**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА**

**И ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Смоленский филиал РАНХиГС**

Направление подготовки

09.02.07 Информационные системы и программирование

**КУРСОВАЯ РАБОТА**

по дисциплине «Разработка программных модулей»

на тему: «Разработка программных модулей для автоматизации прикладной задачи решения нелинейных уравнений методом простой итерации»

**Автор работы:**

студент 3 курса

очной формы обучения

группа 27/11-К/ИТО

\_\_\_\_\_\_\_\_\_\_\_/ Махницкий Д.С.

(подпись)

**Преподаватель:**

\_\_\_\_\_\_\_\_\_\_\_/ Харламов П.С.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Смоленск 2023

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ**

В процессе решения задач высшей математики может возникнуть потребность в быстром и удобном получении результата некоторых уравнений, которое может быть линейным и нелинейным. Нелинейные уравнение по своей сути гораздо сложнее, т.к. они содержат в себе более сложные аспекты вычислений, что делает их автоматизацию востребованной и не простой. Численные методы решения нелинейных уравнений достаточно сложны для того, чтобы их можно было легко понять, поэтому здесь потребуются базовые знания в области высшей математики.

Существует три основных метода решения нелинейного уравнения: аналитический, графический и численный. Не все нелинейные уравнения можно решить аналитическим способом, а графический способ для некоторых уравнений не точен или требует пояснений. Данные минусы отсутствуют у численных методов, в частности у метода простой итерации.

Решение нелинейного уравнения численным методом, в качестве которого используется метод простой итерации, имеет свои недостатки, которые вскоре будут раскрыты. Название метода тесно связано с его смыслом: каждую итерацию вычисляется результат, который точнее предыдущего, а простой итерация называется потому, что для получения следующего результата достаточно применить к предыдущему лишь одну функцию.

Объектом исследования является численный метод решения нелинейного уравнения под названием метод простой итерации. Предметом исследования служит разработка программных модулей для решения этого уравнения данным методом.

Цель курсовой работы заключается в разработке качественного программного обеспечения модульной архитектуры для решения поставленной прикладной задачи.

Задачи курсовой работы следующие:

- описать прикладную задачу решения нелинейных уравнений методом простой итерации,

- проанализировать предметную область и поставить проект разработки программных модулей,

- описать разработанные программные модули для автоматизации прикладной задачи,

- построить схемы взаимосвязи программных модулей и массивов данных разработанного программного обеспечения.

В качестве методологической основы в работе использован метод анализа: перед началом написания проведен анализ предметной области – раздела высшей математики, относящейся к численным методам. Так же использован метод синтеза: обработанная информация собирается в различные выводы.

Информационную базу данной работы составляет техническая литература по нескольким основным темам:

- высшая математика,

- численные методы решения уравнений,

- разработка программного обеспечения на C#,

- библиотека Windows.Forms,

- платформа .NET Framework.

**1 ТЕОРЕТИЧЕСКИЕ И АНАЛИТИЧЕСКИЕ АСПЕКТЫ АВТОМАТИЗАЦИИ ПРИКЛАДНОЙ ЗАДАЧИ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ПРОСТОЙ ИТЕРАЦИИ**

**1.1** **Описание прикладной задачи решения нелинейных уравнений методом простой итерации**

Нелинейное уравнение представляет собой уравнение, не являющееся линейным [1]. То есть, любое уравнение, содержащее логарифмические, показательные или тригонометрические функции, является нелинейным. Пример нелинейного уравнения представлен в формуле (1).

Решение нелинейного уравнения означает нахождение всех таких значений x, при которых оно будет верным равенством [2, 3]. Стандартный вид полинома, являющегося нелинейным уравнением отображен в формуле (2).

Метод итераций [3, 4] представляет собой численный метод решения математических задач, в котором значение переменной после выполнения каждой итерации [2, 5] приближается к верному ответу. Другими словами, суть метода заключается в нахождении по приближённому значению величины следующего приближения, являющегося более точным. Метод позволяет получить значения корня с заданной точностью в результате итерационного процесса [3, 7]. Характер сходимости [8] и сам факт сходимости метода зависит от выбора начального приближения корня. Численное решение нелинейного уравнения методом простой итерации сводится к двум пунктам: отделение корней [1, 6]; нахождение корней.

Отделение корней представляет собой нахождение интервалов изоляции корней [2, 4], на которых располагаются корни. Один интервал содержит один корень.

Далее рассмотрен алгоритм нахождения приближения корня нелинейного уравнения на определенном интервале методом простой итерации.

Обозначения: – первая точка отрезка, – вторая точка отрезка, – начальное приближение к корню, – следующее приближение к корню, – разность итераций, – требуемая точность [5, 9].

Во-первых, необходимо привести уравнение из вида (3) к виду (4).

Данное действие выполняется переносом x в левую часть уравнения.

Во-вторых, проверить уравнение на сходимость [8, 9]. Допустим, на отрезке [a, b] расположен корень уравнения (3), и при том единственный. Задаем начальное приближение к корню в видах (5,6).

Если проверка ни в одном из интервалов не оказалось истинной, то решить данное уравнение методом простых итераций на заданном промежутке невозможно.

В-третьих, вычислить следующее приближение.

Если найден, то вычисляем следующее приближение по формуле.

Данный процесс можно продолжать либо до требуемой точности в k итераций, либо до достижения точности e, либо вообще бесконечно. В данном алгоритме рассматривается процесс приближения до достижения заданной точности e.

В-четвертых, проверить условие выхода из итерационного цикла. Вычисляем разность итераций по формуле (8).

Если , то , а также переходим к третьему этапу вычислений. Иначе, – искомое приближение к корню с требуемой точностью или ответ.

Таким образом, получим формулу (9).

Рассмотрим нелинейное уравнение (10).

Допустим, a = 0, b = 1. То есть, корень уравнения (10) расположен на отрезке [0, 1], и при том единственный. За требуемую точность возьмем e, равное 0,001.

Во-первых, левая часть уравнения и есть , то есть (11).

Необходимо привести уравнение (11) к виду (4), что отражено далее (12).

Таким образом, (13).

Во-вторых, проверяем уравнение на сходимость. В данном случае . Вычислим (14).

Значение уравнения (14) соответствует условию (5), откуда следует (15).

Переходим к третьему пункту вычислений, так как приближенное значение было найдено.

В-третьих, вычисляем следующее приближение (16) согласно формуле (9).

В-четвертых, проверяем условие выхода из итерационного цикла. Вычисляем R (17) по формуле (8).

Проверим условие (18).

Оно является истинным. Значит присваиваем значение . Получаем равно 0,2. Переходим к третьему этапу и повторяем данную итерацию до тех пор, пока не получим значение требуемой точности. В таблице 1 отображены значения x1 на каждой итерации, а также соответствующее значение R.

Таблица 1 – Значения приближения x1 на каждой итерации

|  |  |  |
| --- | --- | --- |
| Номер итерации |  |  |
| 1 | 0,20000 | 0,20000 |
| 2 | 0,24428 | 0,04428 |
| 3 | 0,25534 | 0,01105 |
| 4 | 0,25818 | 0,00283 |
| 5 | 0,25891 | 0,00073 |

Пятая итерация оказалась последней, так как в ней . Соответственно, значение в пятой итерации является приближенным корнем уравнения (10). Произведем проверку. Подставим значение x1 из пятой итерации в уравнение (10). Получим следующее уравнение (19).

Его значение приближенно равно 0,00095 или же равно 0 при заданной точности, а значит приближение корня вычислено корректно.

Таким образом, произведено описание прикладной задачи решения нелинейных уравнений методом простой итерации. Однако, поскольку разработка происходит в учебных целях, допустим, что она осуществляется только для нелинейного уравнения вида полинома максимальной степени четыре.

**1.2 Анализ предметной области и постановка проекта разработки программных модулей**

Назначение системы заключается в удобном нахождении приближения корня на заданном интервале [a, b]. Пользователю доступен ввод следующих данных:

- степень полинома,

- интервал изоляции корня,

- требуемая точность приближения,

- коэффициенты полинома.

Ввод степени полинома доступен в диапазоне от первой до четвертой степени, только целочисленные значения. Интервал изоляции корня представляет собой два вещественных числа a и b, между которыми расположен один из корней полинома. Условия для ввода: a должно быть меньше b, иначе система выдаст уведомление о некорректном вводе. При отсутствии корня на интервале [a, b] пользователь получит об этом сообщение [14, 15]. Требуемая точность вводится в виде вещественного числа в диапазоне от 1 до . Наибольшее и наименьшее значение коэффициентов полинома, а также и интервала изоляции, ограничивается диапазоном возможных значений используемого типа данных.

Цель разрабатываемой системы заключается в удобном и корректном решении рассматриваемой прикладной задачи. Для достижения данной цели, разрабатываемая система должна реализовать следующие функции:

- ввод данных,

- проверка корректности данных,

- вычисление приближение корня,

- очистка данных,

- вывод результата.

Разрабатываемая информационная система будет полезна для широкого круга пользователей, включая ученых, инженеров, школьников и студентов, занимающихся численными методами.

На рисунке 1.1 изображена модель в нотации IDEF0, а на другом рисунке 1.2 ее декомпозиция первого уровня.

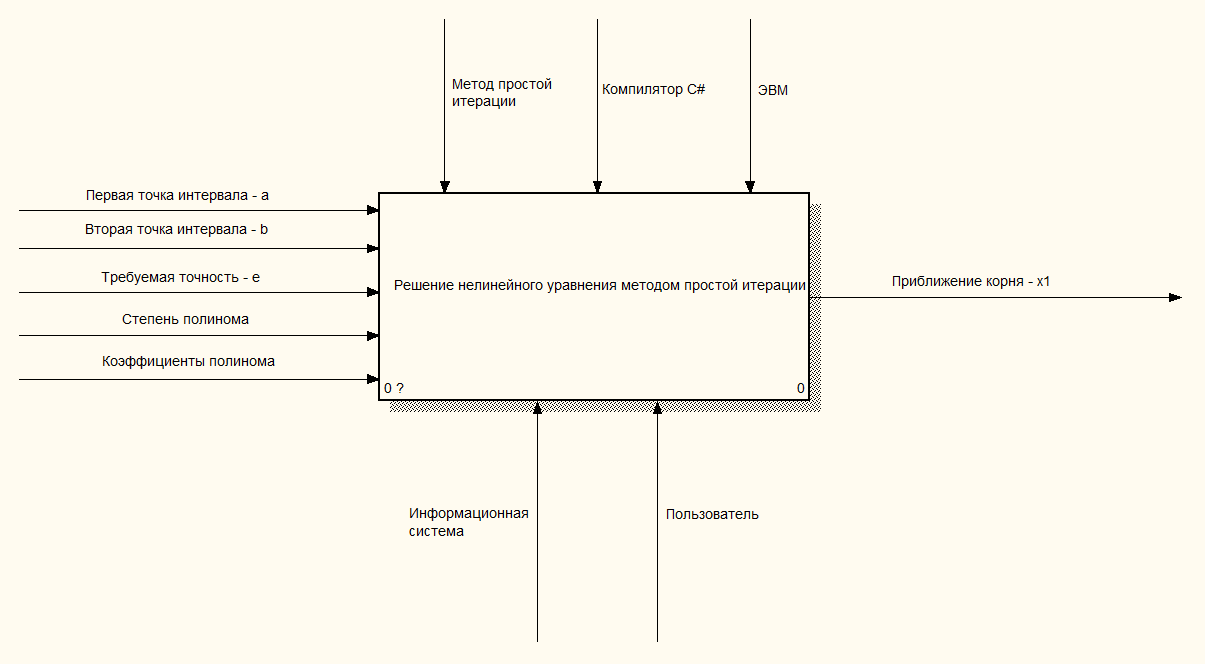


Рисунок 1.1 – Модель в нотации IDEF0

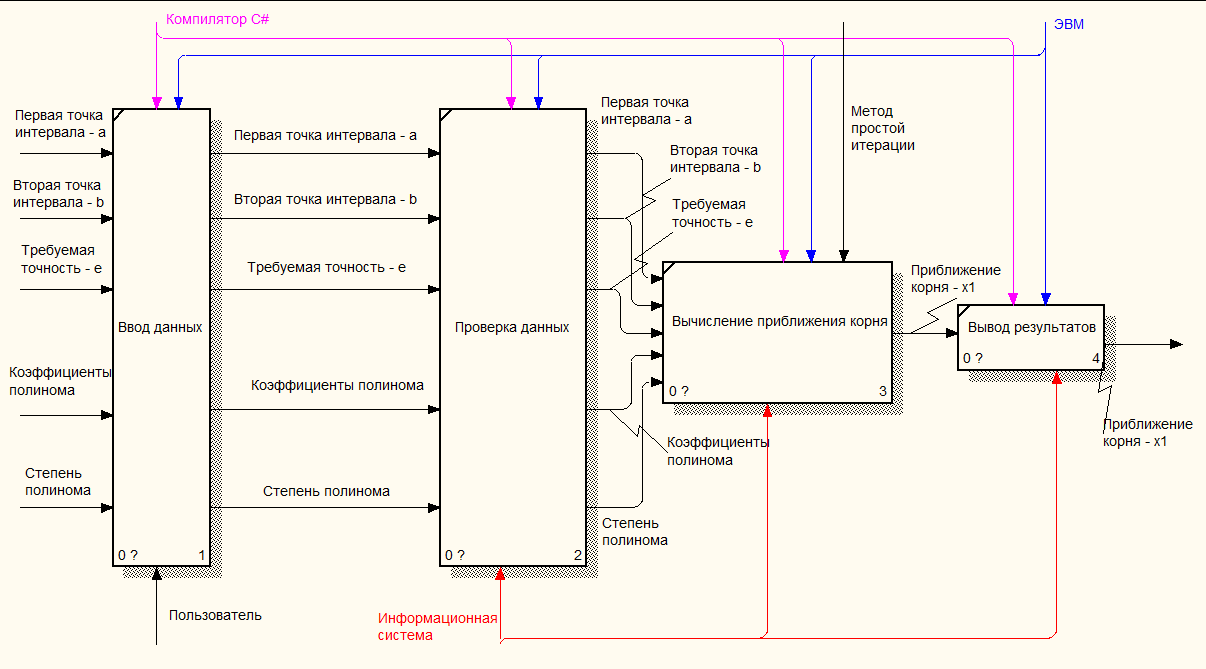


Рисунок 1.2 – Декомпозиция модели в нотации IDEF0

Далее разработана UML-диаграмма деятельности, которая изображена на рисунке 1.3.

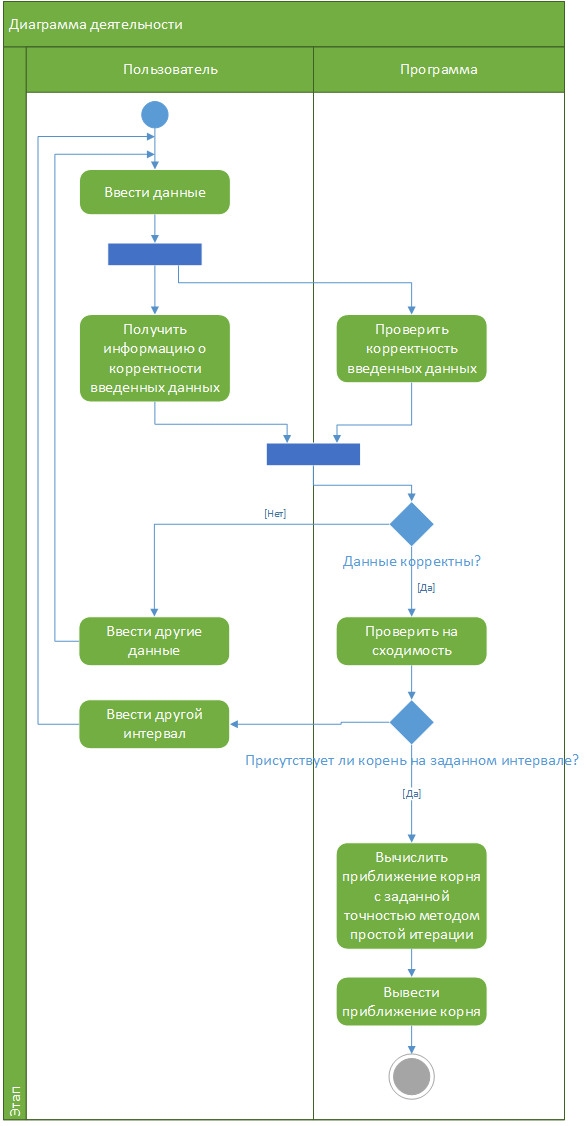


Рисунок 1.3 – UML-диаграмма деятельности

На рисунке 1.4 изображена диаграмма Ганта, а на рисунках 1.5 – 1.7 соответствующие ей данные о затратах в виде двух диаграмм и таблицы.

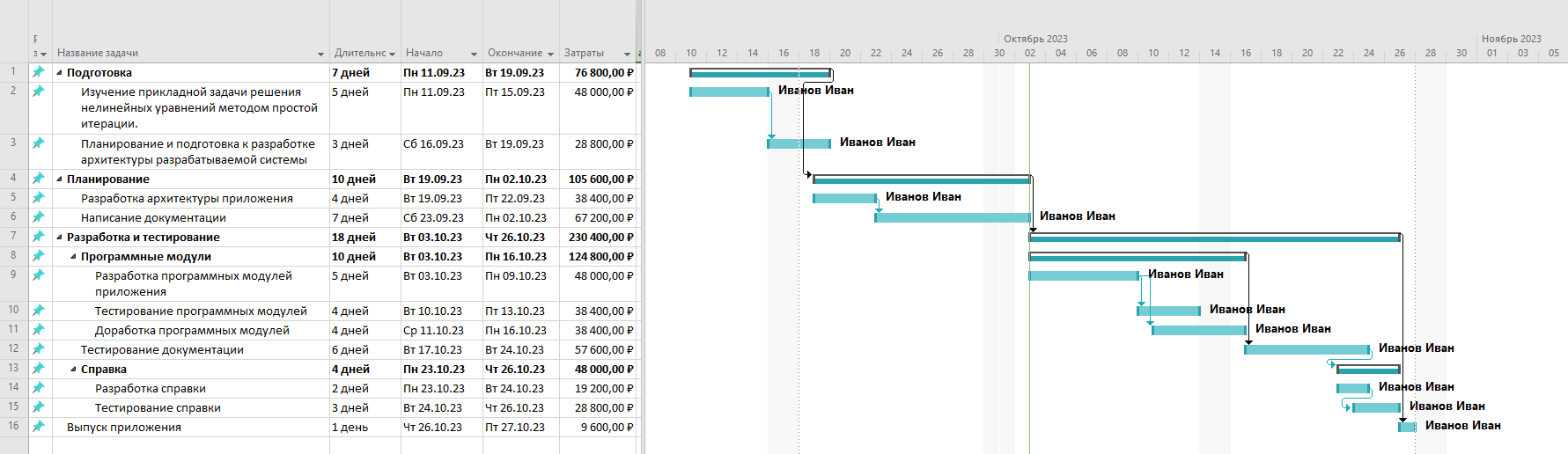


Рисунок 1.4 – Календарное планирование в MS Project

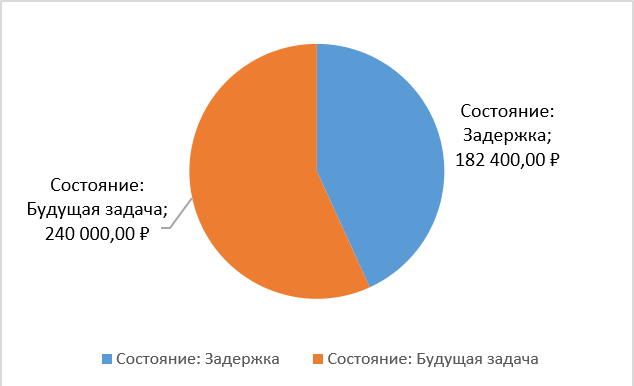


Рисунок 1.5 – Распределение затрат

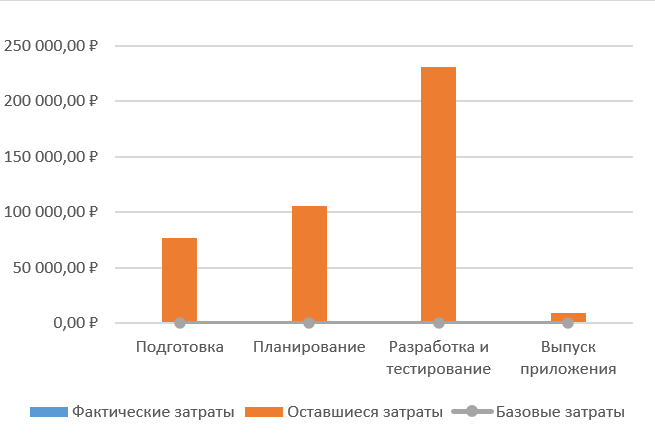


Рисунок 1.6 – Состояние затрат

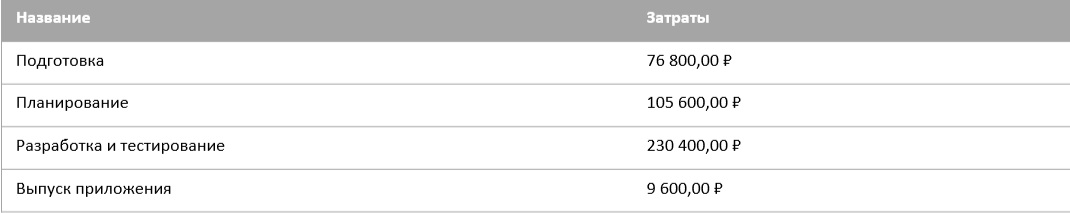


Рисунок 1.7 – Сведения о затратах

Затем разработана концептуальная модель данных, отображенная на рисунке 1.8.

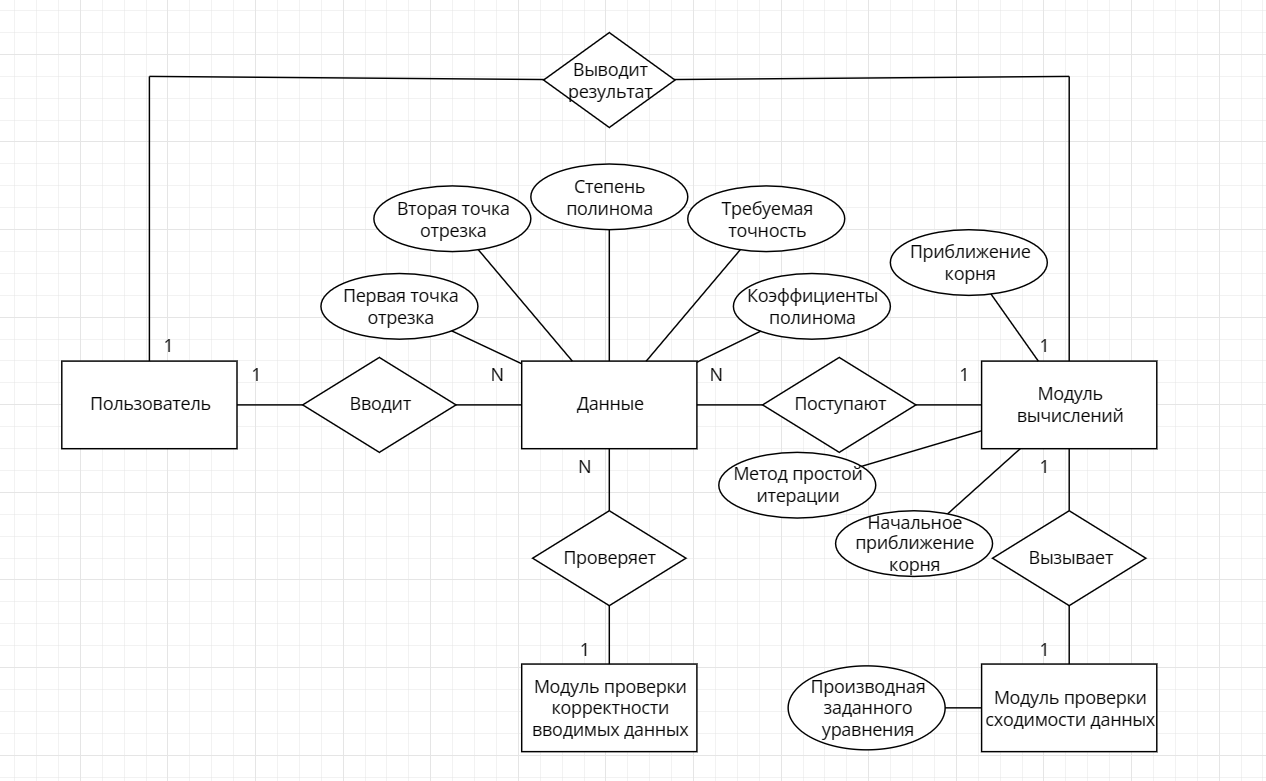


Рисунок 1.8 – Концептуальная модель данных в нотации Чена

Таким образом, в данном подразделе был проведен анализ предметной области, обозначены ограничения на вводимые пользователем данные, установлена стоимость разработки, а также наглядным образом с помощью диаграмм изображены различные внутренние процессы разрабатываемого приложения.

Выводы по разделу

В начале раздела описана прикладная задача для автоматизации с математической и концептуальной точки зрения, далее был рассмотрен пример решения задачи, установлены некоторые ограничения, разработаны различные диаграммы, описывающие разные аспекты поведения основной программы.

**2 РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ ДЛЯ**

**АВТОМАТИЗАЦИИ ПРИКЛАДНОЙ ЗАДАЧИ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ПРОСТОЙ ИТЕРАЦИИ**

**2.1 Описание программных модулей для автоматизации прикладной задачи решения нелинейных уравнений методом простой итерации**

Разработка формы проводилась в программе Visual Studio на языке программирования C# с помощью библиотеки классов Windows.Forms на платформе .NET Framework 4.7.2 [10, 13, 14]. В процессе разработки системы было написано 12 модулей включая конструктор главной формы. Далее слова метод [15] и модуль носят единый смысл – атомарный фрагмент кода, выполняющийся неделимо. Если далее по тексту аргументы метода не упоминаются, то их у него нет. Четыре метода служат обработчиками событий для объектов формы:

- метод обработки нажатия клавиши в текстовом поле – один для всех элементов данного типа,

- метод обработки события переключения элемента RadioButton - один для всех элементов данного типа в групповом поле groupBox1,

- метод обработки события нажатия кнопки “Вычислить”,

- метод обработки события нажатия кнопки “Очистить все”.

Начнем с первого модуля из представленного сверху списка: он носит название AnyCoeff\_KeyPress, принимает стандартные аргументы для события KeyPress [12, 13]. В данном модуле происходит всевозможная обработка вводимых пользователем значений: ввод букв, посторонних символов и прочих значений, не являющихся вещественным, не поддерживается. Под этими словами имеется ввиду, например, отсутствие возможности ввести букву или несколько знаков «-» подряд, а обработка называется всевозможной, т.к. у пользователя все равно есть возможность ввести некорректные данные – например просто не вводить ничего: преобразования пустой строки в тип double выдает исключение FormatException [11]. Таким образом, на мой взгляд, данный модуль защищен от плохого пользователя.

Рассмотрим следующий модуль из списка под названием AnyRadioButton\_CheckedChanged – он обрабатывает событие CheckedChanged в случае, если переключатель был переключен в состояние true, иначе возвращает управление; используя объект, вызвавший событие, он устанавливает значение для целочисленного поля degree значение, означающее степень полинома, которую выбрал пользователь. Также он выключает неиспользующиеся в уравнении текстовые поля и метки [13, 15], например если пользователь выбрал первую степень уравнения, то ввод коэффициента для x^4 ему фактически запрещен, так что все поля и метки, отвечающие за степень полинома выше первой будут отключены с помощью строки obj.Visible = false; где obj – неиспользующееся текстовое поле или метка. Отсюда плавно вытекает описание метода DisableTextBoxesAndLabels, который имеет целочисленный аргумент amount, определяющий количество пар “текстовое поле – метка”, которые необходимо отключить, начиная с левой стороны. Текстовые поля перед отключением получают значение string.Empty для свойства Text; простыми словами, если в этом поле было число, то оно пропадает. Для отключения лишних элементов используются вспомогательные массивы [10, 14]. Так же в начале метода присваивается значение для целочисленного поля coefficientsAmount, которое понадобится в дальнейшем.

Следующий модуль ClearAll\_Click – обработчик события второй кнопки, который устанавливает всем текстовым полям значение string.Empty, выбирает первую степень полинома, что делает ее степенью по умолчанию, а также устанавливает первое значение точности в выпадающем списке – один. Выпадающий список – элемент под названием comboBox1, содержимое которого – степень числа десять от одного до минус десяти, то есть он содержит десять элементов, нулевой из которых – 1, первый – 0,1, второй – 0,01 и т.д.

Последний обработчик событий в данной программе – метод Calculate\_Click, имеющий стандартные аргументы для события, точно так же как события ClearAll\_Click и AnyRadioButton\_CheckedChanged; данный метод с технической точки зрения используется только для вызова других методов, с фактической – для вычисления приближения корня. Он сначала вызывает модуль ввода данных, затем модуль проверки данных на сходимость и вычисляет результат, если данные подходят, в противном случае – выводит сообщение либо о некорректности данных, либо о невозможности решения данного уравнения методом простой итерации.

Модуль ввода данных именуется DataInput, возвращает значение true если данные введены корректно с точки зрения типов вводимых значений и false – если нет. Сначала метод инициализирует массив коэффициентов, затем заполняет их с помощью вспомогательных массивов. Затем происходит считывание интервала изоляции корня и запись его значения в переменные a и b, также считывается значение требуемой точности e, которое сохраняется в переменную requiredAccuracy (значение e хранится в переменной под этим названием во избежание конфликтов с аргументами событий, которые тоже называются e) и используется далее в программе; проверяется условие a больше или равно b. Если считывание данных вызывает исключение или условие истинно, то происходит возврат false.

Методы Construct\_g и Construct\_gDerivative возвращают делегат [13, 14, 15] пользовательского типа Function, который принимает и возвращает один аргумент типа double. Данные методы созданы для более простого понимания происходящего в блоке вычислений. Так как количество одночленов в уравнении задается динамически, то необходим специальный конструктор функции (Здесь и далее под словом функция имеется в виду экземпляр делегата Function, принимающий и возвращающий одно значение типа double) и ее производной. Метод Construct\_g создает функцию из формулы (4), а метод Construct\_gDerivative – ее производную, которая используется в формулах проверки на сходимость (5, 6). Метод Construct\_g сначала переносит x в другую часть уравнения, а затем возвращает делегат на основе лямбда-выражения, в котором используется это уравнение. Второй модуль Construct\_gDerivative создает производную, используя лямбда-выражение; руководствуется определением: «Производная суммы равна сумме производных».

Модуль CheckDataForConvergence выполняет проверку сходимости данных. Если данные подходят для метода простой итерации, он возвращает true, иначе - false. Алгоритм работы данного метода будет рассмотрен далее, пока что будет достаточно информации о том, что данный метод инициализирует делегаты g и gDerivative с помощью последних двух рассмотренных модулей. Следующий модуль – CalculateApproximationToRoot, в нем и происходит вычисление методом простой итерации, что тоже будет рассмотрено далее. Данный метод возвращает значение double, которое хоть и не присваивается в программе, но может быть использовано в дальнейшем, например, для записи в какую-нибудь переменную или массив.

Последним рассмотренным модулем будет ResultOutput – он выводит результат работы программы, который может быть трех типов, в зависимости от принимаемого аргумента:

1. Вывод приближения корня требуемой точности. Данные корректны и подходят по сходимости.
2. Данные не подходят по сходимости (Например, единственный корень уравнения расположен на промежутке [-94; -93,5], а пользователь ввел промежуток [-1,937; 105,15]). Можно попробовать задать другой интервал изоляции или ввести другие данные [3].
3. Введены некорректные данные. Такие данные не могут храниться в переменных типа double, либо нарушена математическая составляющая. (Например, при второй степени полинома введен коэффициент ноль для x^2).

На рисунке 2.1 изображена запущенная главная форма программы. Всего в форме было использовано семь текстовых полей, у которых свойству MaxLength [10] было присвоено значение семь, то есть пользователь может ввести не более семи символов в каждое. Коэффициенты и корень изоляции вводятся с помощью клавиатуры, требуемая точность и степень полинома выбирается щелчком мыши. Снизу две кнопки, выполняющие соответствующие их тексту действия. При изменении степени полинома появляются или исчезают поля и метки слева от уравнения первой степени, которое присутствует на форме постоянно.

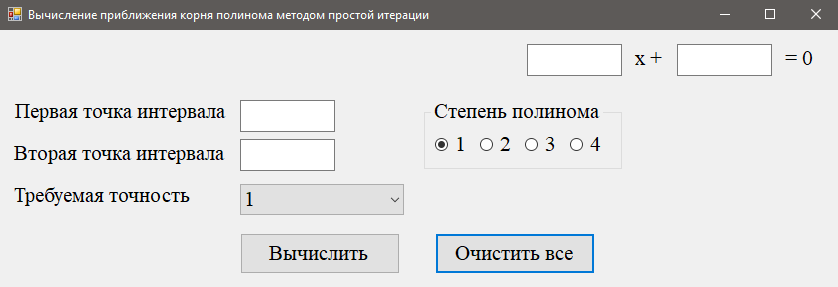


Рисунок 2.1 – Главная форма программы

Далее с помощью программы MS Visio было разработано дерево программных модулей, которое изображено на рисунке 2.2. Здесь по названиям модулей понятно, что при вызове модуля AnyRadioButton\_CheckedChanged, или же при генерации события переключения одного из переключателей, происходит вызов модуля отключения текстовых полей и меток, что было описано ранее. Так же благодаря наглядности данной диаграммы становится очевидным тот факт, что модуль Calculate\_Click служит в качестве посредника для вызова остальных вычислительных модулей. Всего на диаграмме изображено 13 модулей, хотя и описывалось ранее всего лишь 12. Объясняется это тем, что модуль InitializeComponent генерируется средой разработки автоматически [14, 15] для удобного визуального редактирования объектов формы, изменять содержимое данного модуля рекомендуется только опытным пользователям. Так же можно заметить, что метод CheckDataForConvergence не использует делегат g, но имеет в своем распоряжении вызов модуля его построения. Дело в том, что для проверки данных на сходимость нужна производная функции и так как проверка выполняется до вычисления, где используется просто функция, с моей точки зрения кажется правильным сначала сконструировать функцию, а лишь затем ее производную. Конструирование функции включает в себя блок кода, который «перемещает» x в другую часть уравнения, без чего построение производной будет некорректным. Данный блок кода можно вынести в отдельный метод, и вызывать его отдельно, получив прирост производительности при небольшом изменении архитектуры всей программы, но данные действия не были выполнены на эмпирических основаниях: нельзя сначала сконструировать производную функции, а только потом саму функцию.

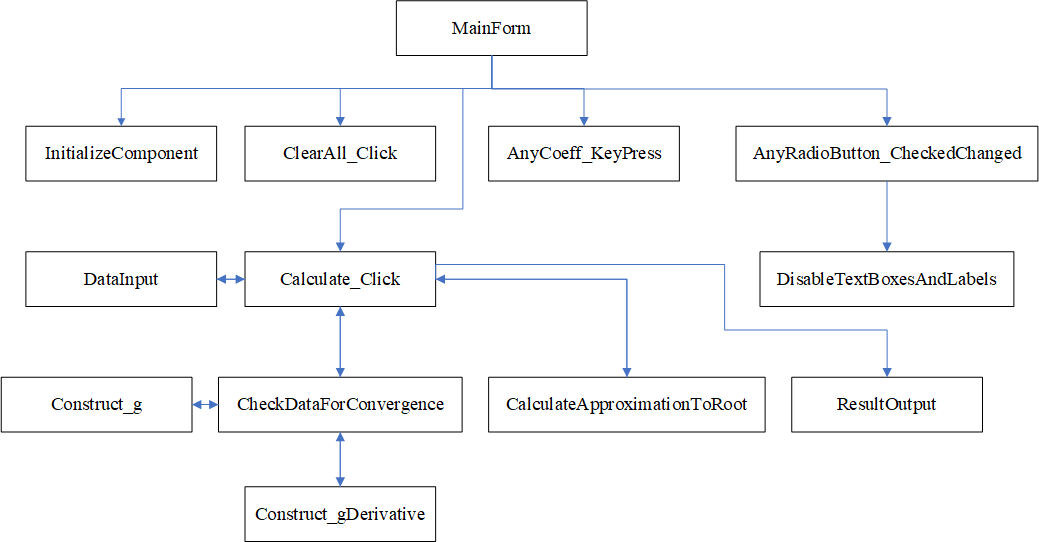


Рисунок 2.2 – Дерево программных модулей

Таким образом, разработанная архитектура программы носит модульный характер и имеет некоторую расширяемость, что делает программный код гибким для дальнейшей поддержки, модификации и обновления приложения.

**2.2 Построение схемы взаимосвязи программных модулей и массивов данных для решения прикладной задачи**

В программе описан единственный класс под названием MainForm. Он содержит закрытые поля разных типов, которые относительно данного класса являются глобальными переменными. Данные переменные кратко описаны в таблице 2. Код, генерируемый средой разработки [11, 13] для конструктора формы, не рассматривается. В основном программные модули программы используют поля класса в качестве данных для взаимодействия.

Таблица 2 – Глобальные переменные и их краткое описание

|  |  |  |
| --- | --- | --- |
| Тип переменной | Название переменной | Краткое описание переменной |
| double | a | Первая точка интервала изоляции |
| double | b | Вторая точка интервала изоляции |
| double | requiredAccuracy | Требуемая точность |
| int | degree | Степень полинома |
| double[] | coefficients | Коэффициенты полинома |
| double | x0 | Начальное приближение корня |
| double | x1 | Следующее приближение корня |
| double | R | Переменна для проверки условия выхода из цикла |
| int | coefficientsAmount | Количество коэффициентов у полинома |
| Function | g | Функция для вычисления x1 |
| Function | gDerivative | Функция для проверки данных на сходимость |
| TextBox[] | textboxes | Хранит все текстовые поля для ввода коэффициентов |
| Label[] | labels | Хранит все текстовые метки, выводящие x в некоторой степени |

Степень полинома degree взаимодействует с тремя методами, ее значение устанавливается через обработчик события AnyRadioButton\_CheckedChanged, а затем используется двумя модулями для построения функции и ее производной. По сути, данная переменная создана для более простого понимания кода, т.к. ее значение равно k минус один, где k – количество коэффициентов уравнения.

Функция g создается и используется в модуле CheckDataForConvergence с помощью метода Construct\_g, так же используется для подсчета приближения корня. Значение поля coefficientAmount устанавливается в методе DisableTextBoxesAndLabels и используется далее для инициализации массива coefficients. Массивы textboxes и labels созданы для хранения некоторых текстовых полей и меток, что необходимо для удобной реализации выключения ненужных элементов формы при изменении пользователем степени полинома.

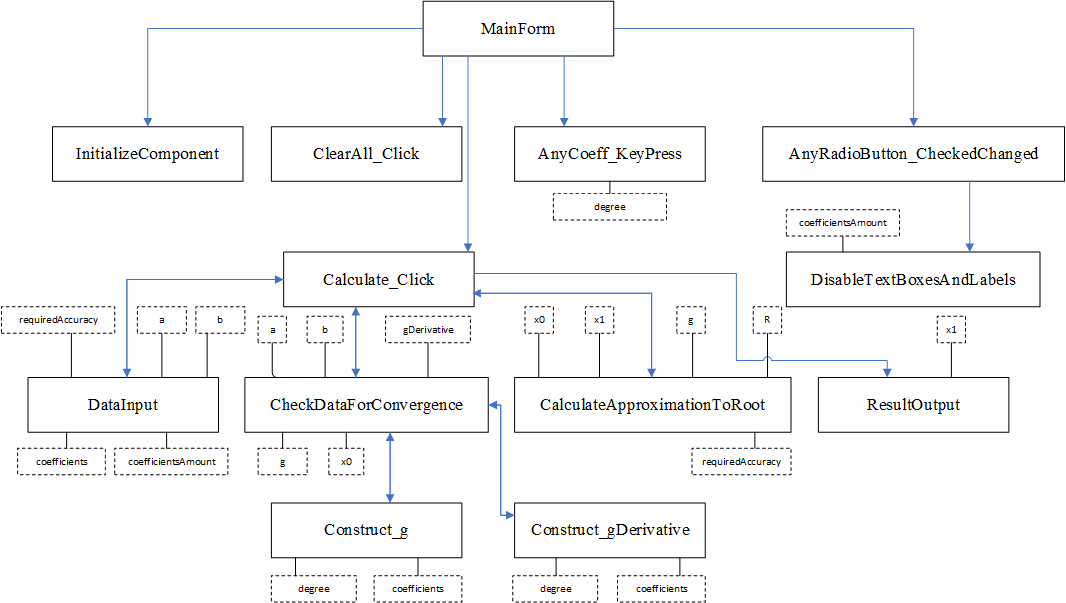


Рисунок 2.3 – Схема взаимосвязи программных модулей и массивов данных

В результате выполнения данного подраздела описано назначение используемых переменных, не считая те переменные, которые генерируются автоматически, например переменная, хранящая ссылку на экземпляр объекта текстового поля или выпадающего списка. Схема взаимосвязи программных модулей, построенная на рисунке (2.3), демонстрирует, что большая часть данных используется в методах, вызываемых модулем Calculate\_Click.

**2.3 Описание алгоритмов работы программных модулей**

Далее будут рассмотрены алгоритмы работы нескольких программных модулей. Алгоритмы определенных методов не нуждаются в рассмотрении, среди которых, например: InitializeComponent, MainForm и ClearAll\_Click. Модуль MainForm – единственный конструктор класса, который вызывает InitializeComponent и инициализирует значения объектов формы при запуске. Алгоритм метода InitializeComponent не рассматривается, так как он генерируется средой разработки автоматически, а не пишется самим программистом. Алгоритм модуля ClearAll\_Click настолько прост, что не нуждается в рассмотрении: его задача сфокусирована на установке определенных значений по умолчанию для объектов формы.

Алгоритм модуля Calculate\_Click доступен для рассмотрения на рисунке (2.4) в виде блок-схемы. Модуль AnyCoeff\_KeyPress так же изображен на рисунке (2.5).

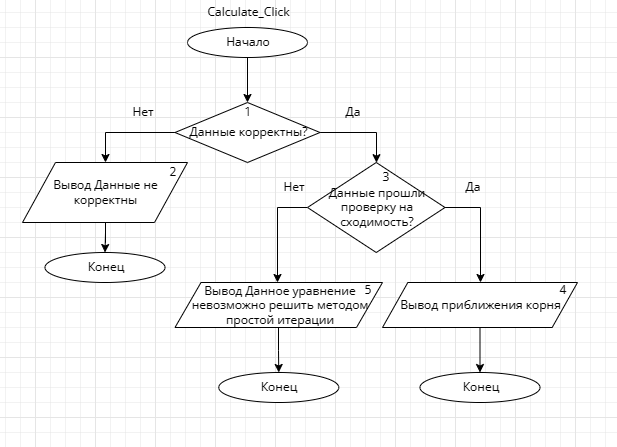


Рисунок 2.4 – Алгоритм работы метода Calculate\_Click

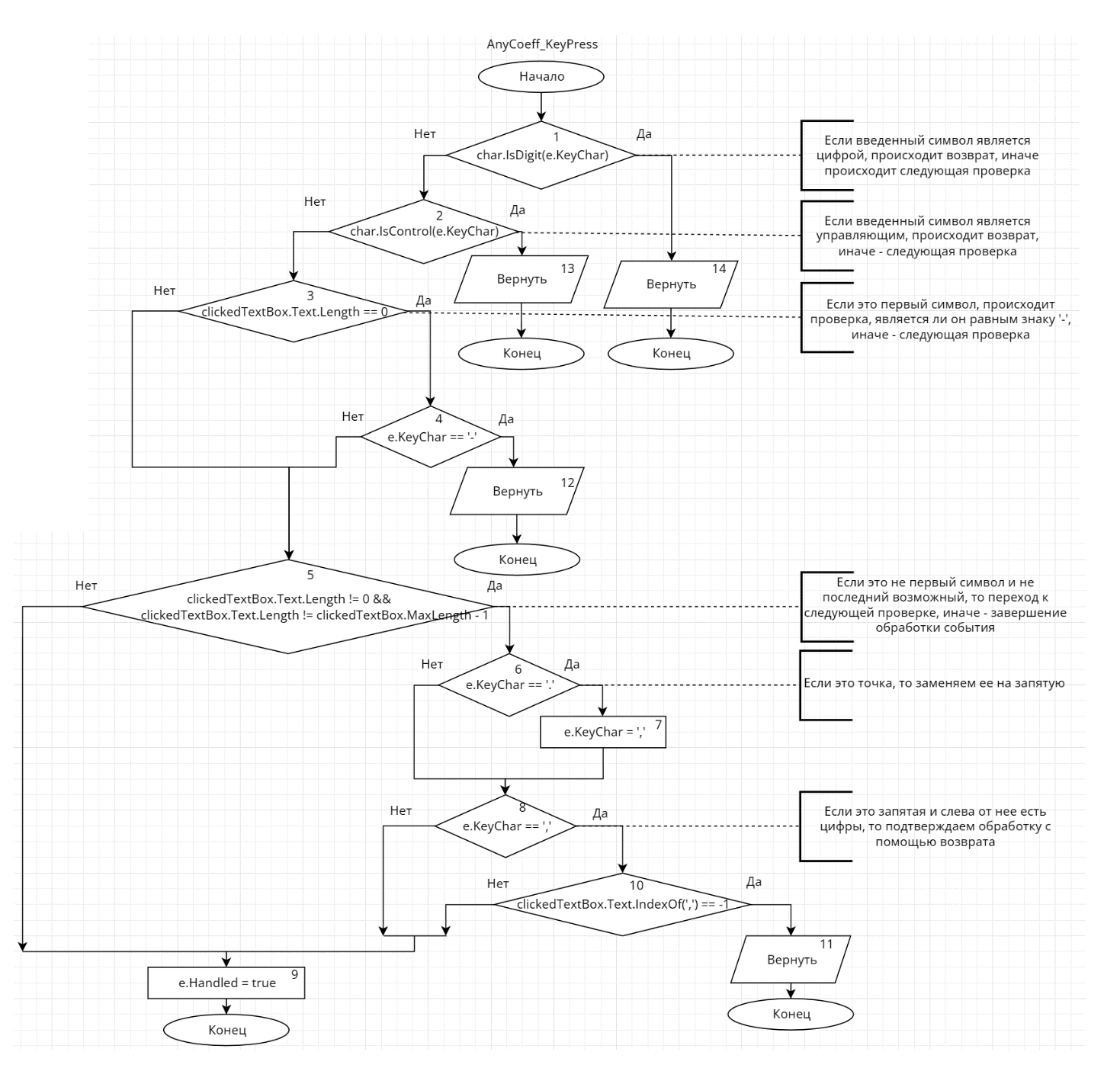
****

Рисунок 2.5 – Алгоритм работы метода AnyCoeff\_KeyPress

**2.4 Способы и результаты тестирования программных модулей в различных режимах**

Способ тестирования – простой ввод данных

Результаты тестирования программы представлены в приложении Б. На рисунке (Б.1) отображена программа с введенными корректными данными, результат вычисления представлен на рисунке (Б.2). На рисунке (Б.3) показана форма с введенными некорректными данными, среди которых пустые поля и не подходящий по условию интервал изоляции корня. В таком случае при нажатии на кнопку «Вычислить» приложение уведомит пользователя о неправильных данных.

Выводы по разделу

В результате выполнения второго раздела были выполнены поставленные задачи разработки программных модулей для поставленной задачи. В первом подразделе всего разработано и рассмотрено 12 модулей, каждый из которых имеет свои особенности. Подробно рассмотрены самые важные черты разработанных модулей. Во втором подразделе присутствует схема взаимосвязи программных модулей и массивов данных, показывающая наглядно взаимодействие модулей между друг другом. Данная схема дает читателю понять, насколько один модуль зависит от какого – либо другого, или наоборот, насколько определенный модуль независим от других.

...

Так как выполнены цели всех подразделов, то можно считать выполненной цель и самого раздела.

**ЗАКЛЮЧЕНИЕ**

В процессе написания данной работы

Таким образом, все поставленные в курсовой работе задачи выполнены, а, соответственно, цель курсовой работы достигнута.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бакушинский А.Б. Некорректные задачи. Численные методы и приложения. М. : 2022. 862 c.

2. Зверев В.А. Выделение сигналов из помех численными методами. М. : 2022. 863 c.

3. Калиткин Н.Н. Численные методы. М. : 2023. 878 c.

4. Костомаров Д.П. Программирование и численные методы. М. : 2022. 996 c.

5. Ортега Дж. Введение в численные методы решения дифференциальных уравнений. М. : 2018. 949 c.

6. Баврин И. И. Высшая математика. М. : 2004. 400 c.

7. Балдин К. В. Высшая математика М. : 2010. 360 c.

8. Зайцев И. А. Высшая математика / И.А. Зайцев. - М. : Дрофа, 2004. 400 c.

9. Ровба Е. А. Высшая математика. М. : 2012. 902 c.

10. Дубовцев А. Microsoft .NET в подлиннике. М. : 2023. 704 c.

11. Мэтью Д. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов. М. : 2021. 877 c.

12. Нейгел К. C# 4.0 и платформа .NET 4 для профессионалов. М. : Вильямс, 2023. 209 c.

13. Рихтер Д. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. М. : Питер, 2018. 896 c.

14. Симан М. Внедрение зависимостей в .NET. - М. : Питер, 2020. 435 c.

15. Троелсен Э. Expression Blend 4 с примерами на C# для профессионалов. М. : Диалектика / Вильямс, 2020. 828 c.

**ПРИЛОЖЕНИЕ А**

**ГЛОССАРИЙ**

Нелинейное уравнение – уравнение, которое не является линейным.

Полином (или многочлен) – это сумма одночленов.

Корень уравнения – такое значение переменной, при которой данное равенство будет верным.

Интервал изоляции – диапазон, содержащий единственный корень.

Численные методы – методы решения математических задач, которые позволяют получать результат в численном виде, обычно приближенном.

Итерация – полное прохождение тела цикла.

Вещественное число – это число, у которого есть дробная часть.

IDEF0 – нотация графического моделирования, которая используется для создания функциональной модели, отображающей информационные потоки, которые связывают материальные объекты.

UML-диаграмма деятельности – графическое изображение процессов, задач, которые выполняются в рамках определенной деятельности.

Диаграмма Ганта – линейная диаграмма с временной шкалой. Позволяет наглядно изобразить все этапы проекта.

Компилятор – программа, преобразующая код, написанный человеком на определенном языке программирования, в машинный код.

Программный модуль – цельная, независимая часть программы, которая выполняет определенную функцию.

Переменная – ячейка в памяти компьютера, которая используется для хранения определенной информации и имеет тип.

Делегат – это тип данных в языке C#, хранящий ссылку на метод.

Visual studio – интегрированная среда разработки для написания, отладки и сборки программного кода.

Microsoft .NET Framework – платформа, которая используется для написания приложений на операционной системе Windows.

**ПРИЛОЖЕНИЕ Б**

**ЭКРАННЫЕ ФОРМЫ ПРОГРАММНЫХ МОДУЛЕЙ**

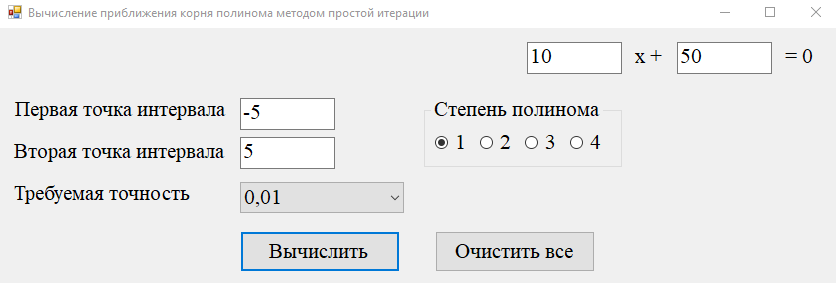


Рисунок Б.1 – Ввод корректных данных в программу

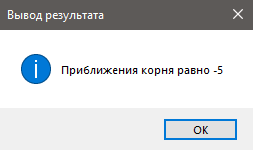


Рисунок Б.2 – Результат вычислений

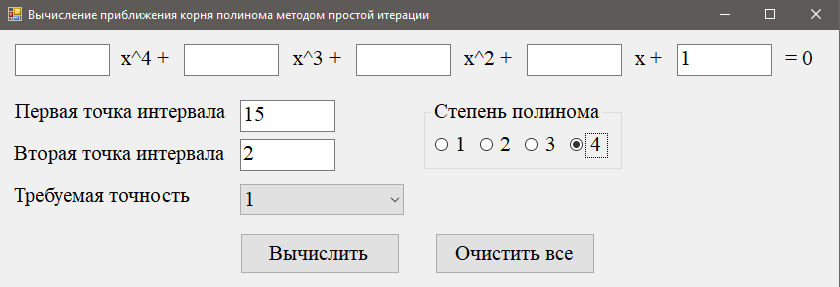


Рисунок Б.3 – Ввод некорректных данных в программу

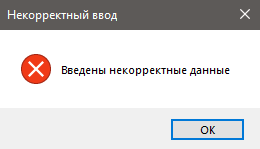


Рисунок Б.4 – Уведомление о некорректном вводе данных

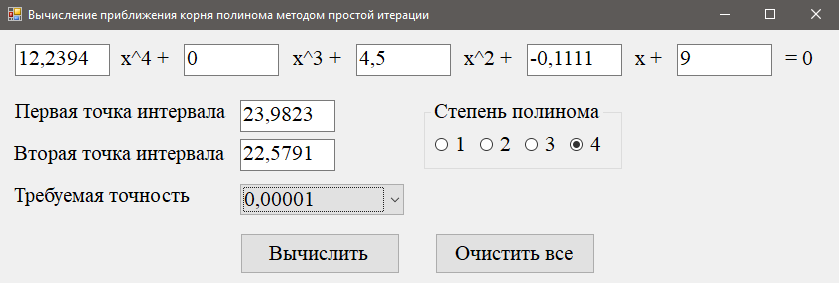


Рисунок Б.5 – Ввод данных

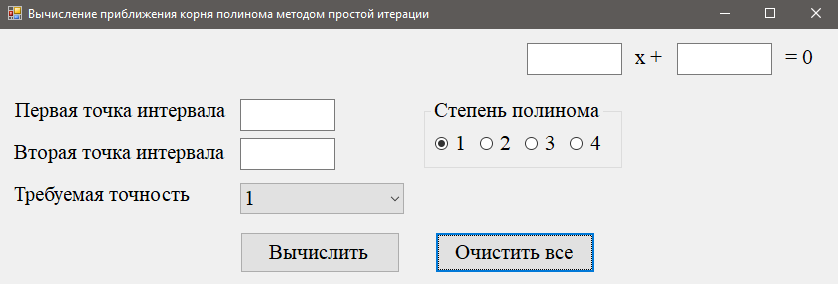


Рисунок Б.6 – Результат нажатия кнопки «Очистить все»

**ПРИЛОЖЕНИЕ В**

**КОД ПРОГРАММЫ**

using System;

using System.Windows.Forms;

namespace Наброски\_2

{

public partial class MainForm : Form

{

// Данные, вводимые пользователем.

private double a, b, requiredAccuracy;

private int degree;

private double[] coefficients;

// Данные, вычисляемые в процессе работы программы.

private double x0, x1, R;

private int coefficientsAmount;

private Function g, gDerivative;

private TextBox[] textboxes;

private Label[] labels;

// Делегат для вызова функций.

private delegate double Function(double x);

// Единственный конструктор формы без аргументов.

public MainForm()

{

InitializeComponent();

// Инициализация значений при запуске формы.

this.Load += (sender, e) =>

{

// Установка первого значения точности в выпадающем списке.

comboBox1.SelectedIndex = 0;

// Вспомогательные массивы, необходимые для дальнейших вычислений.

textboxes = new TextBox[5];

textboxes[0] = FirstCoeff;

textboxes[1] = SecondCoeff;

textboxes[2] = ThirdCoeff;

textboxes[3] = FourthCoeff;

textboxes[4] = FifthCoeff;

labels = new Label[3];

labels[0] = LabelxInFourthDegree;

labels[1] = LabelxInThirdDegree;

labels[2] = LabelxInSecondDegree;

// Установка первой степени полинома.

FirstDegree.Checked = true;

};

}

private void AnyCoeff\_KeyPress(object sender, KeyPressEventArgs e)

{

/\* Данный метод служит обработчиком события нажатия клавиши

в текстовом поле для всех коэффициентов уравнения. \*/

// Если нажатая клавиша - цифра, то она подходит.

if (char.IsDigit(e.KeyChar))

{

return;

}

// Если нажатая клавиша - управляющий символ, то она подходит.

if (char.IsControl(e.KeyChar))

{

return;

}

// Далее необходимо взаимодействие с объектом, вызвавшим событие.

TextBox clickedTextBox = (TextBox)sender;

// Если нажатая клавиша - знак минуса, стоящий на первом месте,

// то он подходит.

if (clickedTextBox.Text.Length == 0)

{

if (e.KeyChar == '-')

{

return;

}

}

// Если нажатая клавиша - запятая(точка), которая стоит не на первом и

// не на последнем месте, то она подходит.

if (clickedTextBox.Text.Length != 0 &&

clickedTextBox.Text.Length != clickedTextBox.MaxLength - 1)

{

// Если нажатая клавиша - точка, заменяем ее на запятую.

if (e.KeyChar == '.')

{

e.KeyChar = ',';

}

if (e.KeyChar == ',')

{

if (clickedTextBox.Text.IndexOf(',') == -1)

{

return;

}

}

}

// Ни одно из условий выше не было верным,

// значит введенный символ не подходит.

e.Handled = true;

}

private void AnyRadioButton\_CheckedChanged(object sender, EventArgs e)

{

/\* Данный метод служит обработчиком события для всех переключателей

группового поля. \*/

// Далее понадобится объект, вызвавший событие.

RadioButton clickedButton = (RadioButton)sender;

// Если переключатель включен, то продолжить обработку события.

if (clickedButton.Checked)

{

switch (clickedButton.Name)

{

case "FirstDegree":

degree = 1;

DisableTextBoxesAndLabels(3);

break;

case "SecondDegree":

degree = 2;

DisableTextBoxesAndLabels(2);

break;

case "ThirdDegree":

degree = 3;

DisableTextBoxesAndLabels(1);

break;

case "FourthDegree":

degree = 4;

DisableTextBoxesAndLabels(0);

break;

default:

throw new Exception("Unknown exception");

}

}

// Выключенный переключатель обрабатывать не нужно.

else

{

return;

}

}

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

{

/\* Данный метод очищает все введенные пользователем данные.

Служит обработчиком для кнопки "Очистить все". \*/

// Установить пустую строку во все текстовые поля.

FirstCoeff.Text = string.Empty;

SecondCoeff.Text = string.Empty;

ThirdCoeff.Text = string.Empty;

FourthCoeff.Text = string.Empty;

FifthCoeff.Text = string.Empty;

FirstIntervalPoint.Text = string.Empty;

SecondIntervalPoint.Text = string.Empty;

// Установка первой степени полинома.

FirstDegree.Checked = true;

// Установка первого значения точности в выпадающем списке.

comboBox1.SelectedIndex = 0;

}

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

{

/\* Данный метод вычисляет приближение корня и выводит его, если пользователь ввел корректные, подходящие данные.

Служит обработчиком для кнопки "Вычислить". \*/

// Если введенные данные корректны, то проверить их на схождение.

if (DataInput())

{

// Если данные пройдут проверку на схождение, то выполняем вычисление.

if (CheckDataForConvergence())

{

// Вычисление приближение корня.

CalculateApproximationToRoot();

// Вывод приближения корня.

ResultOutput("solution exist");

}

else

{

// Иначе уведомить пользователя о невозможности решения данным методом.

ResultOutput("solution doesn't exist");

}

}

else

{

// Если введенные данные некорректны, то уведомить об этом пользователя.

ResultOutput("data was incorrect");

}

}

private bool DataInput()

{

/\* Данный метод предназначен для считывания введенных пользователем данных.

Если пользователь ввел корректные данные, метод возвращает true, иначе - false \*/

// Массив с коэффициентами уравнения. Нулевой элемент - старший коэффициент

// полинома текущей степени.

coefficients = new double[coefficientsAmount];

try

{

// Заполнение массива коэффициентов.

for (int i = coefficientsAmount - 1, j = textboxes.Length - 1; i >= 0; i--, j--)

{

coefficients[i] = double.Parse(textboxes[j].Text);

}

if (coefficients[0] == 0)

{

throw new Exception();

}

// Считывание введенных границ интервала и требуемой точности.

a = double.Parse(FirstIntervalPoint.Text);

b = double.Parse(SecondIntervalPoint.Text);

requiredAccuracy = double.Parse(comboBox1.Text);

// Левая граница не может быть больше или равна правой

// согласно установленному условию.

if (a >= b)

{

return false;

}

// Считывание данных прошло успешно, значит вернуть true.

return true;

}

catch

{

return false;

}

}

private bool CheckDataForConvergence()

{

/\* Данный метод проверяет проверку уравнения на сходимость.

Если уравнение подходит для решения, то возвращает true, иначе - false. \*/

// Создание делегата с помощью вспомогательного метода,

// возвращающего экземлпяр делегата Function.

// Далее g можно рассматривать как функцию g(x).

g = Construct\_g();

// Создание делегата аналогично предыдущей строке.

// gDerivative - производная функции g(x).

gDerivative = Construct\_gDerivative();

// Проверка на сходимость.

if (Math.Abs(gDerivative(a)) < 1)

{

x0 = a;

return true;

}

else if (Math.Abs(gDerivative(b)) < 1)

{

x0 = b;

return true;

}

// Если ни одно из условий выше не было верным, значит невозможно решить.

else

{

return false;

}

}

private double CalculateApproximationToRoot()

{

/\* Данный метод вычисляет приближение корня методом простой итерации.

Возвращает приближение корня требуемой точности. \*/

do

{

x1 = g(x0);

R = Math.Abs(x1 - x0);

x0 = x1;

}

while (R > requiredAccuracy);

return x1;

}

private Function Construct\_g()

{

/\* Данный метод создает и возвращает делегат, использующийся в качестве

функции g(x). \*/

int penultimateCoeffIndex = coefficients.Length - 2;

for (int i = 0; i < coefficients.Length; i++)

{

// Если это не предпоследний коэффициент, стоящий возле x, то "переносим"

// его в другую часть уравнения, изменив его знак и поделив на

// коэффициент.

if (i != penultimateCoeffIndex)

{

coefficients[i] /= -coefficients[penultimateCoeffIndex];

}

}

// Возврат метода, принимающего x и возвращающего значение g(x).

return (x) =>

{

double result = 0;

int n = degree;

for (int i = 0; i < coefficients.Length; i++)

{

if (i != penultimateCoeffIndex)

{

result += Math.Pow(x, n) \* coefficients[i];

}

n--;

}

return result;

};

}

private Function Construct\_gDerivative()

{

/\* Данный метод создает и возвращает делегат,

\* использующийся в качестве производной функции g(x)'. \*/

return (x) =>

{

double result = 0;

int n = degree;

for (int i = 0; i < coefficients.Length - 2; i++)

{

result += coefficients[i] \* n \* Math.Pow(x, n - 1);

n--;

}

return result;

};

}

private void DisableTextBoxesAndLabels(int amount)

{

/\* Данный метод выключает лишние текстовые поля и примыкающие к ним метки.

Выключение происходит слева направо в количестве amount

пар "текстовое поле - метка". \*/

coefficientsAmount = 5 - amount;

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

{

textboxes[i].Text = string.Empty;

labels[i].Visible = false;

textboxes[i].Visible = false;

}

for (int i = amount; i < 3; i++)

{

labels[i].Visible = true;

textboxes[i].Visible = true;

}

}

private void ResultOutput(string condition)

{

/\* Данный метод показывает пользователю окно, уведомляющее о результате. \*/

switch (condition)

{

case "solution exist":

MessageBox.Show(

"Приближения корня равно " + x1,

"Вывод результата",

MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

case "solution doesn't exist":

MessageBox.Show(

"Данное уравнение не может быть решено методом простой итерации",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

break;

case "data was incorrect":

MessageBox.Show("Введены некорректные данные",

"Некорректный ввод",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

break;

default:

return;

}

}

}

}

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Тема: Разработка программных модулей для автоматизации прикладной задачи решения нелинейных уравнений методом простой итерации.

Содержание:

ВВЕДЕНИЕ

1. ТЕОРЕТИЧЕСКИЕ И АНАЛИТИЧЕСКИЕ АСПЕКТЫ АВТОМАТИЗАЦИИ ПРИКЛАДНОЙ ЗАДАЧИ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ПРОСТОЙ ИТЕРАЦИИ

1.1. Описание прикладной задачи решения нелинейных уравнений методом простой итерации

1.2. Анализ предметной области и постановка проекта разработки программных модулей

2. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ ДЛЯ АВТОМАТИЗАЦИИ ПРИКЛАДНОЙ ЗАДАЧИ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ПРОСТОЙ ИТЕРАЦИИ.

2.1. Описание программных модулей для автоматизации прикладной задачи решения нелинейных уравнений методом простой итерации

2.2. Построение схемы взаимосвязи программных модулей и массивов данных для решения прикладной задачи

2.3. Описание алгоритмов работы программных модулей

2.4. Способы и результаты тестирования программных модулей в различных режимах

ЗАКЛЮЧЕНИЕ

Требования к программным модулям:

1. Модуль ввода данных. (Предназначен для ввода уравнения, интервала изоляции и требуемой точности.)

2. Модуль очистки данных. (Очищает все введенные ранее пользователем данные.)

3. Модуль проверки корректности данных. (Проверяет введенные пользователем данные на корректность.)

4. Модуль вычислений. (Выполняет необходимые вычисления методом простой итерации.)

5. Модуль вывода результатов. (Выводит результаты вычислений в понятной человеку форме.)

6. Модуль проверки сходимости данных. (Проверяет пригодность данных для нахождения результата данным методом.)

7. Модуль отключения текстовых полей и меток. (Отключает поля и метки, которые не используются в процессе ввода данных.)

8. Модуль создания функции. (Создает функцию, необходимую для вычислений результата.)

9. Модуль создания производной функции. (Создает производную функции для проверки на сходимость данных.)

10. Модуль обработки события переключения переключателей (Устанавливает выбранный вид вводимых данных пользователем.)

11. Модуль обработки нажатия кнопки вычисления (Служит посредником между остальными модулями.)

Студент Махницкий Д.С.

Преподаватель Харламов П.С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 года