->GetWindow();

    // Получаем размеры прямоугольника
    CRect rc;
    m_DrawArea1->GetClientRect(&rc);
    int w = rc.Width();
    int h = rc.Height();

    // Задаем начальные координаты для рисования

    int y_start = h - 10;

    // Рисуем фон
    CPen pnPenBlack(PS_SOLID, 1, RGB(0, 0, 0));
    CPen* pOldPen = dc1.SelectObject(&pnPenBlack);
    dc1.FillSolidRect(rc, RGB(255, 255, 255));

    // Рисуем ось x
    dc1.MoveTo(x_start - 5, y_start);
    dc1.LineTo(x_start + w - 15, y_start);

    // Рисуем ось y
    dc1.MoveTo(x_start, y_start);
    dc1.LineTo(x_start, y_start - h + 15);

    // Задаем цвет пера - красный
    CPen pnPenRed(PS_SOLID, 1, RGB(255, 0, 0));

    // Вычисляем ширину столбца
    int column_width = (w - x_start - 20) / max_x;

    // Задаем смещение по y

    // Цикл для рисования столбцов
    for (int i = 0; i < max_x; i++)

        // Вычисляем частичную сумму с использованием формулы  $x(x-1)/2$  (-1)
        int sum = (((i + 1) * i) / 2);
        // Вычисляем x-координату
        int x = x_start + column_width * i;

        // Проверяем, чтобы x-координата находилась в пределах области рисования
        if (x > x_start + w - 10)
            x = x_start + w - 10;
        if (x < x_start)

        // Вычисляем y-координату на основе частичной суммы, добавляем смещение и добавляем 1
        int y = y_start - int((h * sum / (50)) + y_offset);

        // Проверяем, чтобы y-координата находилась в пределах формы
        if (y < 10)
            y = 10;
        if (y > y_start)
            y = y_start;
```

```

        // Рисуем столбец
        int column_height = y_start - y;
        dc1.Rectangle(x, y, x + column_width, y_start);

        // Записываем координаты в векторы
        x_coordinates.push_back(i);
        y_coordinates.push_back(sum);
        Sleep(100);
    }
}

CWnd* m_DrawArea2;
void CKursovaiaDlg::OnBnClickedButton1()
{
    std::vector<int> x_coordinates; // Вектор для хранения x-координат точек
    std::vector<int> y_coordinates; // Вектор для хранения y-координат точек

    auto START = std::chrono::high_resolution_clock::now();

    // Время через ctime
    clock_t start1, end1; // Объявления переменных для получения текущего времени

    // Получаем количество частных сумм с клавиатуры
    CString strNumSum;
    GetDlgItemText(IDC_EDIT2, strNumSum);
    int max_x = _ttoi(strNumSum); // Преобразуем строку в число

    // Получаем количество частных сумм с клавиатуры
    GetDlgItemText(IDC_EDIT2, strNumSum);

    if (strNumSum.IsEmpty()) {
        MessageBox(L"Поле ввода пустое!", L"Ошибка", MB_OK | MB_ICONERROR);
        return; // Останавливаем выполнение кода здесь

    // Проверяем, правильно ли заполнено поле ввода

        MessageBox(L"Неправильное заполнение поля ввода!", L"Ошибка", MB_OK |

        return; // Останавливаем выполнение кода здесь

    // Проверяем, правильно ли заполнено поле ввода

        MessageBox(L"Слишком большое число для графиков!", L"Ошибка", MB_OK |

        return; // Останавливаем выполнение кода здесь

    CDC* pDC1 = Graaaf.GetDC();
    {
        FirstDrawGraff(pDC1, max_x, x_coordinates, y_coordinates);
    }
    ReleaseDC(pDC1);

    end1 = clock();
    fintime1 = double(end1 - start1) / CLOCKS_PER_SEC; // Перевод тиков в секунды (CLOCKS_PER_SEC)
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT3);

    ;// Преобразуем результат (L"%d", fintime1); // Форматируем вывод до двух знаков после запятой
    pEdit1->SetWindowText(strResult); // Установка текста в поле ввода

```

```
// Создание матрицы A и вектора b
std::vector<std::vector<double>>> A(3, std::vector<double>(3));
std::vector<double> b1(3);
int size = x_coordinates.size();
```

```
// Заполнение матрицы A и вектора b
for (int i = 0; i < size; i++) {
    double x = x_coordinates[i];
    double y = y_coordinates[i];
```

```
    A[0][0] += x * x * x * x;
    A[0][1] += x * x * x;
    A[0][2] += x * x;
```

```
    A[1][0] += x * x * x;
    A[1][1] += x * x;
    A[1][2] += x;
```

```
    A[2][0] += x * x;
    A[2][1] += x;
    A[2][2] += 1;
```

```
    b1[0] += x * x * y;
    b1[1] += x * y;
```

```
int n = A.size();
```

Решение системы уравнений с коэффициентами(n);

```
for (int i = 0; i < n; i++) {
    // Поиск максимального элемента в столбце i
    int maxIdx = i;
    for (int j = i + 1; j < n; j++) {
        if (abs(A[j][i]) > abs(A[maxIdx][i])) {
```

```
        // Перестановка строк, чтобы максимальный элемент был на диагонали
        swap(A[i], A[maxIdx]);
        std::swap(b1[i], b1[maxIdx]);
```

```
        // Приведение к верхнетреугольному виду
```

```
        double factor = A[j][i] / A[i][i];
        for (int k = i; k < n; k++) {
            A[j][k] -= factor * A[i][k];
        }
        b1[j] -= factor * b1[i];
    }
}
```

```
// Обратный ход
```

```
for (int i = n - 1; i >= 0; i--) {
    double sum = 0;
    for (int j = i + 1; j < n; j++) {
        sum += A[i][j] * coefficients[j];
    }
    coefficients[i] = (b1[i] - sum) / A[i][i];
}
```

```
// Получение коэффициентов a, b и c
```

```

double c = coefficients[2];

CString strC;
strC.Format((L"%16f"), c);
SetDlgItemText(IDC_EDIT1, strC);

double C = c * pow(10, 8);

CString strC1;
strC1.Format((L"%9f"), C);
SetDlgItemText(IDC_EDIT6, strC1);

// Время через ctime
clock_t start2, end2; // Объявления переменных для получения текущего времени
double fintime2;
start2 = clock();

// Рисуем второй график

// Получаем окно области рисования
m_DrawArea2 = Graaaf.GetDC()->GetWindow();

// Получаем размеры прямоугольника
CRect rc; //Класс для представляющий прямоугольную область
m_DrawArea2->GetClientRect(&rc);
int w = rc.Width();

// Задаем начальные координаты для рисования
int x_start = 10;
int y_start = h - 10;

// Задаем цвет пера - красный
CPen pnPenRed(PS_SOLID, 1, RGB(0, 0, 255)); // Перо для рисования - PS_SOLID тип пера
dc2.SelectObject(&pnPenRed);

// Вычисляем ширину линии
int line_width = (w - x_start - 20) / max_x;

// Цикл для рисования линии
for (int i = 0; i <= max_x; i++)

    // Вычисляем частичную сумму с использованием формулы  $x(x-1)/2$ 
    int sum = (((i + 1) * i) / 2) + c; // Добавляем сдвиг c к формуле
    // Вычисляем x-координату
    int x = x_start + line_width * (i);
    // Проверяем, чтобы x-координата находилась в пределах области рисования
    if (x > x_start + w - 10)
        x = x_start + w - 10;
    if (x < x_start)

    // Вычисляем y-координату на основе частичной суммы
    int y = y_start - int((h * sum / (50)));

    // Проверяем, чтобы y-координата находилась в пределах формы
    if (y < 10)
        y = 10;
    if (y > y_start)
        y = y_start;

// Рисуем линию

```

```

        dc2.LineTo(x, y);
        Sleep(100);
    }
    ReleaseDC(pDC2);

    end2 = clock();
    fintime2 = double(end2 - start2) / CLOCKS_PER_SEC; // Перевод тиков в секунды
(CLOCKS_PER_SEC)
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_EDIT4);

    CString strResult; // Преобразование числа в строку
    strResult.Format(L"%0.2f", fintime2); // Форматируем вывод до двух знаков после запятой
    pEdit1->SetWindowText(strResult); // Установка текста в поле ввода
}

// CHRONO
auto END = std::chrono::high_resolution_clock::now(); // Конец расчета
std::chrono::duration<double> duration = END - START; // Все время через chrono
CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT5);

CString strResult3;
strResult3.Format(L"%0.2f", duration.count()); // Форматируем вывод до двух знаков после запятой
pEdit->SetWindowText(strResult3); // Установка текста в поле ввода

```