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 2](#_Toc469279035)

[1 Обзор существующих методов решения задачи 4](#_Toc469279037)

[1.1 Метод LU-разложения 4](#_Toc469279038)

[1.2 Метод Гаусса 6](#_Toc469279039)

[1.3 Параллельное программирование 7](#_Toc469279039)

[1.4 Обзор существующих методов распараллеливания процессов 8](#_Toc469279039)

[1.4.1 TPL (Task Parallel Library) 9](#_Toc469279047)

[1.4.2 Класс Parallel и его методы 10](#_Toc469279048)

[1.4.3 Класс PLINQ и его методы 11](#_Toc469279048)

[2 Алгоритмический анализ задачи 1](#_Toc469279058)2

[2.1 Постановка задачи 12](#_Toc469279059)

[2.2 Анализ исходных данных 12](#_Toc469279059)

[2.3 Алгоритм обработки исходных данных 12](#_Toc469279060)

[2.4 Алгоритм решения СЛАУ с помощью метода LU-разложения 13](#_Toc469279062)

[2.5 Алгоритм решения СЛАУ с помощью метода Гаусса 17](#_Toc469279063)

[2.6 Обоснование выбранного инструментария 20](#_Toc469279063)

[3 Разработка программного кода 21](#_Toc469279064)

[3.1 Описание структуры проекта и его компонентов 21](#_Toc469279066)

[3.2 Пользовательский интерфейс 31](#_Toc469279066)

[3.3 Анализ результатов и выводы 35](#_Toc469279066)

[Заключение 38](#_Toc469279067)

[Список использованных источников 39](#_Toc469279068)

[Приложение А – Исходный код программы 40](#_Toc469279069)

Приложение Б – Блок-схемы алгоритмов 58

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 [1]. 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:

Где:

. (2)

Решениями системы являются значения вектора , при подстановке которых в систему все уравнения обращаются в тождество. [2]

1.1 Метод LU-разложения

Согласно [3], матрицу возможно представить в виде произведения , где:

,

(3)

.

Вычисление коэффициентов в данных матрицах производится по следующим формулам:

,

,

, ,

, (4)

,

,

, ,

Такое разложение дает следующий результат:

*,*  (5)

Положим:

Тогда имеем:

В итоге имеем следующую систему уравнений:

(6)

Из этой системы находим последовательно :

, , (7)

Зная , можно найти из системы:

(8)

Откуда можно выразить как:

, (9)

1.2 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 [4], 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:

, , (10)

Вычитание -й строки из -й строки:

,

, (11)

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

,

, , (12)

## **1.3 Параллельное программирование**

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 [5], 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. Как сказано в источнике [6], 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.

*Многопоточность* – это специализированная форма многозадачности. В основном, выделяют два типа многозадачности: основанную на процессах и основанную на потоках. Источник [7] говорит, что отличия многозадачности на основе процессов и потоков сводится к следующему: многозадачность на основе процессов организуется для параллельного выполнения программ, а многозадачность на основе потоков - для параллельного выполнения отдельных частей одной программы.

*Процесс* – это по сути запущенная программа. Следовательно, основанная на процессах многозадачность – средство, позволяющее компьютеру выполнять несколько операций (программ) одновременно. Например, основанная на процессах многозадачность предоставляет одновременно редактировать текст в текстовом редакторе и работать с другой запущенной программой.

1.4 Обзор существующих методов распараллеливания процессов

1.4.1 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 [8], 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.

В качестве альтернативы также будет рассмотрен фреймворк PLINQ.

1.4.2 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. 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 [9]. Участок кода программы, использующего метод *Parallel.For()*, изображён на рисунке 2.

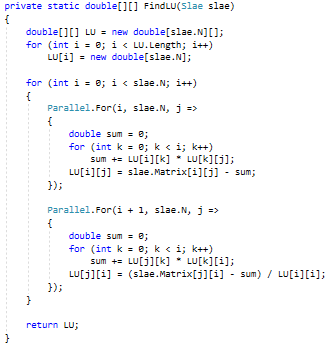


Рисунок 2 – Пример использования метода For класса Parallel

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. Данный метод сначала инициирует выполнение, а затем ожидает завершения всех передаваемых ему методов. Это, в частности, избавляет от необходимости вызывать метод *Wait*. И хотя это не гарантирует, что методы будут действительно выполняться параллельно, тем не менее, именно такое их выполнение предполагается, если система поддерживает несколько процессоров. Кроме того, в данном методе отсутствует возможность указать порядок выполнения методов от первого и до последнего, и этот порядок не может быть таким же, как и в списке аргументов.

1.4.3 Класс PLINQ и его методы

Также для параллельной обработки данных коллекций существует такой фреймворк, как PLINQ. Он позволяет работать в распределённом режиме с некоторой коллекцией данных (например, List).

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

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

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

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

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

Параллельная обработка запросов выполняется фреймворком PLINQ в три этапа, как показано на рисунке 3. Сначала PLINQ решает, сколько потоков следует использовать для выполнения запроса. Затем рабочие потоки извлекают свои фрагменты из исходной коллекции, под защитой блокировок. Все потоки выполняют свои задания независимо и помещают результаты в свои локальные очереди. В заключение, локальные результаты объединяются в единую коллекцию, которая подается в цикл foreach [10].

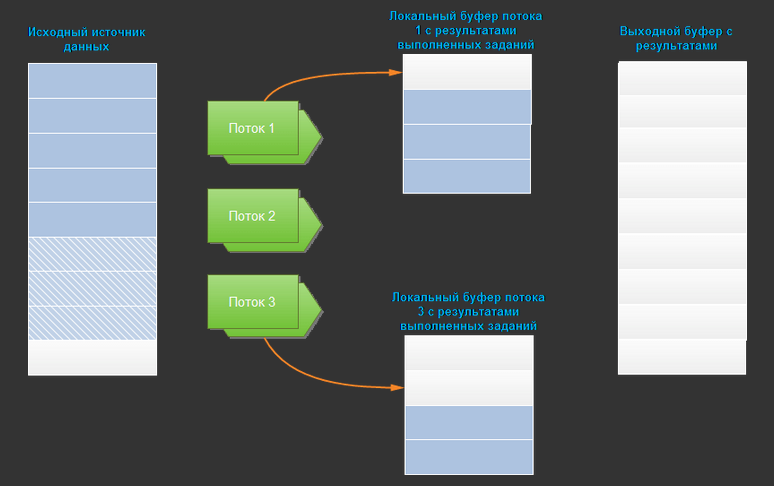


Рисунок 3 – Принцип работы фреймворка PLINQ

Метод *AsParallel* позволяет распараллелить запрос к источнику данных. Он реализован как метод расширения LINQ у массивов и коллекций. При вызове данного метода источник данных разделяется на части (если это возможно) и над каждой частью отдельно производятся операции.

Код также можно дополнительно оптимизировать с точки зрения параллелизации в случаях, когда для вывода результата параллельной операции используется цикл *foreach*. Его использование приводит к увеличению издержек ввиду того, что необходимо склеить полученные в разных потоках данные в один набор и затем их перебрать в цикле. Более оптимально в данном случае будет использовать метод *ForAll*, который выводит данные в том же потоке, в котором они обрабатываются. Отличие от *foreach* наглядно показано на рисунке 4.

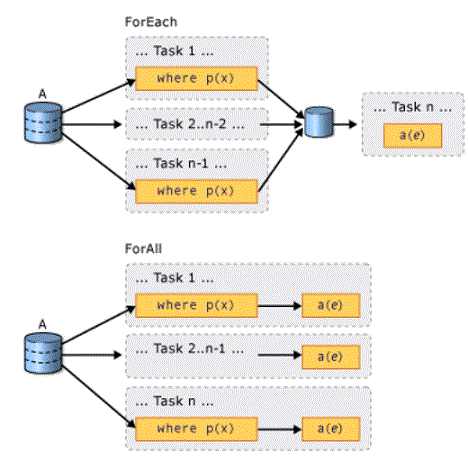


Рисунок 4 – Разница принципов работы foreach и ForAll

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ

2.1 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.

2.2 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.

2.3 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 оперативной 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.

2.4 Алгоритм решения СЛАУ с помощью метода LU-разложения

Данный алгоритм можно разделить на 3 функциональные части:

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

Информация об этом алгоритме и о нахождении данных алгоритма, таких как матрицы L и U, вектора Y и X, расположена в п. 1.1. Блок-схема распределённой реализации алгоритма изображена в приложении Б. Блок-схема линейной реализации алгоритма показана на рисунках 5-8.



Рисунок 5 – Обобщённая схема алгоритма метода LU-разложения



Рисунок 6 –Схема метода FindLU алгоритма метода LU-разложения



Рисунок 7 –Схема метода FindY алгоритма метода LU-разложения



Рисунок 8 –Схема метода FindX алгоритма метода LU-разложения

2.5 Алгоритм решения СЛАУ с помощью метода Гаусса

Данный алгоритм можно разделить на 2 функциональные части:

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

Информация об этом алгоритме расположена в п. 1.2. Блок-схема распределённой реализации алгоритма изображена в приложении Б. Блок-схема линейного алгоритма показана на рисунках 9-11.



Рисунок 9 – Обобщённая схема алгоритма метода Гаусса



Рисунок 10 –Схема метода ForwardElimination алгоритма метода Гаусса



Рисунок 11 –Схема метода BackSubstitution алгоритма метода Гаусса

## **2.6 Обоснование выбранного инструментария**

На сегодняшний день существует огромное множество языков программирования, с помощью которых можно решить поставленную задачу. Для выполнения данной курсовой работы был выбран язык программирования C# ввиду того, что он является простым, удобным, мощным и востребованным языком. В настоящий момент он используется в различных отраслях: от настольных решений до крупных веб-сервисов.

C# является языком с Си-подобным синтаксисом и оттого похож на такие известные языки, как C++ и Java. Он, как Java и С++, является объектно-ориентированным языком. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений, что поможет в разработке приложения для данного курсового проекта.

Данный язык имеет огромный инструментарий в виде стандартных классов платформы .NET Framework, NuGet-пакетов. В том числе среди них присутствует такая библиотека, как TPL, которая предоставляет удобные инструменты для создания распределённых алгоритмов. Это также послужило плюсом при выборе языка программирования.

Текущей версией языка является версия C# 7.0, которая вышла в 7 марта 2017 года. Данную версию и было решено использовать при разработке приложения.

3 РАЗРАБОТКА ПРОГРАММНОГО КОДА

3.1 Описание структуры проекта и его компонентов

В течении разработки данного программного продукта была создана следующая структура проекта (Рис. 12).

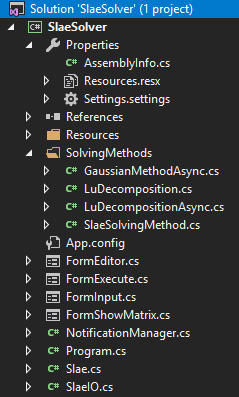


Рисунок 12 – Структура проекта разработанного приложения

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

Таблица 1 – Краткое описание классов 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 100 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. |
| IsCorrect | (1) public bool (double[] decision, double epsillon = 1)  (2) public bool (double[] decision, out int errorIndex, double epsillon = 1) | Проверяет, является ли решением данной СЛАУ входной параметр decision. Проверка производится с определённой точностью epsillon. Второй вариант данного перегруженного метода возвращает индекс строки, в которой решение отличается от необходимого значения на значение больше значения epsillon |

Статический класс *SlaeIO* служит для файлового ввода-вывода объектов СЛАУ. Методы данного класса представлены в таблице 7.

Таблица 7 – 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 разбор строки и преобразование этих данных в объект СЛАУ |
| Write | (1) public static void (Slae slae, Stream stream) | Производит запись в файл, объект файлового потока которого передаётся в качестве аргумента, |

Продолжение таблицы 7

|  |  |  |
| --- | --- | --- |
|  | (2) (double[][] matrix, double[] b, Stream stream)  (3) public static void (double[] decisions, Stream stream)  (4) private static void (string str, Stream stream) | данных, передаваемых в один из перегруженных методов. |

Интерфейс *ISlaeSolvingMethod* представляет собой интерфейс взаимодействия с методом решения СЛАУ. В программе служит для передачи коллекции объектов, реализующих данный интерфейс, в форму для поочерёдного вычисления решений СЛАУ. Интерфейс его взаимодействия представлен в таблице 8.

Таблица 8 – Интерфейс взаимодействия интерфейса *ISlaeSolvingMethod.cs*

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| Solve | double[] (Slae slae) | Метод, решающий систему линейных уравнений и возвращающий её решение. |

Класс *LuDecomposition* представляет собой линейный вариант метода решения СЛАУ «LU-разложение». Реализует интерфейс *ISlaeSolvingMethod*. Методы данного класса представлены в таблице 9.

Таблица 9 – Методы класса *LuDecomposition.cs*

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| Solve | public double[] (Slae slae) | Метод, решающий СЛУ и возвращающий её решение. |
| FindLU | private static void (Slae slae, out double[][] L, out double[][] U) | Метод для нахождения матриц L и U в соответствии с алгоритмом LU-разложения |
| FindY | private static double[] (Slae slae, double[][] L) | Метод для нахождения значений в соответствии с алгоритмом решения СЛАУ методом LU-разложения |
| FindX | private static double[] (Slae slae, double[][] U, double[] Y) | Метод для нахождения значений в соответствии с алгоритмом решения СЛАУ методом LU-разложения |

Класс *LuDecompositionAsync* представляет собой распределённый вариант метода решения СЛАУ «LU-разложение». Реализует интерфейс *ISlaeSolvingMethod*. Список методов данного класса полностью идентичен списку методов из линейного варианта данного способа решения СЛАУ и их можно найти в таблице 9.

Класс *GaussianMethodAsync* представляет собой распределённый вариант метода решения СЛАУ «метод Гаусса». Реализует интерфейс *ISlaeSolvingMethod*. Методы данного класса представлены в таблице 10.

Таблица 10 – Методы класса *GaussianMethodAsync.cs*

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| Solve | public double[] (Slae slae) | Метод, решающий систему линейных уравнений и возвращающий её решение. |
| ForwardElimination | private static void (double[][] A, double[] B, int n) | Метод, выполняющий т.н. «прямой ход» в соответствии с алгоритмом решения СЛАУ методом Гаусса |
| BackSubstitution | private static double[] (double[][] A, double[] B, int n) | Метод, выполняющий т.н. «обратный ход» в соответствии с алгоритмом решения СЛАУ методом Гаусса |

Класс *NotificationManager* предназначен для удобного вывода различных сообщений об ошибках, предупреждений и информационных сообщений средствами стандартного компонента *MessageBox*. Методы данного класса представлены в таблице 11.

Таблица 11 – Методы класса *NotificationManager.cs*

|  |  |  |
| --- | --- | --- |
| Название метода | Сигнатура | Описание |
| ShowError | (1) public static DialogResult (string message)  (2) public static DialogResult (Exception exception) | Метод, показывающий MessageBox, содержащий сообщение об ошибке. Может принимать как сообщение об ошибке как в виде строки, так и в виде объекта Exception. |
| ShowExclamation | public static DialogResult (string message, MessageBoxButtons btns = MessageBoxButtons.OK) | Метод, показывающий MessageBox, содержащий предупреждение. |

Продолжение таблицы 11

|  |  |  |
| --- | --- | --- |
| ShowInfo | public static DialogResult (string message) | Метод, показывающий MessageBox, содержащий информационное сообщение. |

3.2 Пользовательский интерфейс

Для разработанной программы был разработан интуитивно понятный пользовательский интерфейс. Все составляющие визуальной части данного приложения будут рассмотрены на рисунках 13-24. Функциональная составляющая данных форм не будет рассматриваться т.к. она была рассмотрена ранее в описании функционала одноимённых классов.

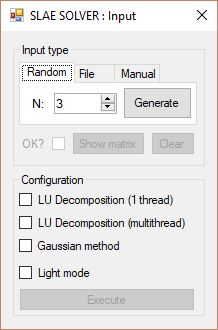


Рисунок 13 – Окно «Ввод данных» (*FormInput.cs*)

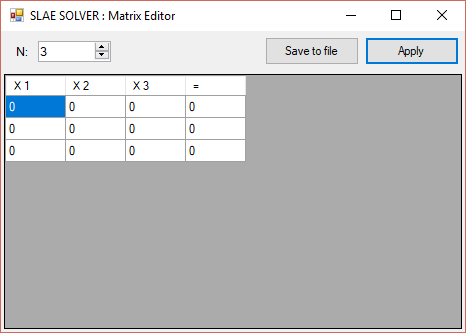


Рисунок 14 – Окно «Редактор матрицы» (*FormEditor.cs*)

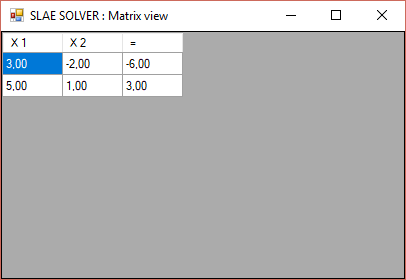


Рисунок 15 – Окно «Просмотр матрицы» (*FormShowMatrix.cs*) в режиме просмотра СЛАУ

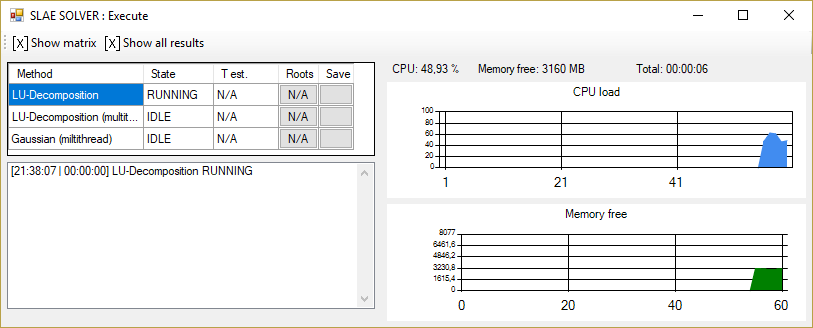


Рисунок 16 – Окно «Запуск вычислений» (*FormExecute.cs*) во время начала процесса вычисления

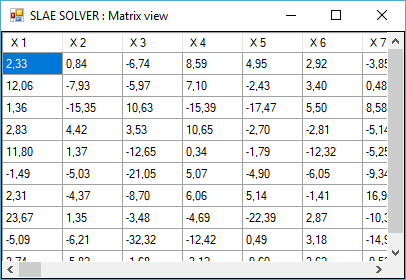


Рисунок 17 – Окно «Просмотр матрицы» (*FormShowMatrix.cs*) в режиме просмотра корней

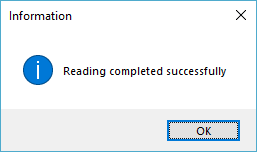


Рисунок 18 – Диалоговое окно с сообщением об успешном чтении данных

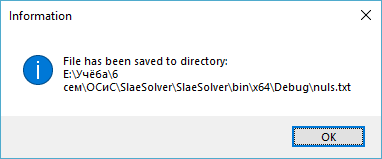


Рисунок 19 – Диалоговое окно с сообщением об успешном сохранении данных в файл

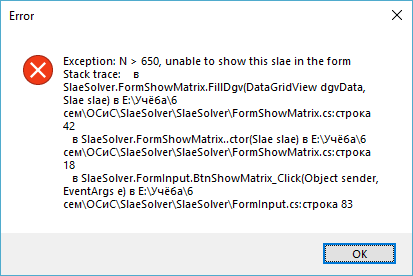


Рисунок 20 – Диалоговое окно с сообщением о невозможности вывода СЛАУ с текущим размером в элемент окна

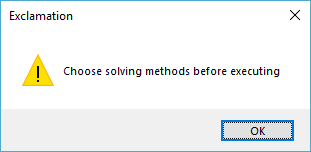


Рисунок 21 – Диалоговое окно с сообщением о необходимости выбора метода для решения СЛАУ

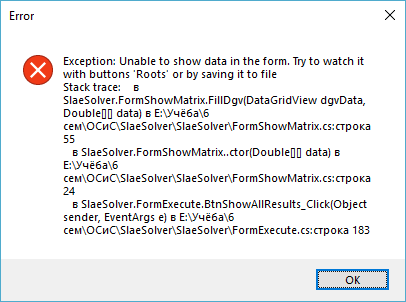


Рисунок 22 – Диалоговое окно с сообщением о невозможности отображения решений СЛАУ в элемент окна

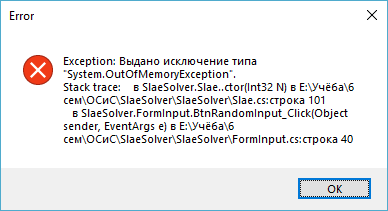


Рисунок 23 – Диалоговое окно с сообщением о нехватке памяти при попытке создания случайной матрицы порядка равном 50000

3.3 Анализ результатов и выводы

В процессе выполнения анализа работы программы был произведён запуск вычислений с помощью всех методов решения задачи с различным набором входных данных. При этом были поочерёдно сгенерированы матрицы размерами до 4000 с шагом в 1000 элементов. Дальнейшую генерацию матриц было решено не проводить ввиду больших временных затрат на каждую последующую итерацию тестирования. Показания временных затрат для каждого размера матрицы для каждого метода решения представлен в таблице 12.

Таблица 12 – Время выполнения вычислений

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы СЛАУ | Линейный вариант LU-разложения | Распределённый вариант LU-разложения | Распределённый метод Гаусса |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

В качестве использованного оборудования для вычислений был использован компьютер со следующими важными для производимых вычислений характеристиками:

* процессор: Intel Pentium N3540, 2666 MHz;
* объем оперативной памяти: 8.00 ГБ;
* операционная система: Windows 10 Pro (версия 1703, сборка 16299.431).

Анализируя показания временных затрат, можно сказать, что линейный вариант метода LU-разложения является самым продолжительным по времени вычисления, тогда как его распределённый вариант оказывается быстрее в среднем в раза. Реализация распределённого метода Гаусса сравнима с распределённым методом LU-разложения по времени до определённого значения порядка матрицы, после достижения которого метод Гаусса становится эффективнее в среднем в раза. Таким образом, можно заключить, что распределённый метод Гаусса является самым быстрым методом вычисления СЛАУ из представленных в программе.

Во время выполнения линейного алгоритма показатели загруженности процессора находятся в районе с учётом влияния фоновых процессов. Также по причине активности фоновых процессов показатели могут достигать значений , что является нормальным явлением. При запуске многопоточных процессов значения загруженности процесса находятся в промежутке , что является предсказуемым поведением для асинхронных задач.

Значения свободной оперативной памяти при заданных при тестировании размерах матриц отличались не так заметно, как заметны были колебания показателей загруженности процессора. Значения показателей свободной оперативной памяти на данной ЭВМ начинает заметно уменьшаться при размерах матрицы порядка 10000 и больше. При задании особо крупных значений размеров матрицы программа показывает ошибку, связанную с нехваткой памяти. Такая ошибка изображена на рисунке 23. Состояние показателей использования ресурсов на всё время вычислений при различных порядках матрицы отображены на рисунках 24-27.

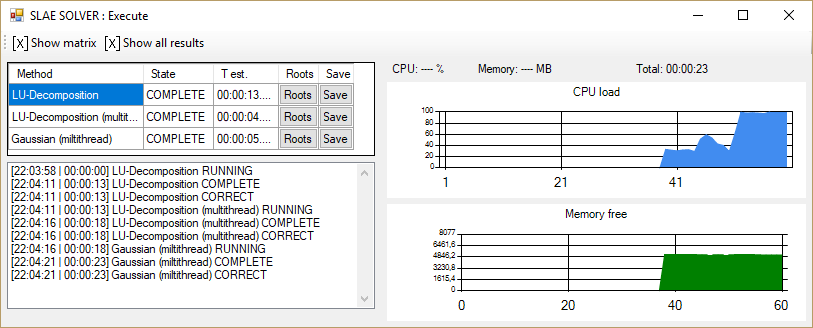


Рисунок 24 – Окно вычисления решений СЛАУ при порядке матрицы равном 1000

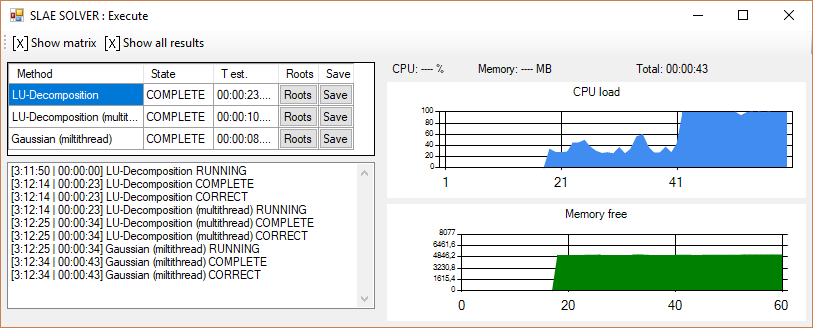


Рисунок 25 – Окно вычисления решений СЛАУ при порядке матрицы равном 1250

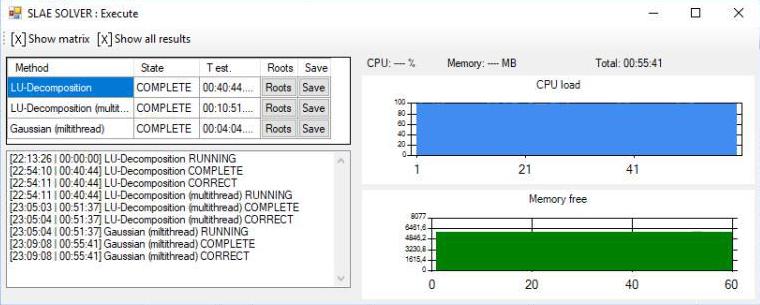


Рисунок 26 – Окно вычисления решений СЛАУ при порядке матрицы равном 3000

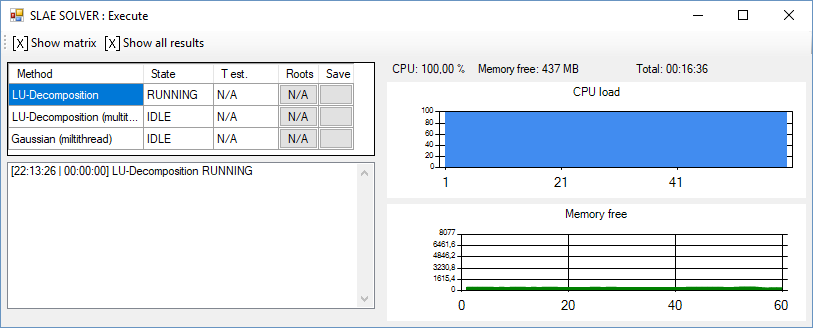


Рисунок 27 – Окно вычисления решений СЛАУ при порядке матрицы равном 30000

Судя по представленным выше рисункам можно судить, что исходные данные в разработанной программе обрабатываются правильно ввиду того, что в журнал производится запись правильности решения вычисленным методом и ни в одной из полученных в процессе тестирования записей не было получено сообщения «INCORRECT [N]», где N является номером уравнения СЛАУ, где погрешность превысила допустимые значения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта было разработано Windows-приложение для решения систем линейных алгебраических уравнений линейным и распределённым методами LU-разложения и распределённым методом Гаусса. 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 методах решения СЛАУ, таких как метод LU-разложения и метод Гаусса. На основе данного материала в дальнейшем были разработаны алгоритмы решения поставленной задачи.

Затем был изучен принцип распараллеливания процессов и средства для распараллеливания в языке программирования C#. Данные знания пригодились при выполнении расчетов в вышеуказанных алгоритмах численных методов.

После получения вышеуказанных сведений был разработан указанный программный продукт. При его использовании конечный пользователь может использовать данные о СЛАУ из различных источников (в т.ч. сгенерировать матрицу случайного размера), выбрать используемые для вычислений методы и запустить вычисления с функцией отслеживания показателей ресурсов или без неё.

В итоге был приобретен опыт разработки подобных приложений с помощью языка программирования C# и интерфейса для разработки оконных приложений WinForms. Приложение было разработано в среде разработки Microsoft Visual Studio Community 2017.

Поставленные задачи в курсовом проекте были решены полностью.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Ортега, Дж. Введение в параллельные и векторные методы решения линейных систем. — М.: Мир, 1991. — 376 с.
  2. Голуб Дж., Ван Лоун Ч. Матричные вычисления. – М.: Мир, 1999.
  3. Метод Гаусса [Электронный ресурс]. – Режим доступа: https://function-x.ru/cpp\_metod\_gaussa.html Дата доступа: 24.04.2018
  4. Метод Гаусса [Электронный ресурс]. – Режим доступа: https://studfiles.net/preview/1938346/page:3/ Дата доступа: 18.04.2018
  5. Введение в параллельное программирование [Электронный ресурс]. – Режим доступа: https://www.intuit.ru/studies/courses/4807/1055/lecture/16369 Дата доступа: 27.04.2018
  6. Введение в многопоточность. Класс Thread [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/11.1.php Дата доступа: 27.04.2018
  7. Многопоточность в .NET Framework [Электронный ресурс]. – Режим доступа: https://www.intuit.ru/studies/courses/4807/1055/lecture/16374 Дата доступа: 27.04.2018
  8. Введение в параллельное программирование .NET Framework 4 [Электронный ресурс]. – Режим доступа: https://www.intuit.ru/studies/courses/4807/1055/lecture/16371 Дата доступа: 27.04.2018
  9. Класс Parallel [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/csharp/thread\_and\_files/level2/2\_6.php Дата доступа: 27.04.2018
  10. Параллелизм данных [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/csharp/optimization/level4/4\_4.php Дата доступа: 28.04.2018

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

**Файл Program.cs**

using System;

using System.Windows.Forms;

namespace SlaeSolver

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new FormInput());

}

}

}

**Файл FormInput.cs**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace SlaeSolver

{

public partial class FormInput : Form

{

Slae slae;

public Slae Slae

{

get { return slae; }

private set { slae = value; SlaeChangingListener(); }

}

private void SlaeChangingListener()

{

cbStateOk.Checked = (slae == null) ? false : true;

btnShowMatrix.Enabled = btnClear.Enabled = btnExecute.Enabled = cbStateOk.Checked;

GC.Collect();

}

public FormInput()

{

InitializeComponent();

}

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

{

btnRandomInput.Text = "Generating";

btnRandomInput.Enabled = false;

try

{

Slae = new Slae((int)numudRandomN.Value);

}

catch (Exception exc)

{

NotificationManager.ShowError(exc);

}

finally

{

GC.Collect();

btnRandomInput.Text = "Generate";

btnRandomInput.Enabled = true;

}

}

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

{

OpenFileDialog ofd;

if ((ofd = new OpenFileDialog()).ShowDialog() == DialogResult.OK)

{

try

{

using (var stream = ofd.OpenFile())

{

Slae = SlaeIO.Read(stream);

}

NotificationManager.ShowInfo("Reading completed successfully");

}

catch(Exception exc)

{

NotificationManager.ShowError(exc);

}

}

}

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

{

new FormEditor((val) => Slae = val).ShowDialog();

}

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

{

try

{

new FormShowMatrix(slae).ShowDialog();

}

catch(Exception exc)

{

NotificationManager.ShowError(exc);

}

}

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

{

Slae = null;

}

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

{

List<ISlaeSolvingMethod> slaeSolvingMethods = new List<ISlaeSolvingMethod>();

if (cbLuDecomposition.Checked)

slaeSolvingMethods.Add(new LuDecomposition());

if (cbLuDecompositionMulti.Checked)

slaeSolvingMethods.Add(new LuDecompositionAsync());

if (cbGaussian.Checked)

slaeSolvingMethods.Add(new GaussianMethod());

if (slaeSolvingMethods.Count == 0)

NotificationManager.ShowExclamation("Choose solving methods before executing");

else

{

new FormExecute(slae, slaeSolvingMethods, cbLight.Checked).ShowDialog();

GC.Collect();

}

}

}

}

**Файл FormEditor.cs**

using System;

using System.IO;

using System.Windows.Forms;

namespace SlaeSolver

{

public delegate void SlaeSetter(Slae values);

public partial class FormEditor : Form

{

public int N { get; private set; }

public double[][] Matrix { get; private set; }

public double[] B { get; private set; }

SlaeSetter ss;

public FormEditor(SlaeSetter ss)

{

InitializeComponent();

sfdSave.InitialDirectory = Directory.GetCurrentDirectory();

NumudN\_ValueChanged(this, EventArgs.Empty);

this.ss = ss;

}

private void CreateDgvTable()

{

Matrix = new double[N][];

B = new double[N];

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

Matrix[i] = new double[N];

CreateDgvTable(dgvData, N);

}

public static void CreateDgvTable(DataGridView dgvData, int N)

{

dgvData.Rows.Clear();

dgvData.Columns.Clear();

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

{

string col = $"X {i+1}";

dgvData.Columns.Add(col, col);

dgvData.Columns.GetLastColumn(DataGridViewElementStates.Visible, DataGridViewElementStates.None).Width = 60;

}

dgvData.Columns.Add("equal", "=");

dgvData.Columns.GetLastColumn(DataGridViewElementStates.Visible, DataGridViewElementStates.None).Width = 60;

dgvData.Rows.Add(N);

}

private void NumudN\_ValueChanged(object sender, EventArgs e)

{

N = (int)numudN.Value;

CreateDgvTable();

}

private void DgvData\_CellValueChanged(object sender, DataGridViewCellEventArgs e)

{

if (e.ColumnIndex != N) // if not at equal place

{

if (!Double.TryParse(dgvData[e.ColumnIndex, e.RowIndex].Value.ToString(), out Matrix[e.RowIndex][e.ColumnIndex]))

dgvData[e.ColumnIndex, e.RowIndex].Value = Matrix[e.RowIndex][e.ColumnIndex] = 0;

}

else

if (!Double.TryParse(dgvData[e.ColumnIndex, e.RowIndex].Value.ToString(), out B[e.RowIndex]))

dgvData[e.ColumnIndex, e.RowIndex].Value = B[e.RowIndex] = 0;

}

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

{

if (sfdSave.ShowDialog() == DialogResult.OK)

{

using (var stream = sfdSave.OpenFile())

{

SlaeIO.Write(Matrix, B, stream);

}

NotificationManager.ShowInfo($"File has been saved to directory:{Environment.NewLine}{sfdSave.FileName}");

}

}

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

{

for (int i = 0; i < dgvData.Rows.Count; i++)

{

DataGridViewRow row = dgvData.Rows[i];

for (int j = 0; j < row.Cells.Count - 1; j++) // contains equal row, so use without last

{

Matrix[i][j] = Convert.ToDouble(dgvData[j, i].Value);

}

}

ss(new Slae(Matrix, B));

DialogResult = DialogResult.OK;

Close();

}

}

}

**Файл FormShowMatrix.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SlaeSolver

{

public partial class FormShowMatrix : Form

{

public FormShowMatrix(Slae slae)

{

InitializeComponent();

FillDgv(dgvData, slae);

}

public FormShowMatrix(double[][] data)

{

InitializeComponent();

FillDgv(dgvData, data);

}

public static void FillDgv(DataGridView dgvData, Slae slae)

{

if (slae.N <= 650)

{

FormEditor.CreateDgvTable(dgvData, slae.N);

for (int i = 0; i < slae.N; i++)

{

for (int j = 0; j < slae.N + 1; j++)

{

dgvData[j, i].Value = slae[i, j];

}

}

}

else

throw new ArgumentException("N > 650, unable to show this slae in the form");

}

public static void FillDgv(DataGridView dgvData, double[][] data)

{

if (data.Length != 0)

{

if (data[0].Length > 650)

if(data.Length == 1)

data = ModifyArray(data[0], 100);

else

throw new ArgumentException("Unable to show data in the form. " +

"Try to watch it with buttons \'Roots\' or by saving it to file");

CreateDgvTable(dgvData, data.Length, data[0].Length);

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

{

for (int j = 0; j < data[i].Length; j++)

{

dgvData[j, i].Value = data[i][j];

}

}

}

else

throw new ArgumentException("Unable to show data in the form");

}

public static double[][] ModifyArray(double[] data, int xCount)

{

int elementsLeft = data.Length;

int iterations = (int)Math.Ceiling((elementsLeft / (double)xCount));

//double[] tmp = new double[xCount];

double[][] newData = new double[iterations][];

for (int k = 0; k < iterations - 1; k++, elementsLeft -= xCount)

{

newData[k] = new double[xCount];

Array.Copy(data, xCount \* k, newData[k], 0, xCount);

}

newData[iterations - 1] = new double[xCount];

Array.Copy(data, xCount \* (iterations - 1), newData[iterations - 1], 0, elementsLeft);

for (int i = elementsLeft; i < xCount; i++)

newData[iterations - 1][i] = double.NaN;

return newData;

}

// NxM

public static void CreateDgvTable(DataGridView dgvData, int N, int M)

{

dgvData.Rows.Clear();

dgvData.Columns.Clear();

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

{

string col = $"X {i + 1}";

dgvData.Columns.Add(col, col);

dgvData.Columns.GetLastColumn(DataGridViewElementStates.Visible, DataGridViewElementStates.None).Width = 60;

}

dgvData.Rows.Add(N);

}

}

}

**Файл FormExecute.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SlaeSolver

{

public enum RunningStates { IDLE, RUNNING, COMPLETE, FAILURE}

public partial class FormExecute : Form

{

Slae slae;

double[][] results;

List<ISlaeSolvingMethod> solvingMethods;

int activeMethod;

bool lightMode;

PerformanceCounter cpuCounter, memCounter;

TimeSpan totalT;

static int perfomanceCounterSeconds = 60;

Queue<double> cpuQueue, memQueue;

public FormExecute(Slae slae, List<ISlaeSolvingMethod> solvingMethods, bool lightMode = false)

{

if (solvingMethods.Count == 0)

throw new ArgumentException("Solving Methods count = 0");

else

{

InitializeComponent();

this.slae = slae;

this.solvingMethods = solvingMethods;

results = new double[solvingMethods.Count][];

this.lightMode = lightMode;

if(!lightMode)

{

totalT = new TimeSpan();

ConfigurePerfomanceCounters();

}

else

{

CleanupWnd();

}

}

}

private void ConfigurePerfomanceCounters()

{

cpuCounter = new PerformanceCounter("Processor", "% Processor Time", "\_Total");

memCounter = new PerformanceCounter("Memory", "Available MBytes");// "% Committed Bytes In Use");//

cpuQueue = new Queue<double>(new double[perfomanceCounterSeconds]);

memQueue = new Queue<double>(new double[perfomanceCounterSeconds]);

chartCpu.ChartAreas[0].AxisY.Minimum = chartMem.ChartAreas[0].AxisY.Minimum = 0;

chartCpu.ChartAreas[0].AxisY.Maximum = 100;

chartMem.ChartAreas[0].AxisY.Maximum =

new Microsoft.VisualBasic.Devices.ComputerInfo().TotalPhysicalMemory / 1024 / 1024;

}

private void DeactivatePerfomanceCounter()

{

timerPerfomance.Stop();

lbCpuLoad.Text = "CPU: ---- %";

lbMemoryLoad.Text = "Memory: ---- MB";

}

private void CleanupWnd()

{

chartCpu = null;

chartMem = null;

cpuQueue = null;

memQueue = null;

lbCpuLoad = null;

lbFullT = null;

lbMemoryLoad = null;

timerPerfomance = null;

tableLayoutPanel3 = null;

Size = MinimumSize = new Size(400, 275);

FormBorderStyle = FormBorderStyle.FixedSingle;

MaximizeBox = false;

GC.Collect();

}

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

{

float cpuValue = cpuCounter.NextValue(), memValue = memCounter.NextValue();

lbCpuLoad.Text = $"CPU: {cpuValue:f2} %";

lbMemoryLoad.Text = $"Memory free: {memValue} MB";

totalT = totalT.Add(TimeSpan.FromSeconds(1));

lbFullT.Text = $"Total: {totalT}";

UpdatePerfomanceCharts(cpuValue, memValue);

}

private void UpdatePerfomanceCharts(float cpuValue, float memValue)

{

cpuQueue.Dequeue();

cpuQueue.Enqueue(cpuValue);

memQueue.Dequeue();

memQueue.Enqueue(memValue);

chartCpu.Series[0].Points.Clear();

chartMem.Series[0].Points.Clear();

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

{

chartCpu.Series[0].Points.AddXY(i + 1, cpuQueue.ElementAt(i));

chartMem.Series[0].Points.AddXY(i + 1, memQueue.ElementAt(i));

}

}

private void AddLog(RunningStates state)

{

lbLog.Items.Add($"[{DateTime.Now.ToLongTimeString()}{(lightMode ? "" : $" | {totalT}")}]" +

$" {solvingMethods[activeMethod].ToString()} {state}");

}

private void AddLog(string value)

{

lbLog.Items.Add($"[{DateTime.Now.ToLongTimeString()}{(lightMode ? "" : $" | {totalT}")}]" +

$" {solvingMethods[activeMethod].ToString()} {value}");

}

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

{

try

{

new FormShowMatrix(slae).ShowDialog();

}

catch (Exception exc)

{

NotificationManager.ShowError(exc);

}

}

private void DgvResults\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex != -1)

{

if (results[e.RowIndex] != null)

{

try

{

switch (e.ColumnIndex)

{

case 3:

// roots

new FormShowMatrix(new double[1][] { results[e.RowIndex] }).ShowDialog();

break;

case 4:

// save

if (sfdSave.ShowDialog() == DialogResult.OK)

{

using (var stream = sfdSave.OpenFile())

{

SlaeIO.Write(results[e.RowIndex], stream);

}

NotificationManager.ShowInfo($"File has been saved to directory:{Environment.NewLine}{sfdSave.FileName}");

}

break;

}

}

catch (Exception exc)

{

NotificationManager.ShowError(exc);

}

}

}

}

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

{

try

{

new FormShowMatrix(results).ShowDialog();

}

catch (Exception exc)

{

NotificationManager.ShowError(exc);

}

}

private async void FormExecute\_Load(object sender, EventArgs e)

{

for (activeMethod = 0; activeMethod < solvingMethods.Count; activeMethod++)

{

dgvResults.Rows.Add(solvingMethods[activeMethod].ToString(), RunningStates.IDLE.ToString(), "N/A");

}

if (!lightMode)

timerPerfomance.Start();

for (activeMethod = 0; activeMethod < solvingMethods.Count; activeMethod++)

{

try

{

dgvResults[1, activeMethod].Value = RunningStates.RUNNING;

AddLog(RunningStates.RUNNING);

DateTime prevT = DateTime.Now;

results[activeMethod] = await ExecMethodAsync(solvingMethods[activeMethod], slae);

dgvResults[1, activeMethod].Value = RunningStates.COMPLETE;

AddLog(RunningStates.COMPLETE);

dgvResults[2, activeMethod].Value = (DateTime.Now - prevT).ToString();

dgvResults[3, activeMethod].Value = "Roots";

dgvResults[4, activeMethod].Value = "Save";

AddLog(slae.IsCorrect(results[activeMethod], out int errorIndex, 5) ? "CORRECT" : $"INCORRECT [{errorIndex}]");

}

catch (Exception exc)

{

NotificationManager.ShowError(exc);

AddLog(RunningStates.FAILURE);

dgvResults[1, activeMethod].Value = RunningStates.FAILURE;

}

}

DeactivatePerfomanceCounter();

}

private static async Task<double[]> ExecMethodAsync(ISlaeSolvingMethod method, Slae slae)

{

return await Task<double[]>.Factory.StartNew(() =>

{

return method.Solve(slae);

});

}

}

}

**Файл NotificationManager.cs**

using System;

using System.Windows.Forms;

namespace SlaeSolver

{

public static class NotificationManager

{

public static DialogResult ShowError(string message)

{

return MessageBox.Show(message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

public static DialogResult ShowExclamation(string message, MessageBoxButtons btns = MessageBoxButtons.OK)

{

return MessageBox.Show(message, "Exclamation", btns, MessageBoxIcon.Exclamation);

}

public static DialogResult ShowInfo(string message)

{

return MessageBox.Show(message, "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

public static DialogResult ShowError(Exception exception)

{

return ShowError($"Exception: {exception.Message}" +

$"{Environment.NewLine}Stack trace: {exception.StackTrace}");

}

}

}

**Файл Slae.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace SlaeSolver

{

public class Slae

{

public int N { get { return B.Length; } }

private double[][] matrix;

public double[][] Matrix

{

get { return matrix; }

private set

{

matrix = value;

}

}

public double[] B { get; private set; }

public bool IsCorrect(double[] decision, double epsillon = 1)

{

bool isCorrect = true;

for (int i = 0; i < N && isCorrect; i++)

{

double s = 0;

for (int j = 0; j < N; j++)

{

s += decision[j] \* Matrix[i][j];

}

double difference = Math.Abs(Math.Abs(s) - Math.Abs(B[i]));

if (difference > epsillon)

{

System.Diagnostics.Debug.WriteLine($"{Math.Abs(s)} - {Math.Abs(B[i])}" +

$" = {difference} AT row [{i}]");

isCorrect = false;

}

}

return isCorrect;

}

public bool IsCorrect(double[] decision, out int errorIndex, double epsillon = 1)

{

bool isCorrect = true;

errorIndex = -1;

for (int i = 0; i < N && isCorrect; i++)

{

double s = 0;

for (int j = 0; j < N; j++)

{

s += decision[j] \* Matrix[i][j];

}

double difference = Math.Abs(Math.Abs(s) - Math.Abs(B[i]));

if (difference > epsillon)

{

System.Diagnostics.Debug.WriteLine($"{Math.Abs(s)} - {Math.Abs(B[i])}" +

$" = {difference} AT row [{i}]");

isCorrect = false;

errorIndex = i;

}

}

return isCorrect;

}

public double this[int i, int j]

{

// can return B[i] as there's Matrix[N;N+1]

get { return j == N ? B[i] : Matrix[i][j]; }

}

/// <summary>

/// Initializing Slae with matrix and array of right part

/// </summary>

/// <param name="matrix">matrix of the SLAE A</param>

/// <param name="b">matrix of free coeffs B</param>

public Slae(double[][] matrix, double[] b)

{

if (matrix.Length == b.Length)

{

Matrix = matrix;

B = b;

}

else

throw new ArgumentException();

}

/// <summary>

/// Creates random slae with size of NxN

/// </summary>

/// <param name="N">Size of the matrix NxN</param>

public Slae(int N)

{

Random random = new Random();

Matrix = new double[N][];

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

{

Matrix[i] = new double[N];

Matrix[i][i] = 1;

}

B = new double[N];

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

{

var next = random.Next();

double k0 = ((next % (N + 1)) - 250) / 100.0;

int k1 = random.Next() % N;

int k2 = random.Next() % N;

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

{

Matrix[j][i] = Matrix[i][j] = random.NextDouble() \* k0 % 25;

B[i] += (j \* (k0 \* (k1 + k2)) \* random.NextDouble()) % 25;

}

}

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

B[i] = (B[i] + random.Next()) % 25;

}

public string GetSLAEString()

{

return GetSLAEString(Matrix, B);

}

public static string GetSLAEString(double[][] values, double[] b)

{

StringBuilder stringBuilder = new StringBuilder();

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

{

for (int j = 0; j < values[0].Length; j++)

stringBuilder.Append($"{ values[i][j] } ");

stringBuilder.Append($"{b[i]}\r\n");

}

return stringBuilder.ToString();

}

}

}

**Файл SlaeIO.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

namespace SlaeSolver

{

public static class SlaeIO

{

static Encoding encoding = Encoding.UTF8;

public static Slae Read(Stream stream)

{

// warn: big files may not be read?

byte[] data = new byte[(int)stream.Length];

stream.Read(data, 0, (int)stream.Length);

return ParseSlaeString(encoding.GetString(data));

}

/// <summary>

/// Parsing input string to SLAE

/// </summary>

/// <param name="slaeString">String which contains elements of slae</param>

private static Slae ParseSlaeString(string slaeString)

{

System.Text.RegularExpressions.Regex rowSplitter = new

System.Text.RegularExpressions.Regex(@"\r?\n");// \\d+\\.?\\d?\\s

var rows = rowSplitter.Split(slaeString).Where(r => r.Length != 0).ToList();

double[][] Matrix = new double[rows.Count][];

double[] B = new double[rows.Count];

for (int i = 0; i < rows.Count; i++)

{

string[] elements = rows[i].Split(' ');

if (rows.Count + 1 == elements.Length)

{

List<double> readValues = new List<double>();

for (int j = 0; j < elements.Length - 1; j++)

readValues.Add(Double.Parse(elements[j]));

var b = Double.Parse(elements[elements.Length - 1]);

Matrix[i] = readValues.ToArray();

B[i] = b;

}

else

throw new ArgumentException();

}

return new Slae(Matrix, B);

}

public static void Write(Slae slae, Stream stream)

{

Write(slae.GetSLAEString(), stream);

}

public static void Write(double[][] matrix, double[] b, Stream stream)

{

Write(Slae.GetSLAEString(matrix, b), stream);

}

public static void Write(double[] decisions, Stream stream)

{

StringBuilder sb = new StringBuilder();

foreach (var d in decisions)

sb.Append($"{d} ");

Write(sb.ToString(), stream);

}

private static void Write(string str, Stream stream)

{

var bytes = encoding.GetBytes(str);

stream.Write(bytes, 0, bytes.Length);

}

}

}

**Файл ISlaeSolvingMethod.cs**

namespace SlaeSolver

{

//public delegate double[] SlaeSolvingMethod(Slae slae);

public interface ISlaeSolvingMethod

{

double[] Solve(Slae slae);

}

}

**Файл LuDecomposition.cs**

namespace SlaeSolver

{

public class LuDecomposition : ISlaeSolvingMethod

{

public double[] Solve(Slae slae)

{

double[][] LU = FindLU(slae);

var Y = FindY(slae, LU);

var X = FindX(slae, LU, Y);

return X;

}

private static double[][] FindLU(Slae slae)

{

double[][] LU = new double[slae.N][];

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

LU[i] = new double[slae.N];

for (int i = 0; i < slae.N; i++)

{

for (int j = i; j < slae.N; j++)

{

double sum = 0;

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

sum += LU[i][k] \* LU[k][j];

LU[i][j] = slae.Matrix[i][j] - sum;

}

for (int j = i + 1; j < slae.N; j++)

{

double sum = 0;

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

sum += LU[j][k] \* LU[k][i];

LU[j][i] = (1 / LU[i][i]) \* (slae.Matrix[j][i] - sum);

}

}

return LU;

}

private static double[] FindY(Slae slae, double[][] LU)

{

double[] y = new double[slae.N];

double sum;

for (int i = 0; i < slae.N; i++)

{

sum = 0;

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

sum += LU[i][k] \* y[k];

y[i] = slae.B[i] - sum;

}

return y;

}

public static double[] FindX(Slae slae, double[][] LU, double[] y)

{

double[] x = new double[slae.N];

double sum;

for (int i = slae.N - 1; i >= 0; i--)

{

sum = 0;

for (int k = i + 1; k < slae.N; k++)

sum += LU[i][k] \* x[k];

x[i] = (y[i] - sum) / LU[i][i];

}

return x;

}

public override string ToString()

{

return "LU-Decomposition";

}

}

}

**Файл LuDecompositionAsync.cs**

using System.Threading.Tasks;

namespace SlaeSolver

{

public class LuDecompositionAsync : ISlaeSolvingMethod

{

public double[] Solve(Slae slae)

{

double[][] LU = FindLU(slae);

var Y = FindY(slae, LU);

var X = FindX(slae, LU, Y);

return X;

}

private static double[][] FindLU(Slae slae)

{

double[][] LU = new double[slae.N][];

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

LU[i] = new double[slae.N];

for (int i = 0; i < slae.N; i++)

{

Parallel.For(i, slae.N, j =>

{

double sum = 0;

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

sum += LU[i][k] \* LU[k][j];

LU[i][j] = slae.Matrix[i][j] - sum;

});

Parallel.For(i + 1, slae.N, j =>

{

double sum = 0;

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

sum += LU[j][k] \* LU[k][i];

LU[j][i] = (slae.Matrix[j][i] - sum) / LU[i][i];

});

}

return LU;

}

private double[] FindY(Slae slae, double[][] LU)

{

double[] y = new double[slae.N];

for (int i = 0; i < slae.N; i++)

{

double sum = 0;

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

sum += LU[i][k] \* y[k];

y[i] = slae.B[i] - sum;

}

return y;

}

public double[] FindX(Slae slae, double[][] LU, double[] y)

{

double[] x = new double[slae.N];

for (int i = slae.N - 1; i >= 0; i--)

{

double sum = 0;

for (int k = i + 1; k < slae.N; k++)

sum += LU[i][k] \* x[k];

x[i] = (y[i] - sum) / LU[i][i];

}

return x;

}

public override string ToString()

{

return "LU-Decomposition (multithread)";

}

}

}

**Файл GaussianMethodAsync.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SlaeSolver

{

public class GaussianMethod : ISlaeSolvingMethod

{

public double[] Solve(Slae slae)

{

// Preparation : copying arrays of slae

// cuz, u know, it changes data in Slae

// object if we'll use data from slae object properly

double[][] A = new double[slae.N][];

Parallel.For(0, slae.N, (i) =>

{

A[i] = new double[slae.N];

System.Array.Copy(slae.Matrix[i], A[i], slae.N);

});

double[] B = new double[slae.N];

Array.Copy(slae.B, B, slae.N);

ForwardElimination(A, B, slae.N);

return BackSubstitution(A,B,slae.N);

}

// прямой ход

private static void ForwardElimination(double[][] A, double[] B, int n)

{

for (int k = 0; k < n; k++)

{

Parallel.For(k + 1, n, j =>

{

double d = A[j][k] / A[k][k];

for (int i = k; i < n; i++)

{

A[j][i] = A[j][i] - d \* A[k][i];

}

B[j] = B[j] - d \* B[k];

});

}

}

// обратный ход

private static double[] BackSubstitution(double[][] A, double[] B, int n)

{

double[] X = new double[n];

for (int k = n - 1; k >= 0; k--)

{

double d = 0;

for (int j = k + 1; j < n; j++)

{

double s = A[k][j] \* X[j];

d = d + s;

}

X[k] = (B[k] - d) / A[k][k];

}

return X;

}

public override string ToString()

{

return "Gaussian (miltithread)";

}

}

}