**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 «Архитектура вычислительных систем»

на тему: **«РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ В РАСПРЕДЕЛЁННЫХ СИСТЕМАХ ОБРАБОТКИ ИНФОРМАЦИИ НА ОСНОВЕ OPENMP»**

Исполнитель: студентка гр. ИП-31

Процкая М. А.

Руководитель: старший преподаватель

Косинов Г. П.

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

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

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

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

Подписи членов комиссии

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

Гомель 2019

**Содержание**

Введение……………………………………………………..……………………...3

1 Обзор существующих методов решения задачи………………..……..………..4

1.1 LU-разложение……………………………....……….…….………….…..4

1.2 Решение СЛАУ методом LU-разложения....………….…..………….…..5

1.3 Решение СЛАУ методом Гаусса…………....……….……….……….…..6

[1.4](#_Toc468949622) Технология OpenMP………………....…………..…..………….……….11

1.5 GDI графический интерфейс…………………...……………………….14

[2](#_Toc468949627) Постановка задачи……………...……………………………………………….18

3 Алгоритмический анализ.………………………………………….…………...19

3.1 Анализ исходных данных и результатов……………………………….19

3.2 Алгоритм решения системы линейных уравнений методом LU- разложения…………………………….....……………………….…...…..…19

3.3 Алгоритм решения системы линейных уравнений методом Гаусса....20

4 Разработка программного кода…….…………………………..……………….21

[5 Разработка графического интерфейса………………...………..………………2](#_Toc468949632)3

[6 Анализ результатов работы и выводы..………………...………………………2](#_Toc468949632)4

[7 Тестирование и верификация………..………………...…………….……….…2](#_Toc468949632)8

[7.1 Тестирование пользовательского интерфейса…...…………….………2](#_Toc468949632)8

[7.2 Тестирование функционала………..………...……...…………..………](#_Toc468949632)30

[Заключение](#_Toc468949633)……………………………………………………………..………….32

[Список использованной литературы](#_Toc468949634)……………………………………..………33

[Приложение А](#_Toc468949635)……………………………………………………………..………34

Приложение Б……………………………………………………………………...56

**ВВЕДЕНИЕ**

Решение систем линейных алгебраических уравнений (СЛАУ) - одна из основных задач вычислительной линейной алгебры. Несмотря на то, что задача решения именно системы линейных уравнений сравнительно редко представляет самостоятельный интерес для прикладных задач, то от умения эффективно решать данные системы часто зависит сама возможность математического моделирования самых разнообразных процессов с применением ЭВМ. Значительная часть численных методов решения различных, в особенности – нелинейных, задач включает в себя решение систем линейных уравнений как элементарный шаг соответствующего алгоритма. код программный гаусс поток

С развитием вычислительной техники, увеличением математических расчетов появляются проблемы, которые заключаются в длительном времени выполнения вычислений и увеличении используемых ресурсов вычислительной машины. Эти проблемы может решить использование многопоточности.

Многопоточность – это возможность параллельно выполнять несколько видов операций в одной прикладной программе. Параллельные вычисления гораздо удобнее реализовывать не на уровне процессов, но на уровне задач. И программа, разработанная с использованием механизма потоков, представляемая как некоторое множество задач в рамках одного процесса, может быть выполнена быстрее за счет параллельного функционирования её отдельных частей. Особенно это выгодно при наличии нескольких процессоров, ибо каждая задача может выполняться на отдельном процессоре. Также, особенно эффективно можно использовать многопоточность для выполнения распределенных приложений.

Мотивация создания распределённого приложения:

* модульность, некоторые части должны быть относительно автономны;
* безопасность, приложение должно взаимодействовать как с сервисами из Internet так и с внутренними корпоративными, в связи с чем некоторые его части могут находиться в различных сегментах сети;
* максимально эффективная утилизация ресурсов оборудования из-за серьезных ограничений на цену транзакции, в противном случае создание и использование приложения может терять экономический смысл;
* требования определённого уровня отказоустойчивости.

Целью курсового проекта является получение навыков использования знаний в области параллельных вычислений, а также знаний в области применения современных программных систем и технологий программирования при создании и реализации прикладных программных приложений в различных предметных областях.

**1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧ**

**1.1 LU-разложение**

Пусть  - данная  матрица, а  и - соотвественно нижняя (левая) и верхняя (правая) треугольные матрицы.

Теорема. Если все главные миноры квадратной матрицы  отличны от нуля, то существуют такие нижняя  и верхняя  треугольные матрицы, что . Если элементы диагонали одной из матриц  или  фиксированы (ненулевые), то такое разложение единственно.

Ниже рассматриваются формулы для фактического разложения матриц в случае фиксирования диагонали нижней треугольной матрицы . Найдем  (при ) и  (при ) такие, что

(1.1)

После перемножения матриц (1) получим следующую систему

(1.2)

Эта система позволяет находить неизвестные  и  в следующем порядке. Из первой строки уравнений имеем

(1.3)

 из оставшейся части первого столбца имеем

(1.4)

  из оставшейся части второй строки имеем

(1.5)

 из оставшейся части второго столбца имеем

(1.6)

 и т.д. Обобщая, все элементы матрицы неизвестных могут быть получены по формулам

(1.7)

(1.8)

**1.2 Решение СЛАУ методом LU-разложения**

После разложения матрицы *А* в *L* и *U*, вместо уравнения можно записать уравнение . Введя вспомогательный вектор , последнее равенство можно записать в виде системы  Исходя из приведенной системы, необходимо решить две системы с треугольными матрицами коэффициентов.

Рассматривая первое уравнение, получим формулы для вычисления вспомогательного вектора .

(1.9)

Все  могут быть последовательно найдены по формуле

(1.10)

Рассмотрим второе уравнение системы

(1.11)

 Отсюда значения неизвестных находятся в обратном порядке по формуле

(1.12)

Вычисление определителя LU-факторизованной матрицы A опирается на свойство определителя произведения матриц и сводится к перемножению n чисел

(1.13)

1.3 Решение СЛАУ методом Гаусса

Рассмотрим один из наиболее известных и широко применяемых прямых методов решения систем линейных уравнений. Обычно этот метод называют методом исключения или методом Гаусса.

Если задана некоторая произвольная система уравнений, то без предварительного исследования нельзя сказать, имеет ли она какое–либо решение и, в случае если решение существует, является ли оно единственным. На этот вопрос существуют три и только три ответа:

* решение системы уравнений существует и является единственным;
* система уравнений вообще не имеет решения;
* система уравнений имеет бесконечное множество решений.

В методе Гаусса матрица СЛАУ с помощью равносильных преобразований преобразуется в верхнюю треугольную матрицу, получающуюся в результате прямого хода.

В обратном ходе определяются неизвестные [4].

Пусть дана СЛАУ



(1.14)

Запишем расширенную матрицу системы:



(1.15)

На первом шаге алгоритма Гаусса выберем диагональный элемент (если он равен 0, то первую строку переставляем с какой–либо нижележащей строкой) и объявляем его ведущим, а соответствующую строку и столбец, на пересечении которых он стоит – ведущими. Обнулим элементы a21, …, an1 ведущего столбца. Для этого сформируем числа . Умножая ведущую строку на число , складывая со второй и ставя результат на место второй строки, получим вместо элемента a21 нуль, а вместо элементов , b2 – соответственно элементы  ,  и т.д. Умножая ведущую строку на число , складывая с n–ой строкой и ставя результат на место n–ой строки, получим вместо элемента an1 нуль, а остальные элементы этой строки будут иметь вид: . Сохраняя ведущую строку неизменной, получим в результате 1–го шага алгоритма Гаусса следующую матрицу:

 (1.16)

На втором шаге алгоритма Гаусса в качестве ведущего элемента выбирается элемент  (если он равен нулю, то вторую строку взаимно меняем на нижележащую строку). Формируются числа , которые ставятся около ведущей строки. Умножая ведущую строку на число  и складывая результат с третьей строкой, получим вместо элемента  нуль, а вместо элементов , ,  – элементы , , и так далее. Умножая ведущую строку на число , складывая результат с n–ой строкой и ставя полученную сумму на место n–ой строки, получим вместо элемента  нуль, а вместо элементов , ,  – элементы , . Сохраняя 1–ую и 2–ую строки матрицы неизменными, получим в результате второго шага алгоритма

Гаусса следующую матрицу:



(1.17)

После (*n - 1*) – го шага алгоритма Гаусса получаем следующую расширенную матрицу, содержащую верхнюю треугольную матрицу СЛАУ:



(1.18)

Прямой ход алгоритма Гаусса завершен.

В обратном ходе алгоритма Гаусса из последнего уравнения сразу определяется xn, из предпоследнего – xn–1 и т.д. Из первого уравнения определяется x1.



(1.19)

Если элементы какой–либо строки матрицы системы в результате преобразований стали равными нулю, а правая часть не равна нулю, то СЛАУ несовместна, поскольку не выполняются условия теоремы Кронекера–Капелли.

Если элементы какой–либо строки матрицы системы и правая часть в результате преобразований стали равными нулю, то СЛАУ совместна, но имеет бесконечное множество решений, получающееся с помощью метода Гаусса для СЛАУ порядка r, где r – ранг матрицы исходной СЛАУ.

В результате прямого хода метода Гаусса можно вычислить определитель матрицы A исходной СЛАУ:

 (1.20)

При этом с помощью множителя , где p – число перестановок строк в процессе прямого хода, учитываются соответствующие перемены знаков вследствие перестановок строк.

Метод Гаусса можно применить для обращения невырожденной () матрицы.

Действительно, пусть требуется обратить невырожденную матрицу , . Тогда, сделав обозначение , , , можно выписать матричное уравнение AX = E, где E – единичная матрица

 (1.21)

на основе которого можно записать цепочку СЛАУ

, , … (1.22)

каждую из которых можно решить методом Гаусса. При этом, поскольку верхняя треугольная матрица для всех этих СЛАУ будет одной и то же, то метод Гаусса применяется один раз. Строится следующая расширенная матрица:



(1.23)

В результате применения (n – 1)-го шага метода Гаусса получаем:



(1.24)

При этом первый столбец  обратной матрицы определяется в обратном ходе метода Гаусса с правой частью b1, столбец  – с правой частью b2 и так далее. Столбец  определяется с правой частью bn.

1.4 Технология OpenMP

OpenMP (Open Multi-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMPсистемах).

Первый стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание легко переносимых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и языки Си и Си++.

Интерфейс OpenMP стал одной из наиболее популярных технологий параллельного программирования. OpenMP успешно используется как при программировании суперкомпьютерных систем с большим количеством процессоров, так и в настольных пользовательских системах или, например, в Xbox 360.

Разработку спецификации OpenMP ведут несколько крупных производителей вычислительной техники и программного обеспечения, чья работа регулируется некоммерческой организацией "OpenMP Architecture Review Board" (ARB) [3]. В OpenMP используется модель параллельного выполнения "ветвление-слияние". Программа OpenMP начинается как единственный поток выполнения, называемый начальным потоком. Когда поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и некоторого числа дополнительных потоков, и становится главным в новой группе. Все члены новой группы (включая главный) выполняют код внутри параллельной конструкции. В конце параллельной конструкции имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главный поток. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

Число потоков в группе, выполняющихся параллельно, можно контролировать несколькими способами. Один из них - использование переменной окружения OMP\_NUM\_THREADS. Другой способ - вызов процедуры omp\_set\_num\_threads(). Еще один способ - использование выражения num\_threads в сочетании с директивой parallel.

Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

Все переменные, созданные до директивы parallel являются общими для всех потоков. Переменные, созданные внутри потока, являются локальными (приватными) и доступны только текущему потоку. При изменении общей переменной одновременно несколькими потоками возникает состояние гонок (невозможно гарантировать какой-либо конкретный порядок записи и, следовательно, результат) — это проблема и допускать такое нельзя. Такая же проблема возникает, когда один поток пытается читать переменную в то время, как другой ее изменяет.

Для решения проблемы существует директива critical. Разделяемым ресурсом в этом примере является не только память (размещенные в ней переменные), но и консоль. В критической секции в один момент времени может находиться только один поток, остальные ожидают ее освобождения. Правильным считается, если критическая секция содержит обращения только к одному разделяемому ресурсу.

Для ряда операций более эффективно использовать директиву atomic, чем критическую секцию. Она ведет себя также, но работает быстрее.

Самый популярный способ распределения задач в OpenMP — параллельный цикл. Программы почти всю свою жизнь проводят, выполняя циклы, при этом если между итерациями цикла нет зависимостей — то цикл называется векторизуемым (его итерации можно поделить между потоками и выполнить независимо друг от друга).

Параллельный цикл позволяет задать опцию schedule, изменяющую алгоритм распределения итераций между потоками. Всего поддерживается 3 таких алгоритма. Далее полагаем, что у нас p потоков выполняют n итераций:

Опции планирования:

* schedule (static) — статическое планирование. При использовании такой опции итерации цикла будут поровну (приблизительно) поделены между потоками. Нулевой поток получит первые np итераций, первый — вторые и т.д.;
* schedule (static, 10) — блочно-циклическое распределение итераций. Каждый поток получает заданное число итераций в начале цикла, затем (если остались итерации) процедура распределения продолжается. Планирование выполняется один раз, при этом каждый поток «узнает» итерации, которые должен выполнить;
* schedule (dinamic), schedule (dynamic, 10) — динамическое планирование. По умолчанию параметр опции равен 1. Каждый поток получает заданное число итераций, выполняет их и запрашивает новую порцию. В отличии от статического планирования, выполняется многократно (во время выполнения программы). Конкретное распределение итераций между потоками зависит от темпов работы потоков и трудоемкости итераций;
* schedule (guided), schedule (guided, 10) — разновидность динамического планирования с изменяемым при каждом последующем распределении числе итераций. Распределение начинается с некоторого начального размера, зависящего от реализации библиотеки до значения, задаваемого в опции (по умолчанию 1). Размер выделяемой порции зависит от количества еще нераспределенных итераций.

В большинстве случаев самым оптимальным вариантом является static, так как выполняет распределение единственный раз [5].

Параллельную обработку в OpenMP иллюстрирует рисунок 1.1. Самая левая стрелка представляет основной поток, который выполняется в одиночестве, пока не достигает первого параллельного региона в точке 1. В этой точке основной поток создает группу потоков, и теперь все они одновременно выполняются в параллельном регионе.

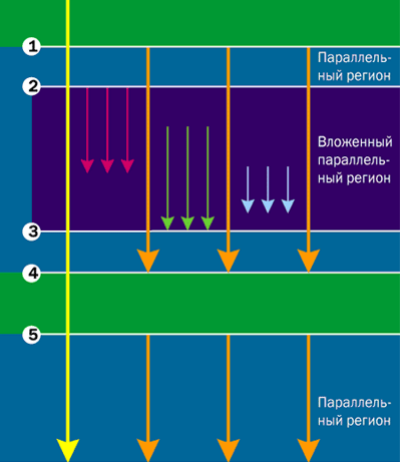


Рисунок 1.1 – Параллельные разделы OpenMP

В точке 2 три из этих четырех потоков, достигнув вложенного параллельного региона, создают новые группы потоков. Исходный основной и потоки, создавшие новые группы, становятся владельцами своих групп (основными в этих группах). Учтите, что потоки могут создавать новые группы в разные моменты или вообще не встретить вложенный параллельный регион.

В точке 3 вложенный параллельный регион завершается. Каждый поток вложенного параллельного региона синхронизирует свое состояние с другими потоками в этом регионе, но синхронизация разных регионов между собой не выполняется.

В точке 4 заканчивается первый параллельный регион, а в точке 5 начинается новый. Локальные данные каждого потока в промежутках между параллельными регионами сохраняются.

**1.5 GDI графический интерфейс**

GDI (Graphics Device Interface, Graphical Device Interface) — один из трёх основных компонентов или «подсистем», вместе с ядром и Windows API, составляющих пользовательский интерфейс (оконный менеджер GDI) Microsoft Windows.

Для определения атрибутов текста и изображения, которые выводятся на экран или принтер, используется программный объект под названием «контекст устройства» (Device Context, DC). DC, как и большинство объектов GDI, инкапсулирует подробности реализации и данные в себе и к ним нельзя получить прямой доступ.

Для любого рисования нужен объект HDC (Handle DC). При выводе на принтер HDC получается вызовом CreateDC, и на нём вызываются специальные функции для перехода на новую страницу печатаемого документа. При выводе на экран также можно использовать CreateDC, но это приведет к рисованию поверх всех окон вне их границ, потому обычно для рисования на экране используются вызовы GetDC и BeginPaint, принадлежащие уже не GDI, а USER, и возвращающие контекст, ссылающийся на регион отсечения окна [2].

***1.5.1*** Все Windows-программы начинают выполнение с вызова функции WinMain().

Все Windows-программы должны содержать специальную функцию, которая не используется в самой программе, но вызывается самой операционной системой. Эту функцию обычно называют функцией окна, или процедурой окна. Она вызывается Windows, когда системе необходимо передать сообщение в программу.

Именно через нее осуществляется взаимодействие между программой и системой. Функция окна передает сообщение в своих аргументах. Согласно терминологии Windows, функции, вызываемые системой, называются функциями обратного вызова. Таким образом, функция окна является функцией обратного вызова.

Помимо принятия сообщения от Windows, функция окна должна вызывать выполнение действия, указанного в сообщении.

Конечно, программа не обязана отвечать на все сообщения, посылаемые Windows. Поскольку их могут быть сотни, то большинство сообщений обычно обрабатывается самой системой, а программе достаточно поручить Windows выполнить действия, предусмотренные по умолчанию.

В большинстве Windows-программ задача создания функции окна лежит на программисте. Но в любом случае, если сообщение получено, то программа должна выполнить некоторое действие.

Хотя она может вызывать для этого одну или несколько API-функций, само действие было инициировано Windows.

Поэтому именно способ взаимодействия с операционной системой через сообщения диктует общий принцип построения всех программ для Windows.

***1.5.2*** Как объяснялось выше, Windows взаимодействует с программой, посылая ей сообщения. Все приложения Windows должны организовать так называемый цикл сообщений (обычно внутри функции WinMain()). В этом цикле каждое необработанное сообщение должно быть извлечено из очереди сообщений данного приложения и передано назад в Windows, которая затем вызывает функцию окна программы с данным сообщением в качестве аргумента. В традиционных Windows-программах необходимо самостоятельно создавать и активизировать такой цикл. Он является неотъемлемой частью любого приложения Windows. Процесс получения и обработки сообщений может показаться чересчур сложным, но тем не менее ему должны следовать все Windows-программы.

***1.5.3*** Каждое окно в Windows приложении характеризуется определенными атрибутами, называемыми классом окна. (Здесь понятие «класс» не идентично используемому в С++. Оно, скорее, означает стиль или тип.) В традиционной программе класс окна должен быть определен и зарегистрирован прежде, чем будет создано окно. При регистрации необходимо сообщить Windows, какой вид должно иметь окно и какую функцию оно выполняет. В то же время регистрация класса окна еще не означает создание самого окна. Для этого требуется выполнить дополнительные действия.

***1.5.4*** Windows является системой, не зависящей от устройств (device independent). Эту независимость со стороны Windows обеспечивает библиотека GDI32.dll, а со стороны устройства – драйвер этого устройства. С точки зрения программы связующим звеном между программой и устройством является контекст устройства (Device Context – DC). Если программе нужно осуществить обмен с внешним устройством, программа должна оповесить GDI о необходимости подготовить устройство для операции ввода-вывода. После того, как устройство подготовленно, программа получает хэндл контекста устройства, т.е. хэндл структуры, содержащей набор характеристик этого устройства. Программа никогда напрямую не обращается к контексту устройства, она обращается к нему опосредствованно, через определенные функции. После того, как все действия произведены, и необходимость в использовании устройства отпала, программа должна освободить контекст устройства, чтобы не занимать память.

Когда программа требует контекст устройства, она получает его уже заполненным значениями по умолчанию. Объект в составе контекста называется текущим объектом. Само слово - текущий - говорит о том, что контекст устройства можно изменить. Замещенный объект автоматически из памяти не удаляется, его необходимо позже удалить отдельно.

В Windows поддерживаются несколько типов контекстов устройств. Одним из них является контекст дисплея.

Windows поддерживает три типа контекста дисплея – контекст класса, приватный контекст и общий контекст. Первые два типа используются в приложениях, которые выводят на экран большое количество информации. Ярким примером такого рода приложений являются настольные издательские системы, графические пакеты и т.д.

Контексты устройств хранятся в кэше, управляемом системой. Хэндл общего контекста программа получает с помощью функций GetDC(), GetDCEx() или BeginPaint(). После того, как программа отработает с дисплеем, она должна освободить контекст, вызвав функцию ReleaseDC() или EndPaint() (в случае, если контекст получался с помощью BeginPaint()). После того, как контекст дисплея освобожден, все изменения, внесенные в него программой.

***1.5.5*** Отрисовка графика производится стандартными средствами WinAPI, а именно с помощью GDI. Для вывода графической информации с помощью данной подсистемы был использован следующий набор функций:

LineTo(HDC hdc, int nXEnd, int nYEnd) – функция рисует линию из текущей позиции до точки с координатами (nXEnd, nYEnd).

MoveTo(HDC hdc, int X, int Y, LPPOINT lppoint) – фукнция устанавливает указатель в позицию с координатами (X, Y).

TextOut(HDC hdc, int nXStart, int nYStart, LPCTSTR lpString, int cchString) – функция выводит текстовую информацию lpString длиной cchString в точку с координатами (nXStart, nYStart)

Работа с графическими объектами производится с помощью их дескрипторов (handles) - HDC, HPEN, HBRUSH, HFONT и т.д. Создание и удаление объектов производится с помощью соответствующих функций - например, объект pen создается с помощью CreatePen, удаляется с помощью DeleteObject. Режимы, задающиеся через графические объекты, переключаются с помощью создания новых объектов и указания контексту (DC) использовать их для вывода графики.

**2 ПОСТАНОВКА ЗАДАЧИ**

Необходимо разработать программу, осуществляющую распределённое решение СЛАУ средствами OpenMP на основе метода LU – разложения, где L – нижнетреугольная матрица, U – верхнетреугольная матрица с единицами на главной диагонали. Предусмотреть возможность ввода исходных данных из текстового файла. Максимальное количество неизвестных – 50000. Сравнить реализованный вариант метода по скорости нахождения с его линейным вариантом и с распределенным вариантом метода Гаусса.

Исходными данными являются:

– матрица А;

– матрица B.

Исходные данные должны удовлетворять следующим условиям:  
– матрица A квадратная;

– количество строк матриц A и B одинаково;

– матрица A должна быть невырожденной.

Для выполнения поставленной задачи, программа должна реализовать следующий функционал:

– ввод исходных данных из файла;

– генерирование матрицы со случайным значениями;

– вычисление системы распределенным и линейным LU-разложением и распределенным методом Гаусса;

– вывод результатов вычисления и времени выполнения.

**3 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ**

3.1 Анализ исходных данных и результатов

Необходимо разработать программу, осуществляющую распределённое решение СЛАУ средствами OpenMP на основе метода LU – разложения, где L – нижнетреугольная матрица, U – верхнетреугольная матрица с единицами на главной диагонали. Предусмотреть возможность ввода исходных данных из текстового файла. Максимальное количество неизвестных – 50000. Сравнить реализованный вариант метода по скорости нахождения с его линейным вариантом и с распределенным вариантом метода Гаусса.

Основываясь на функционале приложения, необходимо составить алгоритм обработки исходных данных, то есть процесс выполнения приложения.

Схема программы показана на рисунке 3.1.



Рисунок 3.1 – Схема программы

**3.2 Алгоритм решения системы линейных алгебраических уравнений методом -разложения**

LU-разложение используется для решения систем линейных уравнений, обращения матриц и вычисления определителя. LU-разложение существует только в том случае, когда матрица обратима, а все ведущие (угловые) главные миноры матрицы невырождены [1].

Этот метод является одной из разновидностей [метода Гаусса](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0).

Алгоритм LU-разложения можно разделить на следующие части:

* нахождение матриц L и U;
* вычисление элементов матрицы;
* проверка на равенство произведения матриц L и U и матрицы A.

Блок схема алгоритма приведена в приложении Б на рисунке Б.1.

3.3 Алгоритм решения системы линейных алгебраических уравнений методом Гаусса

Для решения системы уравнений методом Гаусса необходимо преобразовать матрицу к верхнетреугольной. Это достигается путем проведения ряда операций над матрицей. Сначала происходит поиск строки с максимальным коэффициентом рассматриваемой неизвестной. Затем найденная строка меняется местами с текущей строкой. После этого матрицы и преобразуются по определенным формулам. Вышеописанная часть алгоритма Гаусса называется прямым ходом.

Остальная часть алгоритма называется обратным ходом. Она заключается в нахождении окончательного решения системы, начиная с последней неизвестной, так как именно после преобразований, последняя строка матрицы содержит единственную неизвестную, которую можно вычислить. После этого находятся оставшиеся неизвестные.

Соответствующая блок-схема изображена в приложении Б на рисунке Б.2.

**4 РАЗРАБОТКА ПРОГРАММНОГО КОДА**

Все функции располагаются в файлах:

* CourseProject.cpp;
* Matrix.cpp;
* SLAU.cpp.

В файле CourseProject.cpp описан процесс создания, отображения окна, обработки событий. Список функций и их описание отображено в таблице 4.1.

Таблица 4.1 – Функции файла CourseProject.cpp

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| wWinMain | Точка входа в приложение |
| WndProc | Регистрация класса окна |
| Calculate | Вычисление результата |
| CheckMatrix | Проверка матрицы на квадратность и выявление нулевых значений |

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) Функция принимает параметры:

* HWND hWnd – дескриптор окна, которому адресовано сообщение;
* UINT message – идентификатор сообщения;
* WPARAM wParam – дополнительная информация;
* LPARAM lParam – дополнительная информация.

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow). Данная функция содержит следующие параметры:

* HINSTANCE hInstance – указатель на текущий экземпляр;
* HINSTANCE hPrevInstance – указатель на предыдущий запущенный экземпляр;
* LPSTR lpCmdLine – командная строка;
* int nCmdShow – тип отображения окна программы.

void Calculate(). Данная функция не содержит параметров.

bool CheckMatrix(SLAU slau, int n). Данная функция содержит следующие параметры:

* SLAU slau – система линейных уравнений;
* int n – размерность матриц.

В файле Matrix.cpp описан основной алгоритм приложения для работы с матрицами. Список функций и их описание отображено в таблице 4.2.

Таблица 4.2 – Функции файла Matrix.cpp

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| LU | Реализация линейного метода LU-разложения |
| LUPar | Реализация распределенного метода LU-разложения |
| GaussPar | Реализация распределенного метода Гаусса |
| CopyMatrix | Копирование исходных матриц |

Функция static double\* LU(SLAU slau, int n) необходима для решения СЛАУ линейным методом LU-разложения и принимает следующие параметры:

* SLAU slau – система уравнений;
* int n – размерность матрицы.

Функция static double\* LUPar(SLAU slau, int n, int thread) необходима для решения СЛАУ распределенным методом LU-разложения и принимает параметры:

* SLAU slau – система уравнений;
* int n – размерность матрицы;
* int thread – количество потоков.

Функция static double\* GaussPar(SLAU slau, int n, int thread) необходима для решения СЛАУ распределенным методом Гаусса и принимает параметры:

* SLAU slau – система уравнений;
* int n – размерность матрицы;
* int thread – количество потоков.

Функция static void CopyMatrix(double\*\* A, double\* B, double\*\* newA, double\* newB, int n) необходима для копирования СЛАУ во избежание получения неверных значений в матрицах и принимает следующие параметры:

* double\*\* A – матрица коэффициентов;
* double\* B – столбец свободных членов;
* double\*\* newA – новая матрица коэффициентов;
* double\* newB – новый столбец свободных членов;
* int n – размерность матриц.

Класс SLAU содержит следующие поля:

* А – матрица коэффициентов;
* В – столбец свободных членов;
* Х – результат.

**5 РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА**

Программа состоит из одного графического окна. В данном окне имеется ряд полей ввода информации, а именно – поле ввода количества потоков, и поле ввода размерности матрицы Также присутствуют элементы управления программой в виде кнопок – две кнопки для того, чтобы решить СЛАУ, составленную из случайных значений, и считанную из файла.

Элементы управления создаются в обработчике сообщения WM\_CREATE, при помощи команды CreateWindow. Данная функция принимает следующие параметры:

* LPCTSTR lpClassName – указатель на зарегистрированное имя класса;
* LPCTSTR lpWindowName – указатель на имя окна;
* DWORD dwStyle – стиль окна;
* int x – горизонтальная позиция окна;
* int y – вертикальная позиция окна;
* int nWidth – ширина окна;
* int nHeight – высота окна;
* HWND hWndParent – дескриптор родительского или окна владельца;
* HMENU hMenu – дескриптор меню или идентификатор дочернего окна;
* HANDLE hInstance – дескриптор экземпляра приложения;
* LPVOID lpParam – указатель на данные создания окна.

С помощью данной функции мы создаём:

* Три листбокса с результатами выполнения вычислений различными методами;
* Три текстовых поля, содержащих названия методов вычислений;
* Три текстовых поля, содержащих время выполнения методов вычислений;
* Поля ввода количества потоков и размерности матрицы;
* Набор кнопок для выбора, создать матрицы случайными значениями либо считать из файлов.

**6 АНАЛИЗ РЕЗУЛЬТАТОВ РАБОТЫ И ВЫВОДЫ**

В качестве используемого оборудования выступает компьютер со следующими характеристиками:

* процессор Intel(R) Core(TM) i5-4288U CPU @ 2.60GHz 2.61GHz;
* ОС Windows 10;
* 4.00 ГБ ОЗУ.

Процессор имеет 4 ядра, что позволит использовать распределенные вычисления.

В ходе анализа выполнения вычислений линейным методом LU-разложения, распределенным методом LU-разложения и распределенным методом Гаусса была получена статистика, анализируя которую, можно сказать, что:

* с увеличением размера матрицы растет время вычисления;
* с увеличением числа потоков время вычисления уменьшается;
* быстрее всего вычисления выполняются распределенным методом Гаусса.

Далее приведена подробная статистическая информация.

Рассмотрим зависимость скорости выполнения вычислений от размерности матрицы для каждого метода. (таблица 6.1).

Таблица 6.1 – Время выполнения вычислений

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер** | **Метод LU-разложения (линейный)** | **Метод LU-разложения (параллельный)** | **М-д Гаусса (параллельный)** |
| 100 | 0,002669 | 0,004074 | 0,0014049 |
| 200 | 0,021226 | 0,018194 | 0,0098636 |
| 300 | 0,063852 | 0,062336 | 0,0321451 |
| 400 | 0,151832 | 0,150978 | 0,0775244 |

Проанализировав данные в таблице, можно заметить, что с увеличением размерности матрицы растет время выполнения вычислений.

Также, исходя из результатов наблюдений, вычисления распределенным методом Гаусса происходят значительно быстрее, нежели двумя другими методами.

График зависимости, расположенный на рисунке 6.1, наглядно иллюстрирует это.

Рисунок 6.1 – График зависимости скорости выполнения вычислений от размера матрицы

Рассмотрим зависимость скорости выполнения вычислений от количества потоков (таблица 6.2, рисунки 6.2 и 6.3). Размерность матрицы – 1000 х 1000.

Таблица 6.2 – Время выполнения при различном количестве потоков

|  |  |  |
| --- | --- | --- |
| **Количество потоков** | **Метод LU-разложения (параллельный)** | **Метод Гаусса (параллельный)** |
| 10 | 4,29317 | 1,26519 |
| 20 | 4,37574 | 1,26252 |
| 30 | 4,3902 | 1,38095 |
| 40 | 4,34065 | 1,31162 |
| 50 | 4,5142 | 1,26837 |
| 60 | 4,31689 | 1,26176 |
| 70 | 4,36191 | 1,26653 |
| 80 | 4,43059 | 1,25903 |
| 90 | 4,34729 | 1,2795 |
| 100 | 4,45273 | 1,27753 |

Рисунок 6.2 – Зависимость скорости вычислений от количества потоков

Рисунок 6.3 – Зависимость скорости вычислений от количества потоков

Рассмотрим зависимость скорости выполнения вычислений от выбранного метода вычислений. Результат анализа представлен на рисунке 6.4 и в таблице 6.3. Размерность матрицы – 1000 х 1000.

Таблица 6.3 – Время выполнения вычислений различными методами

|  |  |
| --- | --- |
| **Метод** | **Метод LU-разложения (параллельный)** |
| LU- разложение (линейный) | 50,1341 |
| LU-разложение (параллельный) | 49,2767 |
| Метод Гаусса (параллельный) | 9,94321 |

Рисунок 6.4 – Зависимость скорости вычислений от метода

**7 ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ**

**7.1 Тестирование пользовательского интерфейса**

Весь необходимый функционал приложения был реализован с помощью графических элементов. Все действия сопровождаются соответствующими уведомлениями.

Для каждого уведомления был подобран соответствующий дизайн: шрифт, цвет и содержание. Для приложения были подобраны стили, которые делают приложение более привлекательным, но не мешают читабельности.

Приложение имеет удобный интерфейс. Все элементы выполнены в одном стиле.

Отсутствуют орфографические и пунктуационные ошибки.

При запуске программы открывается окно, которое представлено на рисунке 7.1:

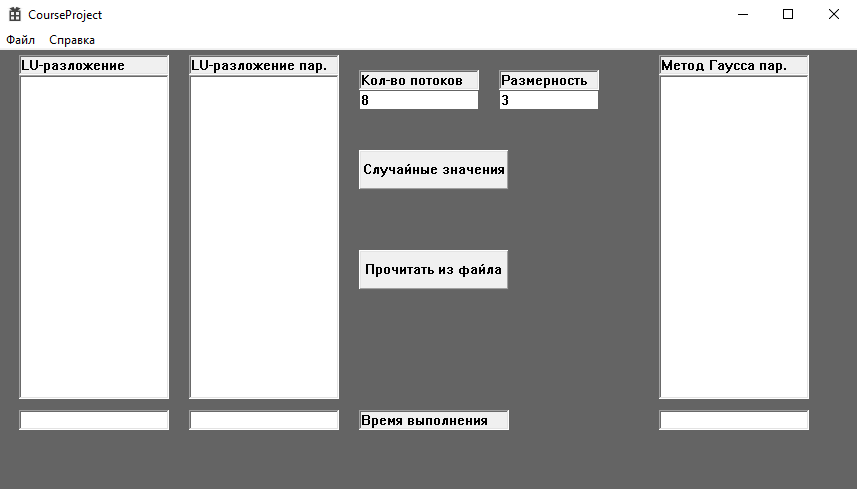


Рисунок 7.1 – Главное окно программы

В этом окне пользователь может ввести количество потоков для распределенного вычисления, размерность матрицы, которая будет сгенерирована случайно.

Если же пользователь нажмет на кнопку «Случайные значения», будет произведено решение системы линейных алгебраических уравнений по коэффициентам, сгенерированным случайным образом.

При нажатии на кнопку «Прочитать из файла», в программу загружаются данные из текстовых файлов. Если при считывании возникают исключения (отсутствует файл и т.п.), пользователь получает уведомление об этом. Если файл был считан успешно, также появляется уведомление, сообщающее об этом (рисунок 7.2).

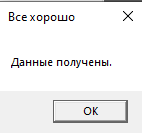


Рисунок 7.2 – Уведомление об удачном считывании файла

Если по случайности пользователь вводит неверные значения в поля ввода количества потоков или размерности матрицы, он получает уведомление об этом с кнопкой «ОК», при нажатии на которую пользователь может вновь ввести желаемые значения. Пример уведомления представлен на рисунке 7.3.

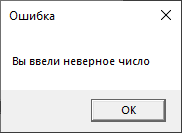


Рисунок 7.3 – Уведомление о вводе неверных данных

Таблица 7.1 – Тестирование пользовательского интерфейса

|  |  |  |
| --- | --- | --- |
| **№** | **Вид проверки** | **Результат** |
| 1 | Реализуется ли функционал приложения при помощи графических элементов | 10 |
| 2 | Реализуется ли размещение сообщений об ошибках | 10 |
| 3 | Отсутствуют ли орфографически, пунктуационные ошибки | 10 |
| 4 | Читабелен ли используемый шрифт | 9 |

**7.2 Тестирование функционала**

Результат тестирования функционала программы представлен в таблице 7.2.

Таблица 7.2 – Тестирование функционала

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Имя элемента** | **Тест** | **Результат** |
| 1 | LU | Решение СЛАУ с помощью метода LU-разложения | Исключительных ситуаций нет |
| 2 | LUPar | Решение с помощью распределенного LU-разложения | Исключительных ситуаций нет |
| 3 | GaussPar | Решение с помощью распределенного метода Гаусса | Исключительных ситуаций нет |

Также было проведено Unit-тестирование программы, результат которого представлен в таблице 7.3 и на рисунке 7.3.

Таблица 7.3 – Тестирование функционала

|  |  |  |
| --- | --- | --- |
| **№** | **Название функции** | **Результат** |
| 1 | TEST\_METHOD(TestLUTrue) | Проверка верности результата для метода LU, при верно заданной матрице |
| 2 | TEST\_METHOD(TestLUFalse) | Проверка верности результата для метода LU, при неверно заданной матрице |
| 3 | TEST\_METHOD(TestLUParTrue) | Проверка верности результата для метода LUPar, при верно заданной матрице |
| 4 | TEST\_METHOD(TestLUParFalse) | Проверка верности результата для метода LUPar, при неверно заданной матрице |
| 5 | TEST\_METHOD(TestGaussParTrue) | Проверка верности результата для метода GaussPar, при верно заданной матрице |
| 6 | TEST\_METHOD(TestGaussParFalse) | Проверка верности результата для метода GaussPar, при неверно заданной матрице |
| 7 | TEST\_METHOD(CheckZeroLine) | Проверка решения системы при наличии нулевой строки в матрице |

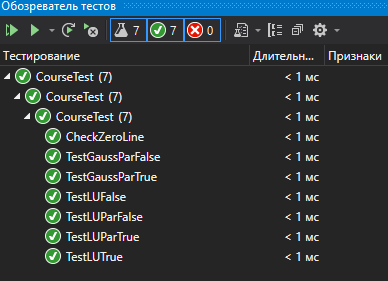


Рисунок 7.3 – Результат выполнения тестов

**ЗАКЛЮЧЕНИЕ**

В результате выполнения данного курсового проекта была разработана программа, предназначенная для решения системы линейных алгебраических уравнений с использованием средств OpenMP.

Поставленная задача решена на языке программирования C++ в среде разработки Microsoft Visual Studio Community 2019.

В ходе реализации проекта были разобраны следующие темы:

– решение СЛАУ линейным методом LU-разложения;

– решение СЛАУ распределенным методом LU-разложения;

– решение СЛАУ распределенным методом Гаусса.

Получены характеристики скорости вычисления, на основании которого сделан следующий вывод об эффективности распараллеливания.

Поставленные задачи в курсовом проекте решены в полном объеме.

**Список использованных источников**

* 1. LU-разложение [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/LU-разложение – Дата доступа: 04.12.2019
  2. Фролов, Г.В. Библиотека системного программиста. Том 14. Графический интерфейс GDI в Microsoft Windows / Г. В. Фролов. – М.: Диалог-МИФИ, 2009. – 288 с.
  3. Гергель В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие / Гергель В.П., Стронгин, Р.Г. – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. 184 с.
  4. Численные методы решения СЛАУ [Электронный ресурс]. – Режим доступа: <http://mathhelpplanet.com/static.php?p=chislennyye-metody-resheniya-slau> – Дата доступа: 26.11.2019
  5. Учебник по OpenMP [Электронный ресурс]. – Режим доступа: <https://pro-prof.com/archives/4335> – Дата доступа: 04.12.2019

**Приложение А**

**(обязательное)**

**Matrix.cpp**

#include "Matrix.h"

#include <math.h>

#include <list>

#include <iostream>

#include <omp.h>

using namespace std;

class Matrix {

public:

static double\* LU(SLAU slau, int n) {

double\*\* a = new double\* [n];

for (int i = 0; i < n; i++) a[i] = new double[n];

double\* b = new double[n];

CopyMatrix(slau.A, slau.B, a, b, n);

double\*\* L = new double\* [n];

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

L[i] = new double[n];

double\*\* U = new double\* [n];

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

U[i] = new double[n];

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

{

if (rand() % 2)

b[i] = 1 - (rand() % 10);

else

b[i] = 1 + (rand() % 10);

}

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

{

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

{

L[i][j] = 0;

U[i][j] = 0;

}

}

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

{

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

{

if (j < i) L[j][i] = 0;

else

{

L[j][i] = a[j][i];

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

L[j][i] += -L[j][k] \* U[k][i];

}

}

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

{

if (j < i) U[i][j] = 0;

else if (j == i) U[i][j] = 1;

else

{

U[i][j] = a[i][j] / L[i][i];

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

U[i][j] += -((L[i][k] \* U[k][j]) / L[i][i]);

}

}

}

double\* result = new double[n];

double\* y = new double[n];

double tempVar = 0;

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

{

if (i == 0)

{

y[0] = b[0] / L[0][0];

continue;

}

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

tempVar += y[j] \* L[i][j];

y[i] = (b[i] - tempVar) / L[i][i];

tempVar = 0;

}

tempVar = 0;

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

{

if (i == n - 1)

{

result[n - 1] = y[n - 1] / U[n - 1][n - 1];

continue;

}

for (int j = n - 1; j > i; j--)

tempVar += result[j] \* U[i][j];

result[i] = (y[i] - tempVar) / U[i][i];

tempVar = 0;

}

return result;

}

static double\* LUPar(SLAU slau, int n, int thread)

{

double\*\* a = new double\* [n];

for (int i = 0; i < n; i++) a[i] = new double[n];

double\* b = new double[n];

CopyMatrix(slau.A, slau.B, a, b, n);

double\*\* L = new double\* [n];

for (int i = 0; i < n; i++) L[i] = new double[n];

double\*\* U = new double\* [n];

for (int i = 0; i < n; i++) U[i] = new double[n];

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

{

if (rand() % 2)

b[i] = 1 - (rand() % 10);

else

b[i] = 1 + (rand() % 10);

}

#pragma omp parallel for private(i,j)

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

{

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

{

L[i][j] = 0;

U[i][j] = 0;

}

}

omp\_set\_num\_threads(thread);

#pragma omp parallel for shared(U,n,k) private(i) schedule(static, 64)

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

{

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

{

if (j < i) L[j][i] = 0;

else

{

L[j][i] = a[j][i];

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

L[j][i] += -L[j][k] \* U[k][i];

}

}

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

{

if (j < i) U[i][j] = 0;

else if (j == i) U[i][j] = 1;

else

{

U[i][j] = a[i][j] / L[i][i];

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

U[i][j] += -((L[i][k] \* U[k][j]) / L[i][i]);

}

}

}

double\* result = new double[n];

double\* y = new double[n];

double tempVar = 0;

#pragma omp parallel for private(i)

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

{

if (i == 0)

{

y[0] = b[0] / L[0][0];

continue;

}

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

tempVar += y[j] \* L[i][j];

y[i] = (b[i] - tempVar) / L[i][i];

tempVar = 0;

}

tempVar = 0;

#pragma omp parallel for private(i,n)

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

{

if (i == n - 1)

{

result[n - 1] = y[n - 1] / U[n - 1][n - 1];

continue;

}

for (int j = n - 1; j > i; j--)

tempVar += result[j] \* U[i][j];

result[i] = (y[i] - tempVar) / U[i][i];

tempVar = 0;

}

return result;

}

static double\* GaussPar(SLAU slau, int n, int thread)

{

double\*\* a = new double\* [n];

for (int i = 0; i < n; i++) a[i] = new double[n];

double\* b = new double[n];

CopyMatrix(slau.A, slau.B, a, b, n);

double\*\* matrixSLAU = new double\* [n];

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

{

matrixSLAU[i] = new double[n + 1];

}

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

{

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

{

matrixSLAU[i][j] = a[i][j];

}

matrixSLAU[i][n] = b[i];

}

omp\_set\_num\_threads(thread);

#pragma omp parallel for private(i,n)

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

{

double maxItem = abs(matrixSLAU[i][i]);

int maxRow = i;

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

{

if (abs(matrixSLAU[k][i]) > maxItem)

{

maxItem = abs(matrixSLAU[k][i]);

maxRow = k;

}

}

#pragma omp parallel for private(k)

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

{

double tmp = matrixSLAU[maxRow][k];

matrixSLAU[maxRow][k] = matrixSLAU[i][k];

matrixSLAU[i][k] = tmp;

}

#pragma omp parallel for private(k, j)

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

{

double c = -matrixSLAU[k][i] / matrixSLAU[i][i];

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

{

if (i == j)

{

matrixSLAU[k][j] = 0;

}

else

{

matrixSLAU[k][j] += c \* matrixSLAU[i][j];

}

}

}

}

double\* result = new double[n];

#pragma omp parallel for private(i,n,k)

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

{

result[i] = matrixSLAU[i][n] / matrixSLAU[i][i];

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

{

matrixSLAU[k][n] -= matrixSLAU[k][i] \* result[i];

}

}

return result;

}

static void CopyMatrix(double\*\* A, double\* B, double\*\* newA, double\* newB, int n) {

for (int i = 0; i < n; i++) {

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

{

newA[i][j] = A[i][j];

}

newB[i] = B[i];

}

}

};

**CourseProject.cpp**

#include "framework.h"

#include "CourseProject.h"

#include <string>

#include <sstream>

#include <fstream>

#include <iostream>

#include <OleCtl.h>

#include <tchar.h>

#include "atlstr.h"

#include <tlhelp32.h>

#include <iostream>

#include <regex>

#include <Windows.h>

#include <ctime>

//#include <omp.h>

#define MAX\_LOADSTRING 100

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

static HWND hRangeTextBox, luList, luParList, gaussParList, timeLU, timeLUParallel, timeGaussParallel, hThreadTextBox;

static SLAU slau;

double\* xLu;

double\* xLuPar;

double\* xGaussPar;

static int n;

static int thread;

void Calculate();

bool CheckMatrix(SLAU slau, int n);

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_COURSEPROJECT, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_COURSEPROJECT));

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_COURSEPROJECT));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+2);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_COURSEPROJECT);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, 900, 500, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

setlocale(LC\_ALL, "Russian");

HWND button = CreateWindowExW(WS\_EX\_TRANSPARENT, L"BUTTON", L"Прочитать из файла", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 360, 200, 150, 40, hWnd, (HMENU)1, GetModuleHandle(NULL), 0);

HWND buttonRandom = CreateWindowExW(WS\_EX\_TRANSPARENT, L"BUTTON", L"Случайные значения", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 360, 100, 150, 40, hWnd, (HMENU)2, GetModuleHandle(NULL), 0);

HWND rangeText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("Размерность"), WS\_CHILD | WS\_VISIBLE, 500, 20, 100, 20, hWnd, NULL, NULL, NULL);

hRangeTextBox = CreateWindowEx(NULL, TEXT("edit"), TEXT("3"), WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 500, 40, 100, 20, hWnd, NULL, NULL, NULL);

HWND threadText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("Кол-во потоков"), WS\_CHILD | WS\_VISIBLE, 360, 20, 120, 20, hWnd, NULL, NULL, NULL);

hThreadTextBox = CreateWindowEx(NULL, TEXT("edit"), TEXT("8"), WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 360, 40, 120, 20, hWnd, NULL, NULL, NULL);

HWND luText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("LU-разложение"), WS\_CHILD | WS\_VISIBLE, 20, 5, 150, 20, hWnd, NULL, NULL, NULL);

luList = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 20, 25, 150, 330, hWnd, NULL, NULL, NULL);

HWND luParText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("LU-разложение пар."), WS\_CHILD | WS\_VISIBLE, 190, 5, 150, 20, hWnd, NULL, NULL, NULL);

luParList = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 190, 25, 150, 330, hWnd, NULL, NULL, NULL);

HWND gaussParText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("Метод Гаусса пар."), WS\_CHILD | WS\_VISIBLE, 660, 5, 150, 20, hWnd, NULL, NULL, NULL);

gaussParList = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 660, 25, 150, 330, hWnd, NULL, NULL, NULL);

HWND timeText = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("static"), TEXT("Время выполнения"), WS\_CHILD | WS\_VISIBLE, 360, 360, 150, 20, hWnd, NULL, NULL, NULL);

timeLU = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 20, 360, 150, 20, hWnd, NULL, NULL, NULL);

timeLUParallel = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 190, 360, 150, 20, hWnd, NULL, NULL, NULL);

timeGaussParallel = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("listbox"), TEXT(""), WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL, 660, 360, 150, 20, hWnd, NULL, NULL, NULL);

}

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case 1:

{

SendMessage(luList, LB\_RESETCONTENT, 0, 0);

SendMessage(gaussParList, LB\_RESETCONTENT, 0, 0);

SendMessage(luParList, LB\_RESETCONTENT, 0, 0);

SendMessage(timeLU, LB\_RESETCONTENT, 0, 0);

SendMessage(timeLUParallel, LB\_RESETCONTENT, 0, 0);

SendMessage(timeGaussParallel, LB\_RESETCONTENT, 0, 0);

TCHAR\* bufThr = new TCHAR[100];

GetWindowText(hThreadTextBox, bufThr, 100);

thread = \_ttoi(bufThr);

if (thread > 0) {

ifstream in("input.txt");

ifstream inB("inputB.txt");

if (in.is\_open())

{

int count = 0;// число чисел в файле

int temp;

while (!in.eof())

{

in >> temp;//в пустоту считываем из файла числа

count++;

}

in.seekg(0, ios::beg);

in.clear();

int count\_space = 0;

char symbol;

while (!in.eof())

{

in.get(symbol);//считали текущий символ

if (symbol == ' ') count\_space++;//Если это пробел, то число пробелов увеличиваем

if (symbol == '\n') break;//Если дошли до конца строки, то выходим из цикла

}

in.seekg(0, ios::beg);

in.clear();

n = count / (count\_space + 1);

int m = count\_space + 1;//число столбцов на единицу больше числа пробелов

slau.A = new double\* [n];

for (int i = 0; i < n; i++) slau.A[i] = new double[m];

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

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

{

in >> temp;

slau.A[i][j] = temp;

}

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

{

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

{

temp = slau.A[i][j];

}

}

in.close();

}

if (inB.is\_open())

{

int count = 0;

int temp;

while (!inB.eof())

{

inB >> temp;

count++;

}

inB.seekg(0, ios::beg);

inB.clear();

int count\_space = 0;

char symbol;

while (!in.eof())

{

in.get(symbol);

if (symbol == ' ') count\_space++;

if (symbol == '\n') break;

}

in.seekg(0, ios::beg);

in.clear();

n = count / (count\_space + 1);

slau.B = new double[n];

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

{

inB >> temp;

slau.B[i] = temp;

}

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

{

temp = slau.B[i];

}

inB.close();

if (!CheckMatrix(slau, n))

{

MessageBox(hWnd, L"Ошибка в матрице.", NULL, NULL);

slau.A = NULL;

slau.B = NULL;

break;

}

MessageBox(hWnd, L"Данные получены.", L"Все хорошо", NULL);

Calculate();

}

else

{

MessageBox(hWnd, L"Файл не найден.", NULL, NULL);

}

}

else

{

MessageBox(hWnd, L"Вы ввели неверное число", NULL, NULL);

}

}

break;

case 2:

{

TCHAR\* buf = new TCHAR[100];

GetWindowText(hRangeTextBox, buf, 100);

n = \_ttoi(buf);

TCHAR\* bufThr = new TCHAR[100];

GetWindowText(hThreadTextBox, bufThr, 100);

thread = \_ttoi(bufThr);

if (n > 0 && thread > 0) {

slau.A = new double\* [n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.B = new double[n];

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

{

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

{

slau.A[i][j] = 0;

}

slau.B[i] = 0;

}

double a, b;

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

{

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

{

slau.A[i][j] = rand() % 20 + 1;

a = slau.A[i][j];

}

slau.B[i] = rand() % 20 + 1;

b = slau.B[i];

}

if (slau.A != NULL && slau.B != NULL)

{

SendMessage(luList, LB\_RESETCONTENT, 0, 0);

SendMessage(gaussParList, LB\_RESETCONTENT, 0, 0);

SendMessage(luParList, LB\_RESETCONTENT, 0, 0);

SendMessage(timeLU, LB\_RESETCONTENT, 0, 0);

SendMessage(timeLUParallel, LB\_RESETCONTENT, 0, 0);

SendMessage(timeGaussParallel, LB\_RESETCONTENT, 0, 0);

Calculate();

}

}

else

{

MessageBox(hWnd, L"Вы ввели неверное число", NULL, NULL);

}

}

break;

case IDM\_ABOUT:

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

void Calculate()

{

TCHAR\* buf = new TCHAR[100];

GetWindowText(hRangeTextBox, buf, 100);

n = \_ttoi(buf);

GetWindowText(hThreadTextBox, buf, 100);

thread = \_ttoi(buf);

clock\_t start\_c\_time = clock();

xLu = Matrix::LU(slau, n);

clock\_t end\_c\_time = clock();

double dt = (float)(end\_c\_time - start\_c\_time)/CLOCKS\_PER\_SEC;

stringstream sstime;

sstime << dt << endl;

string newString = sstime.str();

USES\_CONVERSION;

LPWSTR pstTime = A2T(newString.c\_str());

SendMessage(timeLU, LB\_ADDSTRING, 0, (LPARAM)pstTime);

start\_c\_time = end\_c\_time = dt = 0;

start\_c\_time = clock();

xLuPar = Matrix::LUPar(slau, n, thread);

end\_c\_time = clock();

dt = (float)(end\_c\_time - start\_c\_time) / CLOCKS\_PER\_SEC;

stringstream sstime2;

sstime2 << dt << endl;

string newString1 = sstime2.str();

pstTime = A2T(newString1.c\_str());

SendMessage(timeLUParallel, LB\_ADDSTRING, 0, (LPARAM)pstTime);

start\_c\_time = end\_c\_time = dt = 0;

start\_c\_time = clock();

xGaussPar = Matrix::GaussPar(slau, n, thread);

end\_c\_time = clock();

dt = (float)(end\_c\_time - start\_c\_time) / CLOCKS\_PER\_SEC;

stringstream sstime3;

sstime3 << dt << endl;

newString = sstime3.str();

pstTime = A2T(newString.c\_str());

SendMessage(timeGaussParallel, LB\_ADDSTRING, 0, (LPARAM)pstTime);

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

{

stringstream stringstream1, stringstream2, stringstream3;

stringstream1 << floor(xLu[i] \* 100) / 100 << endl;

string newString = stringstream1.str();

USES\_CONVERSION;

LPWSTR pst = A2T(newString.c\_str());

SendMessage(luList, LB\_ADDSTRING, 0, (LPARAM)pst);

stringstream2 << floor(xGaussPar[i]\*100)/100 << endl;

newString = stringstream2.str();

pst = A2T(newString.c\_str());

SendMessage(gaussParList, LB\_ADDSTRING, 0, (LPARAM)pst);

stringstream3 << floor(xLuPar[i]\*100)/100 << endl;

newString = stringstream3.str();

pst = A2T(newString.c\_str());

SendMessage(luParList, LB\_ADDSTRING, 0, (LPARAM)pst);

}

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

delete[] slau.A[i];

delete[] slau.A;

delete[] slau.B;

}

bool CheckMatrix(SLAU slau, int n)//проверяем матрицу на нулевые значения

{

int countZero = 0;

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

{

for (int j = 0; j < n; j++) {

if (slau.A[i][j] == 0)

countZero++;

}

if (countZero == n)

return false;

countZero = 0;

}

return true;

}

**SLAU.cpp**

#include "SLAU.h"

class SLAU {

public:

double\*\* A;

double\* B;

double\* x;

};

**CourseTest.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include "..\\CourseProject\Matrix.h"

#include "..\\CourseProject\Matrix.cpp"

#include "../CourseProject/CourseProject.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace CourseTest

{

TEST\_CLASS(CourseTest)

{

public:

TEST\_METHOD(TestLUTrue)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double [n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::LU(slau, n);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(TestLUFalse)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double[n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::LU(slau, n);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(TestLUParTrue)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double[n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::LUPar(slau, n, 4);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(TestLUParFalse)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double[n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::LUPar(slau, n, 4);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(TestGaussParTrue)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double[n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::GaussPar(slau, n, 4);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(TestGaussParFalse)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

slau.B = new double[n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 40;

slau.A[0][2] = 28;

slau.B[0] = 1;

slau.A[1][0] = 3;

slau.A[1][1] = 4;

slau.A[1][2] = 5;

slau.B[1] = 2;

slau.A[2][0] = 9;

slau.A[2][1] = 12;

slau.A[2][2] = 15;

slau.B[2] = 3;

Matrix matrix;

double\* actual = new double[n];

actual = Matrix::GaussPar(slau, n, 4);

double\* rezult = new double[n];

rezult[0] = 0.067;

rezult[1] = -0.145;

rezult[2] = 0.394;

for (int i = 0; i < n; i++) {

actual[i] = round(actual[i] \* 100) / 100;

rezult[i] = round(rezult[i] \* 100) / 100;

}

Assert::AreEqual(actual[2], rezult[2]);

}

TEST\_METHOD(CheckZeroLine)

{

int n = 3;

SLAU slau;

slau.A = new double\* [n];

for (int i = 0; i < n; i++) slau.A[i] = new double[n];

slau.B = new double[n];

slau.A[0][0] = 4;

slau.A[0][1] = 5;

slau.A[0][2] = 6;

slau.B[0] = 1;

slau.A[1][0] = 1;

slau.A[1][1] = 2;

slau.A[1][2] = 3;

slau.B[1] = 78;

slau.A[2][0] = 0;

slau.A[2][1] = 0;

slau.A[2][2] = 0;

slau.B[2] = 4;

bool check = CheckMatrix(slau, n);

Assert::AreEqual(false, check);

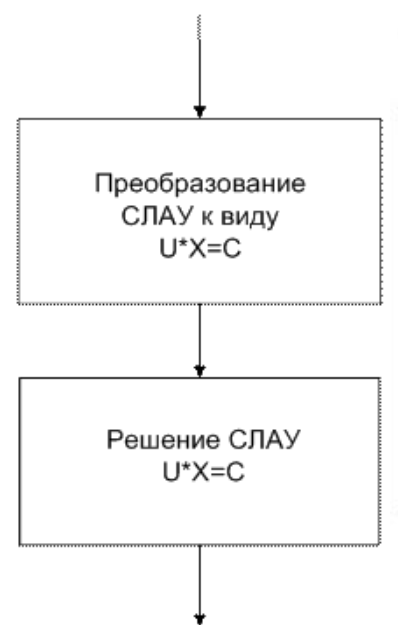
}

};

}

**Приложение Б**

**(обязательное)**

****

****

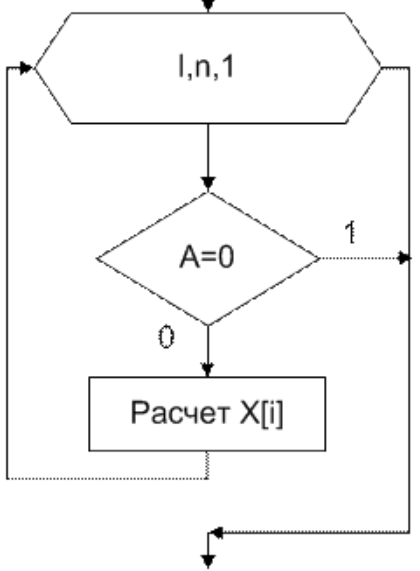
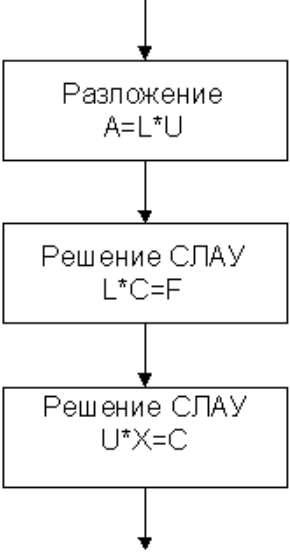
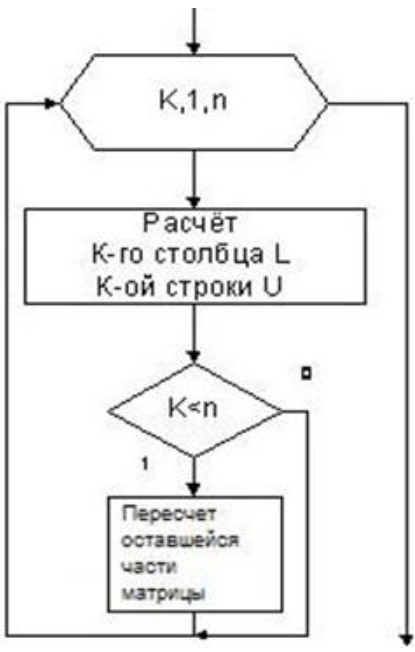
****

Рисунок Б.1 – Блок-схема метода LU-разложения





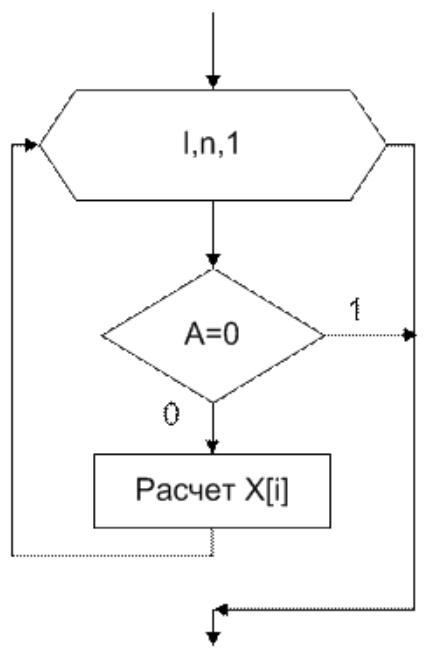


Рисунок Б.2 – Блок-схема алгоритма метода Гаусса