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: **«РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ В РАСПРЕДЕЛЁННЫХ СИСТЕМАХ ОБРАБОТКИ ИНФОРМАЦИИ»**

Исполнитель: студент гр. ИП-31

Д.С. Соловьев

Руководитель: преподаватель Н.В. Самовендюк

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2018

СОДЕРЖАНИЕ

[Введение 3](#_Toc469279035)

[1 Обзор существующих методов решения задачи 5](#_Toc469279037)

[1.1 Метод Гаусса 5](#_Toc469279038)

[1.2 Метод LU-разложения 5](#_Toc469279039)

[2 Параллельное программирование 7](#_Toc469279041)

[2.1 Общая информация 7](#_Toc469279042)

[2.2 Определение процесса 9](#_Toc469279043)

[2.3 Определение потока 10](#_Toc469279043)

[2.4 Определение ресурса 10](#_Toc469279043)

[2.5 Обзор существующих методов распараллеливания процессов 5](#_Toc469279039)

[2.5.1 TPL (Task Parallel Library) 11](#_Toc469279047)

[2.5.2 Класс Parallel и его методы 12](#_Toc469279048)

[3 Алгоритмический анализ задачи 22](#_Toc469279058)

[3.1 Постановка задачи 22](#_Toc469279059)

[3.2 Анализ исходных данных 22](#_Toc469279059)

[3.3 Алгоритм обработки исходных данных 22](#_Toc469279060)

[3.4 Алгоритм решения СЛАУ с помощью метода LU-разложения 23](#_Toc469279062)

[3.5 Алгоритм решения СЛАУ с помощью метода Гаусса 23](#_Toc469279063)

[4 Разработка программного кода и верификация 25](#_Toc469279064)

[5 Описание результатов и выводы 29](#_Toc469279066)

[Заключение 31](#_Toc469279067)

[Список использованных источников 32](#_Toc469279068)

[Приложение А – Исходный код программы 33](#_Toc469279069)

[Приложение В – Диаграмма классов 48](#_Toc469279070)

ВВЕДЕНИЕ

Система линейных алгебраических уравнений – система уравнений, каждое уравнение в которой является линейным (т.е. алгебраическим уравнением первой степени). Решение систем линейных алгебраических уравнений является одной из классических задач линейной алгебры, во многом определившая её объекты и методы. Кроме того, линейные алгебраические уравнения и методы их решения играют важную роль во многих прикладных направлениях, в том числе в линейном программировании, эконометрике.

Системы линейных уравнений малых порядков могут быть решены вручную при знании необходимых алгоритмов, однако нахождение решений СЛАУ больших порядков является довольно затруднительной и трудоёмкой задачей для человека. Это создаёт востребованность в инструменте, автоматизирующем данные вычисления. Более того, для выполнения компьютерного моделирования и некоторых алгоритмов требуется решать системы уравнений, что делает выполнение данной задачи человеком нецелесообразным. Электронно-вычислительные машины справляются с этой задачей за значительно короткое время, а также могут работать с большим количеством входных данных.

С течением времени увеличивается количество применяемых в расчётах данных, что позволяет, например, уточнить выходные данные, полученные с помощью компьютерного моделирования. Но с ростом количества данных растёт и время, затрачиваемое на выполнение расчётов. Однако технический прогресс не стоит на месте, что позволяет использовать аппаратные (и в следствии программные) средства с большей эффективностью. Одним из способов увеличения производительности вычислительной системы является распараллеливание процессов, что позволяет при правильной реализации алгоритма производить вычисление данных за относительно короткое время.

Целями данного курсового проекта являются разработка приложения, позволяющего решать системы линейных алгебраических уравнений распределённым методом LU-разложения, сравнение его эффективности относительно его линейного варианта реализации и распределённого варианта метода Гаусса, изучение алгоритмов этих методов и изучение методов распараллеливания процессов.

Задачи данного курсового проекта следующие:

* Изучить теоретический материал о заданных методах решения систем линейных алгебраических уравнений;
* Изучить принцип устройства и способы работы с программными средствами методов распараллеливания процессов;
* Выполнить алгоритмический анализ поставленной задачи;
* Разработать программный код, ориентируясь на информации, полученной и проанализированной на предыдущих этапах;
* Провести сравнительный анализ эффективности реализаций, полученных в результате разработки заданных методов.

1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ

Рассмотрим систему из линейных алгебраических уравнений

(1)

В матричном виде система может быть представлена как

(2)

Где:

, (3)

Решениями системы являются значения вектора , при подстановке которых в систему все уравнения обращаются в тождество.

1.1 Метод Гаусса

Метод Гаусса – классический метод решения системы линейных алгебраических уравнений. Входные данные для решения СЛАУ этим методом должны соответствовать следующим условиям: матрица коэффициентов системы является квадратной и её определитель отличен от нуля (что значит, что матрица невырожденная).

Согласно [1], вычислительная схема метода Гаусса состоит из двух этапов. Первый этап заключается в приведении системы к трапециевидному виду. Этот этап называется прямым ходом. Второй этап – определение неизвестных – называется обратным ходом.

Прямой ход метода Гаусса состоит в последовательном исключении коэффициентов при неизвестных начиная с первого столбца.

Прямой ход реализуется по следующим формулам (индекс k в квадратных скобках означает номер цикла (номер столбца)):

Умножение -й строки на число:

, , (2)

Вычитание -й строки из -й строки:

, (3)

, (3)

Обратный ход – вычисление неизвестных – реализуется по следующим формулам, начиная с последнего уравнения системы:

, (3)

, , (3)

2.2 Метод LU-разложения

Согласно [1], можно показать, что применение метода Гаусса эквивалентно LU-разложению матрицы , т.е. ее представлению в виде произведения , где:

, (4)

, (4)

Вычисление коэффициентов в данных матрицах производится по следующим формулам:

, (4)

, (4)

, , (4)

, (4)

, (4)

, (4)

, , (4)

Такое разложение дает следующий результат:

*,*  (4)

Положим:

(4)

Тогда имеем:

(4)

В итоге имеем следующую систему уравнений:

(1)

Из этой системы находим последовательно :

, , (4)

Зная , можно найти из системы:

(1)

Откуда можно выразить как:

, (4)

3 Параллельное программирование

## **3.1 Общая информация**

Ранее параллельное программирование было уделом узкого круга людей, которых интересовали задачи для огромных суперкомпьютеров. В настоящее время многоядерные процессоры стали нормой для рядового пользователя. Приложения для широкого круга пользователей научились работать в многопоточном режиме, что увеличило скорость их работы.

Как говорит источник [3], использование параллельного программирования становится наиболее необходимым, поскольку позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счет размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно - оптимизировать программы под многоядерные системы.

Ключевым понятием при работе с многопоточностью является поток. Поток представляет некоторую часть кода программы. При выполнении программы каждому потоку выделяется определенный квант времени. При помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Без многопоточности в случае графического приложения, которое, к примеру, посылает запрос к серверу или считывает и обрабатывает огромный файл, графический интерфейс не будет отзывчив на время выполнения задачи. Благодаря потокам имеется возможность выделить такую задачу в отдельный поток, что позволит графическому интерфейсу приложения сохранить отзывчивость.

3.5 Обзор существующих методов распараллеливания процессов

3.5.1 TPL (Task Parallel Library)

В прошлом распараллеливание алгоритмов требовало управления потоками и взаимоблокировками на низком уровне. Visual Studio 2010 и .NET Framework 4 улучшают поддержку параллельного программирования, путем предоставления новой среды выполнения, новых типов библиотек класса (TPL) и новых средств диагностики (Concurrency Visualizer). Как оговаривается в источнике [5], эти возможности упрощают параллельную разработку, что позволяет разработчикам писать эффективный, детализированный и масштабируемый параллельный код с помощью естественных выразительных средств без необходимости непосредственной работы с потоками или пулом потоков. Ниже (Рис. 1) представлен общий обзор архитектуры параллельного программирования в .NET Framework 4.



Рисунок 1 – Архитектура параллельного программирования в .NET Framework 4

Одним из самых главных среди нововведений, внедренных в среду .NET Framework 4.0, является библиотека распараллеливания задач (TPL). Эта библиотека упрощает создание и применение многих потоков, а также позволяет автоматически использовать несколько процессоров, что совершенствует многопоточное программирование.

Проще говоря, TPL предоставляет возможности для автоматического масштабирования приложений с целью эффективного использования ряда доступных процессоров.

Библиотека TPL определена в пространстве имен *System.Threading.Tasks*, но для работы с ней обычно требуется также включать в программу класс *System.Threading*, поскольку он поддерживает синхронизацию и другие средства многопоточной обработки.

3.5.2 Класс Parallel и его методы

Одним из главных классов в TPL является *System.Threading.Tasks.Parallel*. Этот класс поддерживает набор методов, которые позволяют выполнять итерации по коллекции данных в параллельном режиме.

У данного класса есть несколько методов. К примеру, *Parallel.For()* и *Parallel.ForEach()*, для каждого из которых определены многочисленные перегруженные версии. Эти методы позволяют создавать тело операторов кода, которое может выполнятся в параллельном режиме. Как сказано в [6], эти операторы представляют собой логику того же рода, которая была бы написана в нормальной циклической конструкции (с использованием ключевых слов C# *for* и *foreach*). Однако их преимущество состоит в том, что класс *Parallel* самостоятельно берет потоки из пула потоков и управляет конкуренцией.

Также данный класс располагает методом *Invoke*, который позволяет выполнять один или несколько методов, указываемых в виде его аргументов. Он также масштабирует исполнение кода, используя доступные процессоры, если имеется такая возможность.

5 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ

5.1 Постановка задачи

Постановка задачи звучит следующим образом:

Необходимо разработать приложение, осуществляющее распределенное решение системы линейных алгебраических уравнений на основе метода LU-разложения, где – нижнетреугольная матрица, – верхнетреугольная матрица с единицами на главной диагонали. Также необходимо предусмотреть возможность ввода исходных данных из текстового файла. Требуется сравнить время выполнения реализованного метода по скорости с его линейным вариантом и с распределенным вариантом метода Гаусса.

Исходными данными являются данные методы решения СЛАУ:

* линейный метод LU-разложения;
* распределённый метод LU-разложения;
* распределенный метод Гаусса.

5.1 Анализ исходных данных

Исходные данные должны удовлетворять следующим условиям:

* матрица квадратная;
* количество строк матриц и одинаково;
* матрица должна быть невырожденной.

Для выполнения поставленной задачи программа должна реализовать следующий функционал:

* ввод исходных данных из файла с форматом .txt;
* ввод исходных данных вручную с возможностью сохранения введённых данных в формате .txt;
* ввод исходных данных с помощью генератора случайной матрицы коэффициентов;
* возможность просмотра и очистки введённых данных;
* вычисление системы распределенным и линейным методами LU-разложения и распределенным методом Гаусса;
* проверка результатов вычисления;
* вывод результатов вычисления, значений потребления ресурсов и времени выполнения.

5.2 Алгоритм обработки исходных данных

В начале работы программы требуется ввести данные в программу с помощью одного из трёх способов ввода: с помощью ручного ввода, с помощью ввода из текстового файла или с помощью генератора случайной матрицы коэффициентов. После удачного ввода данных становятся доступными функции очистки и просмотра введённых данных. Перед тем, как начать процедуру вычисления СЛАУ требуется также выбрать методы, с помощью которых будет производиться расчёт решения. В случае выбора одного или более методов функция вычисления СЛУ становится доступной.

После активации вышеупомянутой функции начинается расчёт решений с помощью каждого выбранного метода, при этом запускается мониторинг производительности, выводящий показатели потребления ресурсов (процент процессорного времени и количество свободной памяти в мегабайтах) на соответствующие графики, и таймер, определяющий время, затраченное на вычисление решений всеми методами. Производится инициализация таблицы, содержащей информацию о запускаемых методах. Во время каждой процедуры расчёта каждым методом в начале, при ошибке или в конце вычислений производится добавление записи о состоянии метода в журнал, содержащей время на рабочей машине, относительное время выполнения метода, название метода и его состояние. В то же время производится запись состояния метода и относительного времени его выполнения в таблицу, а также становятся доступными кнопки показа решений и их сохранения в текстовый файл. После окончания расчётов таймер и счётчик производительности останавливают работу. На протяжении всего времени расчётов пользователь также имеет доступ к функциям показа исходной матрицы и показа всех решений.

5.3 Алгоритм решения СЛАУ с помощью метода LU-разложения

Данный алгоритм можно разделить на 3 составляющие части:

* нахождение матриц L и U;
* нахождение вектора Y;
* нахождение вектора X.

Блок-схемы алгоритмов изображены на рисунках /////

5.4 Алгоритм решения СЛАУ с помощью метода Гаусса

Данный алгоритм можно разделить на 2 составляющие части:

* выполнение прямого хода – приведение системы к трапециевидному виду;
* выполнение обратного хода – нахождение вектора X.

Блок-схемы алгоритмов изображены на рисунках /////

6 РАЗРАБОТКА ПРОГРАММНОГО КОДА

В течении разработки данного программного продукта была создана следующая структура проекта (Рис.2).

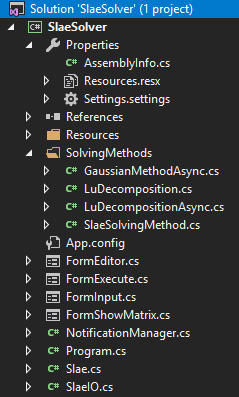


Рисунок 2 – Структура проекта разработанного приложения

Классы проекта, важные для задачи, поставленной в данном курсовом проекте, описаны в таблице 1. Исходный код данных классов представлен в приложении А данной пояснительной записки.

Таблица 1 – Классы проекта

|  |  |
| --- | --- |
| Название класса | Описание |
| Program.cs | Точка входа в приложение, создание и запуск главного окна |
| FormInput.cs | Класс формы для инициализации данных, выбора методов для вычисления решений и запуска процедуры вычисления |
| FormEditor.cs | Класс формы для ввода значений вручную с функцией сохранения данных в файл |
| FormShowMatrix.cs | Класс формы для показа записей СЛАУ либо полученных решений |

Продолжение таблицы 1

|  |  |
| --- | --- |
| FormExecute.cs | Класс формы, запускающей поочерёдно вычисление решений СЛАУ с помощью выбранных методов.  Также позволяет отслеживать затраты ресурсов и времени на вычисления текущим методом |
| Slae.cs | Класс, представляющий собой объект СЛАУ, хранящий матрицу коэффициентов, вектор свободных членов и размерность матрицы |
| SlaeIO.cs | Класс, производящий операции ввода-вывода с помощью объектов потока Stream в качестве входного параметра, получаемого из диалога форм |
| ISlaeSolvingMethod.cs | Интерфейс, обеспечивающий интерфейс взаимодействия такой, что все объекты, реализующие данный интерфейс, могут решать СЛАУ и возвращать решение |
| LuDecomposition.cs | Класс, производящий вычисление решений СЛАУ синхронным методом LU-разложения |
| LuDecompositionAsync.cs | Класс, производящий вычисление решений СЛАУ распределённым методом LU-разложения |
| GaussianMethodAsync.cs | Класс, производящий вычисление решений СЛАУ распределённым методом Гаусса |
| NotificationManager.cs | Класс, позволяющий выводить сообщения об ошибках, предупреждениях и сообщениях информационного характера |

Каждый указанный класс включает в себя методы, необходимые для решения поставленной задачи. В таблицах 2-11 будут перечислены методы соответствующего класса и их описание. Класс Program.cs не будет представлен в подобной таблице ввиду того, что имеет один метод Main, который является точкой входа в приложение.

Класс FormInput является классом первой формы приложения. В процессе её работы может быть инициализирована СЛАУ одним из трёх методов инициализации: случайно, с помощью файла или вручную. Там же могут быть выбраны способы расчёта СЛАУ и запущены функции просмотра данных матрицы, очистки данных СЛАУ и запуска расчётов решения СЛАУ. Методы данного класса представлены в таблице 2.

Таблица 2 – Методы класса FormInput.cs

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| SlaeChanging  Listener | private void () | Используется в свойстве *Slae* в методе *set*, что означает, что при |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
|  |  | изменении значения свойства вызывается данный метод. Метод устанавливает активность кнопок «Показать матрицу», «Очистить», «Запустить вычисления», а также состояние элемента «чек-бокс» в зависимости от наличия значения в переменной данного свойства |
| FormInput | public () | Конструктор, инициализирует окно |
| BtnRandomInput  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки ввода данных матрицы «Случайно». Блокирует данную кнопку и начинает генерацию СЛАУ с помощью соответствующего метода класса *Slae*. |
| BtnFileInput  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки ввода данных матрицы «Файл». Показывает стандартное диалоговое окно выбора файла и при успешном выборе файла начинает считывание данных из него с помощью соответствующего метода класса *SlaeIO*. Возвращаемым значением инициализируется переменная свойства *Slae*. |
| BtnManualInput  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки ввода данных матрицы «Вручную». Инициализирует форму *FormEditor*, в которой может быть произведена инициализация СЛАУ |
| BtnShowMatrix  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Показать матрицу». Инициализирует форму *FormShowMatrix*, в которой будет показано значение переменной свойства *Slae* |
| BtnClear  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Очистить». Очищает значение переменной свойства *Slae* |
| BtnExecute  \_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Запустить вычисления». Определяет выбранные методы решения СЛАУ, затем передаёт их и СЛАУ при инициализации в форму *FormExecute* |

Класс FormEditor является классом формы редактора, с помощью которого можно произвести ручной ввод СЛАУ различного порядка в программу и, при необходимости, сохранить её в файл. Методы данного класса представлены в таблице 3.

Таблица 3 – Методы класса FormEditor.cs

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| FormEditor | public (SlaeSetter ss) | Конструктор. Принимает единственным параметром делегат, изменяющий данные СЛАУ, позволяющий передавать данные между формами. |
| CreateDgvTable | (1) private void (),  (2) public static void (DataGridView dgv, int N) | Перегруженный метод.  В случае 1: не имеет входных параметров, инициализирует пустую таблицу.  В случае 2: принимает аргументами объект DataGridView и порядок матрицы N. Создаёт в объекте DataGridView пустую таблицу соответствующего размера, готовую для заполнения данными. |
| NumudN\_  ValueChanged | private void (object sender, EventArgs e) | Обработчик события изменения значения элемента формы «NumericalUpDown», хранящего порядок матрицы. Считывает значение порядка матрицы N из данного элемента и запускает метод CreateDgvTable с данным значением. |
| DgvData\_  CellValueChanged | private void (object sender, DataGridView  CellEventArgs e) | Обработчик события изменения значения ячейки элемента формы «DataGridView». Устанавливает введённое значение на соответствующее место в матрице. |
| BtnSave\_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Сохранить». Сохраняет данные в файл, выбранный в стандартном диалоговом окне сохранения файла. |
| BtnApply\_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Применить». Передаёт данные в родительскую форму и закрывает текущую форму. |

Класс FormShowMatrix является классом формы, с помощью которой можно просмотреть введённую в программу СЛАУ, а также решения СЛАУ. Имеет ограничение на вывод: максимальный порядок выводимой матрицы СЛАУ равен 650. Однако на отображение решений уравнения это не влияет, так как в случае вывода решений одномерный массив больше 650 элементов преобразовывается в двумерный массив, где каждая строка имеет размер в 650 элементов. Методы данного класса представлены в таблице 4.

Таблица 4 – Методы класса FormShowMatrix.cs

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| FormShowMatrix | (1) public (Slae slae)  (2) public (double[][] data) | Конструктор. Перегружен.  Передаёт соответствующие данные в метод FillDgv данного класса. |
| FillDgv | (1) public static void (DataGridView dgvData, Slae slae)  (2) public static void (DataGridView dgvData, Slae double[][] data) | Перегруженный метод. Помещает соответствующие данные в таблицу элемента DataGridView формы. |
| CreateDgvTable | public static void (DataGridView dgvData, int N, int M) | Принимает объект DataGridView и размеры матрицы N и M. Создаёт в объекте DataGridView таблицу соответствующего размера |
| ModifyArray | public static double[][] (double[] data, int xCount) | Принимает в качестве параметров массив данных и максимальный размер таблицы. Преобразует массив данных размером больше максимального размера в двумерный массив из массивов с максимальным размером. |

Класс FormExecute является классом формы, в которой поочерёдно происходит расчёт решений введённой ранее СЛАУ выбранными ранее методами. Интерфейс данной формы показывает названия методов, временные затраты на их выполнение, позволяет просмотреть полученные решения и сохранить их в файл. Более того, также имеются функции просмотра исходной матрицы и просмотра всех полученных решений. Также производится запись состояния вычислений в лог, где отображаются абсолютное и относительное время, название метода и его состояние. В правой части интерфейса имеются значения затраченного на вычисления время, показатели ресурсов, таких как процент процессорного времени и количество свободной оперативной памяти, и соответствующие графики. Данные инструменты позволят отслеживать затраты ресурсов и таким образом оценивать их затраты при процедуре вычисления решений. Форма также имеет т.н. «лёгкий» режим работы, где отключается правая часть интерфейса, что позволяет не затрачивать ресурсы компьютера на отслеживание затрат ресурсов. Методы класса представлены в таблице 5.

Таблица 5 – Методы класса FormExecute.cs

|  |  |  |  |
| --- | --- | --- | --- |
| Название метода | Сигнатура | | Описание |
| FormExecute | public (Slae slae, List <ISlaeSolvingMethod> solvingMethods, bool lightMode = false) | | Конструктор. Принимает в качестве аргументов объект СЛАУ, методы решения СЛАУ и необязательный логический параметр, определяющий режим работы (стандартный или т.н. «лёгкий»). Инициализирует форму и соответствующие данные формы. |
| Configure  Perfomance  Counters | private void () | | Настраивает часть интерфейса, ответственную за отслеживание потребления ресурсов. |
| CleanupWnd | private void () | | Удаляет объекты части интерфейса, ответственной за отслеживание потребления ресурсов, и освобождает память, запуская процесс сборки мусора (т.н. Garbage Collection). |
| Deactivate  Perfomance  Counter | private void () | | Останавливает процесс отслеживания потребления ресурсов. |
| TimerPerfomance\_  Tick | private void (object sender, EventArgs e) | | Обработчик события срабатывания таймера, обновляет элементы формы, предназначенные для отслеживания потребления ресурсов. |
| Update  PerfomanceCharts | private void (float cpuValue, float memValue) | | Обновляет графики потребления ресурсов. |
| AddLog | (1) private void (RunningStates state)  (2) private void (string value) | | Перегруженный метод. Добавляет запись журнала в соответствующий элемент формы. |
| BtnShowMatrix\_  Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Показать матрицу». Запускает форму FormShowMatrix с текущей СЛАУ. | |

Продолжение таблицы 5

|  |  |  |
| --- | --- | --- |
| DgvResults\_  CellContentClick | private void (object sender, DataGridView  EventArgs e) | Обработчик события нажатия на содержимое ячейки объекта DataGridView. При нажатии на кнопки «Показать корни» и «Сохранить» при условии существования решения выполняет соответствующие действия. |
| BtnShowAll  Results\_Click | private void (object sender, EventArgs e) | Обработчик события нажатия кнопки «Показать все результаты». Запускает форму FormShowMatrix с матрицей результатов. |
| FormExecute  \_Load | private async void (object sender, EventArgs e) | Обработчик события запуска формы. Поочерёдно запускает вычисление решений СЛАУ заданными методами. Одновременно ведётся запись состояний в журнал и объект DataGridView. Асинхронен, что позволяет работать с формой во время вычисления решений. |
| ExecMethod  Async | private static async Task<double[]>  (ISlaeSolvingMethod method, Slae slae) | Запускает расчёт решений СЛАУ вычислительным методом. В качестве аргументов выступают объект вычислительного метода и объект СЛАУ |

Класс Slae представляет собой интерпретацию объекта «система линейных алгебраических уравнений». Хранит матрицу коэффициентов перед неизвестными, вектор свободных членов и размерность матрицы. Методы данного класса представлены в таблице 6.

Таблица 6 – Методы класса Slae.cs

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| Slae | (1) public (double[][] matrix, double[] b)  (2) public (int N) | Перегруженный конструктор.  В случае 1: принимает в качестве параметров матрицу коэффициентов и вектор свободных членов. |

Продолжение таблицы 6

|  |  |  |
| --- | --- | --- |
|  |  | Инициализирует объект этими данными.  В случае 2: принимает в качестве параметра значение порядка матрицы. Генерирует случайную матрицу такого порядка. |
| GetSLAEString | (1) public string ()  (2) public static string (double[][] values, double[] b) | Перегруженный метод. Генерирует строку, содержащую представление СЛАУ в формате матрицы соответствующего порядка с добавленным справа вектором свободных коэффициентов. В первом случае генерирует строку на основе текущего объекта, во втором – на основе входных данных. |

Статический класс SlaeIO служит для файлового ввода-вывода объектов СЛАУ. Методы данного класса представлены в таблице 6.

Таблица 6 – Методы класса SlaeIO.cs

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| Read | public static (Stream stream) | Метод для чтения из файла, объект файлового потока которого передаётся в качестве аргумента, объекта СЛАУ, представленного в определённом строковом формате |
| ParseSlaeString | private static Slae (string slaeString) | Производит считывание объекта СЛАУ |

Разработанная программа имеет вид, представленный на рисунке 6.1.

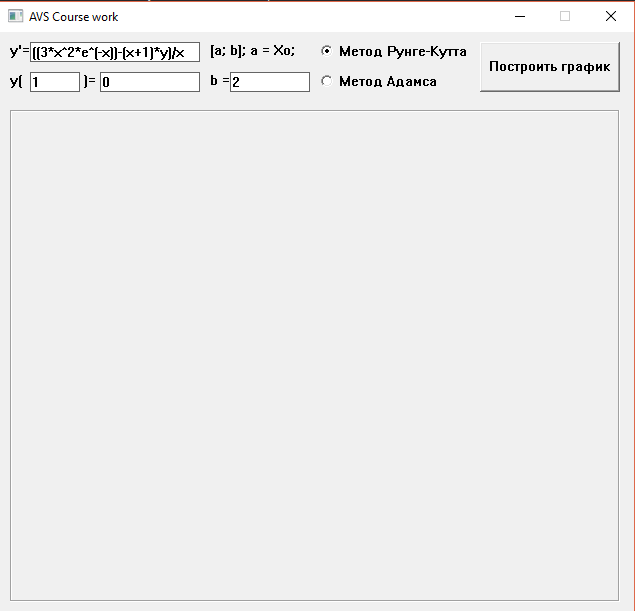


Рисунок 6.1 – Главное окно программы

7 РЕАЛИЗАЦИЯ ЗАДАЧИ В МАТЕМАТИЧЕСКОМ ПАКЕТЕ

Решение ОДУ в математическом пакете необходимо для того, чтобы удостовериться в правильности решения и построения графика в разработанной программе.

Для реализации задачи будет использоваться бесплатный математический пакет SciLab.

Чтобы решить обыкновенное дифференциальное уравнение, в первую очередь определяем функцию вида . Затем создаем переменные и , в которых будут храниться начальные условия. Также создаем переменную и указываем в ней начальное и конечное значения , а также шаг. После всех подготовительных действий необходимо вызвать функцию , в результате выполнения которой в переменной будет храниться массив значений функции. Стоит заметить, что у функции для вычисления дифференциального уравнения есть необязательные параметры, например, метод решения или тип решаемой задачи, относительная и абсолютная погрешности вычислений и т.д. Для последующего сравнения результатов решения дифференциального уравнения в математическом пакете с результатами решения в разработанной программе построим две функции: одна решена методом Рунге-Кутта четвертого порядка точности, а другая – методом Адамса. То есть необходимо вызвать функции и . Конечным результатом должно быть построение двух решений дифференциального уравнения. Чтобы построить график по точкам нужно вызвать функцию . Чтобы на одной координатной оси построить обе функции, также нужно указать значения и второй функции. Для оформления функции на графике необходимо использовать ещё один параметр, строку, в котором указывается цвет, стиль линии и точек. То есть в результате необходимо вызвать функцию . Также на графике, для удобства анализа результатов, построим сетку с помощью вызова функции Чтобы не запоминать какого цвета определенная функция, с помощью вызова функции добавим на график легенду. Полный текст .sci документа представлен в приложении Б.

8 ОПИСАНИЕ РЕЗУЛЬТАТОВ И ВЫВОДЫ

В ходе выполнения курсового проекта была разработана программа для решения обыкновенного дифференциального уравнения первого порядка методами Рунге-Кутта 4 порядка точности и Адамса с использованием вычислительных возможностей сопроцессора. Был использован математический пакет SciLab для проверки правильности полученных решений с помощью данных численных методов. Заданное дифференциальное уравнение было решено правильно. Результаты работы программы изображены на рисунках 8.1 и 8.2, результат проверки правильности вычислений в математическом пакете изображён на рисунке 8.3.

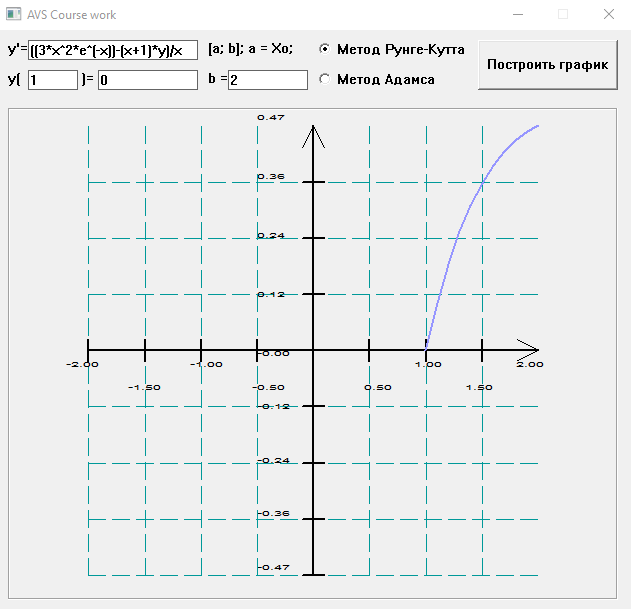


Рисунок 8.1 – Результат вычислений методом Рунге-Кутта четвёртого порядка точности

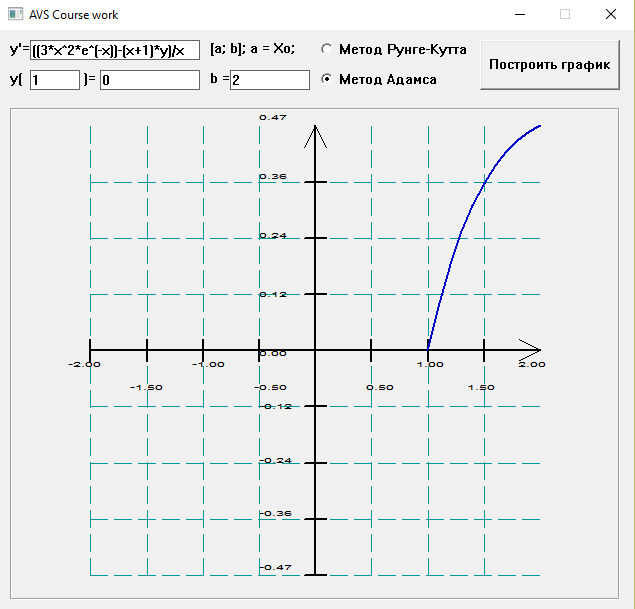


Рисунок 8.2 – Результат вычислений методом Адамса

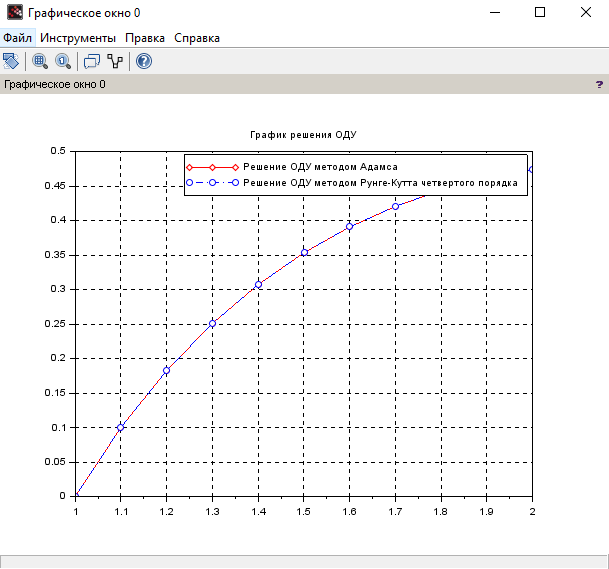


Рисунок 8.3 – График решения заданного ОДУ в SciLab

Судя по представленным выше рисункам можно судить, что исходные данные в разработанной программе обрабатываются правильно.

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было разработано Windows-приложение, предназначенное для решения обыкновенного дифференциального уравнения первого порядка. Были проведены такие этапы разработки, как изучение теоретического материала, алгоритмический анализ задачи и составление алгоритма работы приложения.

В течение выполнения данного курсового проекта был изучен теоретический материал о численных методах Рунге-Кутта четвертого порядка точности и Адамса, на основе которого в дальнейшем были разработаны алгоритмы решения поставленной задачи.

Затем был изучен принцип устройства GDI и математического сопроцессора, а также методы работы с ними. Данные знания пригодились при построении графика в окне программы и выполнении расчетов в вышеуказанных алгоритмах численных методов.

После получения вышеуказанных сведений был разработан указанный программный продукт. При его использовании конечный пользователь может задавать вид решаемого дифференциального уравнения, его начальные условия и используемый для расчета численный метод. В итоге был приобретен опыт разработки подобных приложений с помощью языка программирования C++ и набора функций WinAPI. Приложение было разработано в среде разработки Microsoft Visual Studio Community 2017.

Для проверки правильности полученного решения в разработанной программе был использован бесплатный математический пакет SciLab. Проверка осуществлялась на основе заданного дифференциального уравнения и была пройдена успешно, что свидетельствует о корректной работе программы.

Поставленные задачи в курсовом проекте были решены полностью.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Холл, Дж. Современные численные методы решения обыкновенных дифференциальных уравнений / Дж. Холл, Дж. Уамл. —  М.: Мир, 1979. – 312 с.
  2. Мышенков, В.И. Численные методы. Ч. 2. Численное решение обыкновенных дифференциальных уравнений: Учебное пособие для студентов специальности 073000 / В.И. Мышенков, Е.В. Мышенков. – М.: МГУЛ, 2005. – 109 с.: ил.
  3. Фролов, Г.В. Библиотека системного программиста. Том 14. Графический интерфейс GDI в Microsoft Windows / Г. В. Фролов. – М.: Диалог-МИФИ, 1993. – 288 с.
  4. Григорьев, В.Л. Архитектура и программирование арифметического сопроцессора / В.Л. Григорьев. – М.: Энергоатомиздат, 1991. – 209 с.
  5. Юров, В.И. Assembler: специальный справочник / В.И. Юров. – Спб: Питер, 2001. – 624 с.
  6. Страуступ, Б. Язык программирования С++. Специальное издание / Б. Страуступ. — М.: Бином, 2015. – 1136 c.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

**Файл CourseWork.cpp**

**Файл DiffEquationSolver.h**

**Файл ReversedPolishNotation.h**

ПРИЛОЖЕНИЕ Б

(обязательное)

**Документ СКМ**

ПРИЛОЖЕНИЕ В

**Диаграмма классов**