МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Нижегородский государственный технический университет

им. Р.Е. Алексеева» (НГТУ)

Кафедра: «Цифровая экономика»

Дисциплина: «Численные методы»

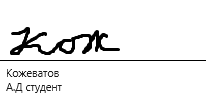
**Лабораторная работа №2**

**«Численные методы решения систем алгебраических линейных уравнений»**

Выполнил:

студент 3-го курса группы 21-САИ

Кожеватов Алексей Дмитриевич



Проверил:

д.ф.м.н., проф. Катаева Лилия Юрьевна

11.10.2023

Подпись преподавателя:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород, 2023

Содержание

[1. Постановка задачи 2](#_Toc156850823)

[1.1 Задание №1 3](#_Toc156850824)

[1.2 Задание №2 3](#_Toc156850825)

[1.3 Задание №3 3](#_Toc156850826)

[1.4 Задание №4 3](#_Toc156850827)

[1.5 Задание №5 3](#_Toc156850828)

[1.6 Задание №6 3](#_Toc156850829)

[2. Методы решений и инструменты 4](#_Toc156850830)

[2.1 Инструменты 4](#_Toc156850831)

[2.2 Методы решений 5](#_Toc156850832)

[2.2.1 Метод отражений 5](#_Toc156850833)

[2.2.2 Метод квадратного корня 6](#_Toc156850834)

[2.2.3 Метод сопряженных градиентов 7](#_Toc156850835)

[2.2.4 Метод ортогонализации 8](#_Toc156850836)

[2.2.5 Метод простой итерации 10](#_Toc156850837)

[2.2.6 Метод Зейделя 11](#_Toc156850838)

[3. Таблица идентификаторов 13](#_Toc156850839)

[4. Ручной счет 18](#_Toc156850840)

[4.1 Метод отражений 18](#_Toc156850841)

[4.2 Метод квадратного корня. 19](#_Toc156850842)

[4.3 Метод сопряжённых градиентов 21](#_Toc156850843)

[4.4 Метод ортогонализации 23](#_Toc156850844)

[4.5 Метод простой итерации 26](#_Toc156850845)

[4.6 Метод Зейделя 28](#_Toc156850846)

[5. Реализация задачи в Mathcad 31](#_Toc156850847)

[5.1 Метод отражений 31](#_Toc156850848)

[5.2 Метод квадратных корней 32](#_Toc156850849)

[5.3 Метод сопряжённых градиентов 33](#_Toc156850850)

[5.4 Метод ортогонализации 34](#_Toc156850851)

[5.5 Метод простой итерации 35](#_Toc156850852)

[5.6 Метод Зейделя 36](#_Toc156850853)

[Реализация задачи в Microsoft Office Excel 37](#_Toc156850854)

[Задание 1 37](#_Toc156850855)

[Задание 2 38](#_Toc156850856)

[Задание 3 39](#_Toc156850857)

[Задание 4 40](#_Toc156850858)

[Задание 5 41](#_Toc156850859)

[Задание 6 42](#_Toc156850860)

[7. Реализация на С++ 43](#_Toc156850861)

[7.1 Меню 43](#_Toc156850862)

[7.2 Метод квадратных корней 44](#_Toc156850863)

[7.3 Метод сопряжённых градиентов 49](#_Toc156850864)

[7.4 Метод отражений 54](#_Toc156850865)

[7.5 Метод ортогонализации 60](#_Toc156850866)

[7.6 Метод простой итерации 63](#_Toc156850867)

[6.7 Метод Зейделя 69](#_Toc156850868)

[8. Реализация на Java 74](#_Toc156850869)

[8.1 Меню 74](#_Toc156850870)

[8.2 Класс комплексных чисел 76](#_Toc156850871)

[8.3 Метод сопряжённых градиентов 78](#_Toc156850872)

[8.4 Метод квадратных корней 84](#_Toc156850873)

[8.5 Метод ортогонализации 87](#_Toc156850874)

[8.6 Метод отражений 90](#_Toc156850875)

[8.7 Метод простой итерации 95](#_Toc156850876)

[8.8 Метод Зейделя 99](#_Toc156850877)

[9. Результат и анализ реализаций 105](#_Toc156850878)

[10. Вывод 106](#_Toc156850879)

[10. Использованная литература 108](#_Toc156850880)

# 1. Постановка задачи

Вариант 7

## 1.1 Задание №1

Решить систему методом отражений.

## 1.2 Задание №2

Решить систему методом квадратного корня.

## 1.3 Задание №3

Решить систему методом сопряженных градиентов.

## 1.4 Задание №4

Решить систему методом ортогонализации.

## 1.5 Задание №5

Решить систему методом простой итерации.

## 1.6 Задание №6

Решить систему методом Зейделя.

# 2. Методы решений и инструменты

## 2.1 Инструменты

При решении лабораторной работы были использованы следующие инструменты:

1. Mathcad 15 (версия: M050 [], разрядность: x64);
2. Eclipse IDE (версия: 2022-09 (4.25.0), разрядность: x64);
3. Visual Studio IDE (версия: 17.7.3, разрядность: x64).
4. Microsoft Excel (версия: 2023, разрядность: x64).

Mathcad 15 — это программное обеспечение для математического моделирования и анализа технических и научных данных. Оно предоставляет удобную среду для создания и решения математических выражений и уравнений, а также выполнения численных и символьных вычислений.

Eclipse IDE — это интегрированная среда разработки (IDE), используемая в компьютерном программировании. Оно содержит базовое рабочее пространство и расширяемую систему подключаемых модулей для настройки среды. Это вторая по популярности среда IDE для разработки на Java, и до 2016 года она была самой популярной. Eclipse написан в основном на Java и его основное применение заключается в разработке приложений Java, но он также может быть использован для разработки приложений на других языках программирования с помощью плагинов.

Visual Studio IDE- это интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для разработки программного обеспечения. В Visual Studio предоставляются различные инструменты и функциональные возможности, упрощающие процесс разработки, отладки и тестирования приложений.

Visual Studio поддерживает множество языков программирования, включая C++, C#, Visual Basic, F#, JavaScript и другие. Для разработки на C++ в Visual Studio используется компилятор Microsoft C++, который обеспечивает мощные возможности компиляции и оптимизации кода.

Visual Studio обеспечивает разработчиков на C++ всеми необходимыми инструментами для создания высококачественного программного обеспечения. Отличительной чертой Visual Studio является его обширная функциональность и поддержка различных платформ и технологий, что делает его одним из популярных выборов для разработки на C++.

## 2.2 Методы решений

### 2.2.1 Метод отражений

Метод отражений используется для вычисления QR-разложения матрицы. Он позволяет представить исходную матрицу в виде произведения ортогональной матрицы Q и верхнетреугольной матрицы R. Метод отражений основан на преобразовании матрицы с помощью последовательных отражений относительно гиперплоскостей. Эти отражения позволяют привести матрицу к треугольному виду. Одним из преимуществ метода отражений является его устойчивость к ошибкам округления.

Пусть и произвольные вектор-столбцы, причем вектор имеет единичную длину. Тогда найдется такой вектор , что построенная по нему матрица отражения переведет вектор в вектор, коллинеарный вектору , т.е. .

Вектор строится по правилу

,

где , , .

Будем преобразовывать расширенную матрицу систему по правилу

,

с помощью умножения слева на последовательность матриц отражения . Для построения матрицы на первом шаге метода в качестве вектора берется первый столбец расширенной матрицы, а в качестве вектора - координатный вектор . В силу выбора векторов и все координаты первого столбца расширенной матрицы, кроме первой, после выполнения первого шага метода будут равны нулю.

Пусть уже построена матрица , у которой , , . Теперь в качестве и берутся вектора

,

,

где в векторе единица стоит на -ом месте. После выполнения -го шага метода отражений получим матрицу , у которой все элементы, стоящие ниже главной диагонали, в первых -ом столбцах будут равны нулю. Невозможность выполнения очередного шага связана только с равенством нулю вектора , а это невозможно, так как матрица является невырожденной.

После - шага получим матрицу, первые столбцов которой образуют верхнюю треугольную матрицу . Система уравнений, соответствующая полученной расширенной матрице, равносильна исходной системе. Значения неизвестных находятся аналогично обратному ходу метода Гаусс , ,

### 2.2.2 Метод квадратного корня

Пусть дана линейная система , где – симметрическая матрица, т.е . Тогда матрицу можно представить в виде произведения двух транспонированных между собой матриц.

Где и

Производя перемножение матриц , для определения элементов используем формулы (19):

Корни уравнения найдем через формулы (20) и (21)

Находим y по формулам (22), и x по формулам (23)

### 2.2.3 Метод сопряженных градиентов

Метод градиентов используется для оптимизации и поиска минимума или максимума функции. Он основан на использовании градиента функции, который показывает направление наиболее быстрого возрастания функции. В методе градиентов начальная точка выбирается произвольно, и затем на каждой итерации вычисляется градиент функции в текущей точке. Затем производится шаг в направлении, противоположном градиенту, с целью приближения к оптимальной точке. Процесс повторяется до достижения заданной точности или заданного числа итераций.

Метод сопряженных градиентов предназначен для решения систем линейных алгебраических уравнений с симметричной положительно определенной матрицей. Матрица называется положительно определенной, если скалярное произведение для всех ненулевых векторов . Для того чтобы матрица была положительно определенной необходимо и достаточно, чтобы все ее главные миноры были положительны.

Рассмотрим систему линейных алгебраических уравнений с положительно определенной симметричной матрицей. Пусть - решение этой системы. Покажем, что в этом случае дает минимум функционала . Имеем

.

Так как матрица положительно определенная, то из последнего равенства следует, что решение системы (4.1) дает минимум функционала .

Будем искать этот минимум итерационным методом. Для этого предположим, что заданы два вектора и . Последовательные приближения к решению системы (4.1) будем строить по формулам

,

, где

.

Параметры и будем определять из условия минимума функционала в плоскости, проходящей через точку и натянутой на векторы и , т.е. минимум функционала ищется на множестве . Преобразуем наш функционал на этом множестве

.

Для определения минимума используем необходимое условие экстремума функции. Продифференцируем по и и приравняем к нулю

,

.

Таким образом, получаем систему уравнений:

,

.

Так как

, то последнюю систему можно представить в виде

Это система двух уравнений с двумя неизвестными и . Решая ее, получаем

,

Окончательно получаем

.

### 2.2.4 Метод ортогонализации

Метод ортогонализации используется для построения ортогонального базиса в линейном пространстве. Он применяется для преобразования линейно независимого набора векторов в ортогональный набор. Метод ортогонализации Грама-Шмидта начинается с выбора первого вектора из исходного набора. Затем каждый следующий вектор вычитается из проекции на предыдущие ортогональные векторы, чтобы получить ортогональный вектор. Процесс повторяется для всех векторов в наборе, пока не будет построен полный ортогональный базис.

Рассмотрим систему и запишем ее в виде

,

где , . Обозначим , , . Тогда система (4.19) перепишется в виде

Решение системы уравнений с невырожденной матрицей сводится к нахождению такого вектора , который имеет последнюю координату, равную единице, и ортогонален к линейно независимым векторам . Ортогональность вектора к векторам влечет за собой ортогональность ко всему подпространству , натянутому на них, и, следовательно, к любому его базису. И наоборот, ортогональность вектора к некоторому базису подпространства влечет ортогональность ко всем векторам . Поэтому для решения системы достаточно построить ненулевой вектор , ортогональный к какому-нибудь базису подпространства . Если - последняя координата вектора , то .

Добавим к системе векторов линейно независимый с ними вектор , а затем будем последовательно строить систему ортонормированных векторов , что для всех векторы являются базисом подпространства , натянутого на вектора . В этом случае вектор и будет искомым вектором .

Применим правило Шмидта для построения ортонормированного базиса пространства, натянутого на заданные линейно независимые векторы. Обозначим через ортогональный базис подпространства , а через - ортонормированный в евклидовой метрике базис того же подпространства. Так как векторы нормированы, то их можно представить в виде

, .

На первом шаге метода положим , . Пусть для некоторого шага уже построен ортогональный базис и ортонормированный базис подпространства . Вектор будем искать как линейную комбинацию векторов

Условие ортогональности вектора к ортогональным векторам или, что то же самое, к векторам дает . Поэтому

.

Таким образом, с помощью этого итерационного процесса и соотношения строится ортонормированная система векторов . Векторы являются базисом подпространства . Следовательно, решение исходной системы имеет вид , , где и - соответственно -я и я

координаты вектора .

### 2.2.5 Метод простой итерации

Метод простой итерации также является итерационным методом решения систем линейных уравнений. Он основан на преобразовании исходной системы уравнений в эквивалентную форму, позволяющую построить итерационную последовательность. В этом методе система уравнений приводится к виду, где каждое уравнение содержит одну переменную, выраженную через остальные переменные. Затем задаются начальные значения переменных и выполняется итерационный процесс, в котором каждая переменная пересчитывается на основе предыдущих значений переменных. Процесс повторяется до достижения заданной точности или заданного числа итераций.

Если от уравнения перейти к эквивалентному уравнению

,

то возможно использование итерационной процедуры нахождения корня уравнения. При этом функцию часто называют итерационной.

Процедура нахождения корня достаточно проста: выберем какое-либо начальное приближение и подставим его в уравнение. В результате получим первое приближение , которое опять подставим в уравнение. Продолжая этот процесс неограниченно, получим последовательность приближений к корню, вычисляемую по формуле

.

Очевидно, что если существует предел последовательности и функция непрерывна, то получим равенство

,

а это значит, что – корень уравнения. Отметим, что метод простой итерации является одношаговым.

Геометрическая интерпретация метода представлена на Рисунке. Из рисунка видно, что метод простой итерации может приводить как к сходящейся к решению процедуре, так и к расходящейся.

Для использования этого метода необходимо установить критерий его сходимости. Будем считать, что в итерационной формуле

где и – отклонения -го и ()-го приближения от корня . Если процесс уточнения осуществляется вблизи корня , то функцию приближенно можно представить в виде первых двух членов разложения в ряд Тейлора. Используя выражение, получаем

.

Принимая во внимание равенство, имеем

.

Для обеспечения сходимости итерационного процесса необходимо потребовать, чтобы , что приводит к условию , откуда

.

Если , то имеет место односторонняя сходимость к решению ; если же выполняется условие , то сходимость будет двухсторонней. В случае, когда или , итерационный процесс расходится.

Зная условие сходимости итерационного процесса , можно разработать процедуру перехода от уравнения к уравнению так, чтобы итерационный процесс сходился наиболее эффективным образом. Для этого умножим обе части равенства на некоторую константу  и прибавим , получим

.

Таким образом, искомая функция  будет иметь вид

.

Произвольный параметр выбирается так, чтобы было обеспечено условие сходимости. Наиболее удобно выбрать этот параметр из условия для обеспечения двухсторонней сходимости, при этом критерий окончания итерационного процесса будет наиболее простым

.

### 2.2.6 Метод Зейделя

Метод Зейделя является итерационным методом решения систем линейных уравнений. Он основан на последовательном приближенном вычислении компонентов вектора-решения. В начале итерационного процесса используются начальные приближения для компонентов вектора-решения. Затем каждая компонента рассчитывается на основе предыдущих значений компонент исходного вектора. Процесс повторяется до достижения заданной точности или заданного числа итераций. Метод Зейделя сходится для диагонально-доминантных систем или систем с симметричной положительно определенной матрицей.

Метод Зейделя представляет собой некоторую модификацию метода итераций. Основная его идея заключается в том, что при вычислении (*k* + 1)-го приближения неизвестной *xi* учитываются уже вычисленные ранее (*k* + 1)-е приближения неизвестных *x*1, *x*2, …, *xi* - 1.

Пусть получена эквивалентная система. Выберем произвольно начальные приближения корней . Далее, предполагая, что *k*-ые приближения корней известны, согласно Зейделю будем строить (*k* + 1)-е приближения корней по формулам:

Процесс повторяется до достижения заданной точности или заданного числа итераций.

**3. Таблица идентификаторов**

Таблица 1

Таблица идентификаторов, переменных, используемых для решения задачи

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название переменной | | Mathcad | | | | C++ | Java | | | | Комментарий | | | |
| Метод отражений | | | | | | | | | | | | | | |
| i | i | | i | | | | | i | | | | | Переменная для записи порядкового номера | |
| j | j | | j | | | | | j | | | | | Переменная для записи порядкового номера | |
| A | A | | massA[i][j] | | | | | massA[i][j] | | | | | Матрица коэффициентов при x и ответов уравнений | |
| x | xi | | x[i] | | | | | x[i] | | | | | Неизвестное для нахождения | |
| E | E | | E[i][j] | | | | | E[i][j] | | | | | Единичная матрица | |
| A0 | A0 | |  | | | | |  | | | | | Матрица коэффициентов при x и ответов уравнений для проверки решения | |
| s | si | | s[i] | | | | | s[i] | | | | | Вектор диагональных элементов матрицы А | |
| l | li | | l[i] | | | | | l[i] | | | | | Вектор, состоящий из единиц | |
| d | di | | d[i] | | | | | d[i] | | | | | Сумма квадратов элементов вектора si | |
| b | bi | | b[i] | | | | | b[i] | | | | | Вектор для расчётов | |
| ρ | pi | | p[i] | | | | | p[i] | | | | | Вектор для расчётов | |
| W | Wi | | W[i] | | | | | W[i] | | | | | Вектор для построения матриц отражений | |
| U | Ui | | U[i][j] | | | | | U[i][j] | | | | | Матрица отражений | |
| Метод квадратных корней | | | | | | | | | | | | | | |
| i | i | | | i | | | | | i | | | Переменная для записи порядкового номера | | |
| j | j | | | j | | | | | j | | | Переменная для записи порядкового номера | | |
| k |  | | | k | | | | | k | | | Переменная для записи порядкового номера | | |
| A | A | | | massA[i][j] | | | | | massA[i][j] | | | Матрица коэффициентов при x | | |
| B | B | | | massB[i] | | | | | massB[i] | | | Вектор ответов уравнений | | |
| x | xi | | | x[i] | | | | | x[i] | | | Неизвестное для нахождения | | |
| U | U | | | U[i][j] | | | | | U[i][j] | | | Матрица | | |
| Ut | UT | | | Ut[i][j] | | | | | Ut[i][j] | | | Транспонированная матрица U | | |
| uij | uij | | | u[i][j] | | | | | u[i][j] | | | Элементы матрицы U | | |
|  | AP | | |  | | | | |  | | | Матрица А для проверки правильности нахождения х | | |
| y | yi | | | Y[i] | | | | | y[i] | | | Вектор | | |
| B | Bi | | |  | | | | |  | | | Матрица B для проверки правильности нахождения х | | |
| Atrans |  | | | Atrans[i][k] | | | | |  | | | Транспонированная матрица А | | |
| a |  | | | a[i][j] | | | | | a[i][j] | | | Матрица, полученная умножением А на А транспонированную | | |
| b |  | | | b[j] | | | | | b[j] | | | Транспонированная матрица **B** | | |
| n |  | | | n | | | | | n | | | Количество неизвестных | | |
| sum |  | | | sum | | | | | sum | | | Переменная для записи суммы по строке массива | | |
| Метод сопряжённых градиентов | | | | | | | | | | | | | | |
| i | i | | | | i | | | | | i | | | | Переменная для записи порядкового номера |
| j | j | | | | j | | | | | j | | | | Переменная для записи порядкового номера |
| A | A | | | | massA[i][j] | | | | | massA[i][j] | | | | Матрица коэффициентов при x |
| B | B | | | | massB[i] | | | | | massB[i] | | | | Вектор ответов уравнений |
| x | xi | | | | x[i] | | | | | x[i] | | | | Неизвестное для нахождения |
| r | r | | | | r[i] | | | | | r[i] | | | | Вектор |
| Δ | Δ | | | | delta[i] | | | | | delta[i] | | | | Вектор |
| Ar | Ar | | | | Ar[i] | | | | | Ar[i] | | | | Произведение матрицы A на вектор r |
| AΔ | AΔ | | | | Adelta[i] | | | | | Adelta[i] | | | | Произведение матрицы A на вектор Δ |
| (r,r) | (r,r) | | | | rr | | | | | rr | | | | Скалярное произведение векторов r |
| (r,Δ) | (r,Δ) | | | | rDelta | | | | | rDelta | | | | Скалярное произведение векторов r и Δ |
| (Ar,Δ) | (Ar,Δ) | | | | ArDelta | | | | | ArDelta | | | | Скалярное произведение векторов Ar и Δ |
| (Ar,r) | (Ar,r) | | | | Arr | | | | | Arr | | | | Скалярное произведение векторов Ar и r |
| (A,Δ) | (A,Δ) | | | | Adeltadelta | | | | | Adeltadelta | | | | Скалярное произведение векторов AΔ и Δ |
| β | β | | | | bt | | | | | bt | | | | Параметр |
| α | α | | | | al | | | | | al | | | | Параметр |
| e |  | | | | e | | | | | e | | | | Переменная, которая хранит полученную точность результата |
| E |  | | | | E | | | | | E | | | | Переменная, в которую записывается нужная точность для сравнения с результатом |
| Ax |  | | | | Ax[i] | | | | | Ax[i] | | | | Произведение матрицы A на вектор x предыдущей итерации |
| count |  | | | | count | | | | | count | | | | Переменная для подсчёта итераций |
| n |  | | | | n | | | | | n | | | | Переменная, хранящая информацию о количестве неизвестных |
| Метод ортогонализации | | | | | | | | | | | | | | |
| i | i | | | | i | | | | | i | | | | Переменная для записи порядкового номера |
| j | j | | | | j | | | | | j | | | | Переменная для записи порядкового номера |
| A | A | | | | massA[i][j] | | | | | massA[i][j] | | | | Матрица коэффициентов при x |
| B | B | | | | massB[i] | | | | | massB[i] | | | | Вектор ответов уравнений |
| x | xi | | | | x[i] | | | | | x[i] | | | | Неизвестное для нахождения |
| r | ri | | | | r[i] | | | | | r[i] | | | | Вектор для расчётов |
| t | ti,j | | | | t[i] [j] | | | | | t[i] [j] | | | | Вектор для расчётов |
| d | di | | | | d[i] | | | | | d[i] | | | | Вектор для расчётов |
| f | fi | | | | f[i] | | | | | f[i] | | | | Вектор для расчётов |
| b | bi | | | | b[i] | | | | | b[i] | | | | Вектор для расчётов |
| Метод простой итерации | | | | | | | | | | | | | | |
| i | i | | | | i | | | | | i | | | | Переменная для записи порядкового номера |
| j | j | | | | j | | | | | j | | | | Переменная для записи порядкового номера |
| A | A | | | | massA[i][j] | | | | | massA[i][j] | | | | Матрица коэффициентов при x |
| B | B | | | | massB[i] | | | | | massB[i] | | | | Вектор ответов уравнений |
| x | xi | | | | x[i] | | | | | x[i] | | | | Неизвестное для нахождения |
| xf |  | | | | xf[i] | | | | | xf[i] | | | | Переменная для хранения предыдущего значения неизвестного |
| n |  | | | | n | | | | | n | | | | Переменная, хранящая информацию о количестве неизвестных |
| eps |  | | | | eps | | | | | eps | | | | Переменная, хранящая необходимую точность результата |
| delta |  | | | | delta | | | | | delta | | | | Переменная для сравнения точности полученного результата с ожидаемым e |
| count |  | | | | count | | | | | count | | | | Переменная для подсчёта итераций |
| check |  | | | | check | | | | |  | | | | Переменная для проверки выполнения условия сходимости |
| key |  | | | | key | | | | |  | | | | Переменная, хранящая ответ пользователя на показ ответа, если условие 2 не выполняется |
| det |  | | | | det | | | | | det | | | | Переменная для хранения определителя матрицы |
| sum |  | | | | sum | | | | | sum | | | | Переменная для записи суммы по строке массива |
| Метод Зейделя | | | | | | | | | | | | | | |
| i | i | | | | i | | | | | i | | | | Переменная для записи порядкового номера |
| j | j | | | | j | | | | | j | | | | Переменная для записи порядкового номера |
| A | A | | | | massA[i][j] | | | | | massA[i][j] | | | | Матрица коэффициентов при x |
| B | B | | | | massB[i] | | | | | massB[i] | | | | Вектор ответов уравнений |
| x | xi | | | | x[i] | | | | | x[i] | | | | Неизвестное для нахождения |
| xf |  | | | | xf[i] | | | | | xf[i] | | | | Переменная для хранения предыдущего значения неизвестного |
| n |  | | | | n | | | | | n | | | | Переменная, хранящая информацию о количестве неизвестных |
| eps |  | | | | eps | | | | | eps | | | | Переменная, хранящая необходимую точность результата |
| delta |  | | | | delta | | | | | delta | | | | Переменная для сравнения точности полученного результата с ожидаемым e |
| count |  | | | | count | | | | | count | | | | Переменная для подсчёта итераций |
| check |  | | | | check | | | | |  | | | | Переменная для проверки выполнения условия сходимости |
| key |  | | | | key | | | | |  | | | | Переменная, хранящая ответ пользователя на показ ответа, если условие 2 не выполняется |
| det |  | | | | det | | | | | det | | | | Переменная для хранения определителя матрицы |
| sum |  | | | | sum | | | | | sum | | | | Переменная для записи суммы по строке массива |

# 4. Ручной счет

## 4.1 Метод отражений

Дана система:

По системе уравнений составим матрицу А:

Шаг 1.

Шаг 2.

Шаг 3.

Исходная система преобразована к системе с верхней треугольной матрицей. Пользуясь обратным ходом метода Гаусса, находим X.

Проверка:

## 4.2 Метод квадратного корня.

Дана система:

По системе уравнений составим матрицу А и B:

Приводим матрицу А к симметричному виду, а также изменяем матрицу B:

Найдем элементы матрицы T:

6,2150515+1,9485875i

12,10272643+0,492218973i

Найдем x методом обратного хода

Проверка:

## 4.3 Метод сопряжённых градиентов

Дана система:

По системе уравнений составим матрицу А и B:

Возьмём произвольный вектор и построим векторы и

Шаг 2.

Шаг 3.

Проверка:

## 4.4 Метод ортогонализации

Дана система:

По системе уравнений составим матрицу А и B:

СЛАУ записывается в векторном представлении и выбирается первый вектор ортогональной матрицы R

Вектор записываем в виде линейной комбинации двух ортогональных векторов

Аналогично находим

0,39259695

Аналогично находим

-0,108153149

-0,1116332

Находим корни

;

;

Проверка:

## 4.5 Метод простой итерации

Проверим условия применимости метода простой итерации

Условие не выполняется для

Условие не выполняется для

Условие не выполняется для

Условие не выполняется для

Выполним элементарные преобразования, приведя систему к виду, при котором условия применимости простой итерации будут выполняться.

1. новая вторая строка = меняем первую и вторую строку местами  
2. новая четвертая строка = вычитаем из 4 строки 3  
3. новая первая строка = четвертая строка + третья умноженная на 0,8

4. новая третья строка = умножим 3 строку на 3 прибавим 1 и 4, и 3 униженную на 1,5

Результат:

Условие выполняется для всех x.  
  
Для данной системы метод простой итерации можно применить

Найдем формулы для вычисления

В качестве начального приближения векторов выбираем нулевой вектор

1 итерация:

-2,200085

0,010111

11,052697

6,704545

Проверим выполнение условия для ε = 0,001

не выполняется для

не выполняется для

не выполняется для

не выполняется для

2 итерация:

Проверим выполнение условия для ε = 0,001

не выполняется для

не выполняется для

не выполняется для

не выполняется для

Аналогично находим остальные итерации, проверяя выполняется ли условия точности.  
  
***Проверка для 9 итерации***

Проверим выполнение условия для ε = 0,001

выполняется для

выполняется для

выполняется для

выполняется для

Проверка:

## 4.6 Метод Зейделя

Проверим условия применимости метода Зейделя

Условие не выполняется для

Условие не выполняется для

Условие не выполняется для

Условие не выполняется для

Выполним элементарные преобразования, приведя систему к виду, при котором условия применимости метода Зейделя будут выполняться.

1.меняем 1 строку с 3 и 2 с 4

2. 1 строчка = 1 + 2 \*0,8 4 строчка = + 3+1+2\*0,9

Запишем новую систему

Для данной системы метод Зейделя можно применить

Найдем формулы для вычисления

В качестве начального приближения векторов выбираем нулевой вектор

1 итерация:

Проверим выполнение условия для ε = 0,001

не выполняется для

не выполняется для

не выполняется для

не выполняется для

2 итерация:

Проверим выполнение условия для ε = 0,001

не выполняется для

не выполняется для

не выполняется для

не выполняется для

***Проверка для 9 итерации***

Проверим выполнение условия для ε = 0,001

выполняется для

выполняется для

выполняется для

выполняется для

Проверка:

# 5. Реализация задачи в Mathcad

*5.1 Метод отражений*

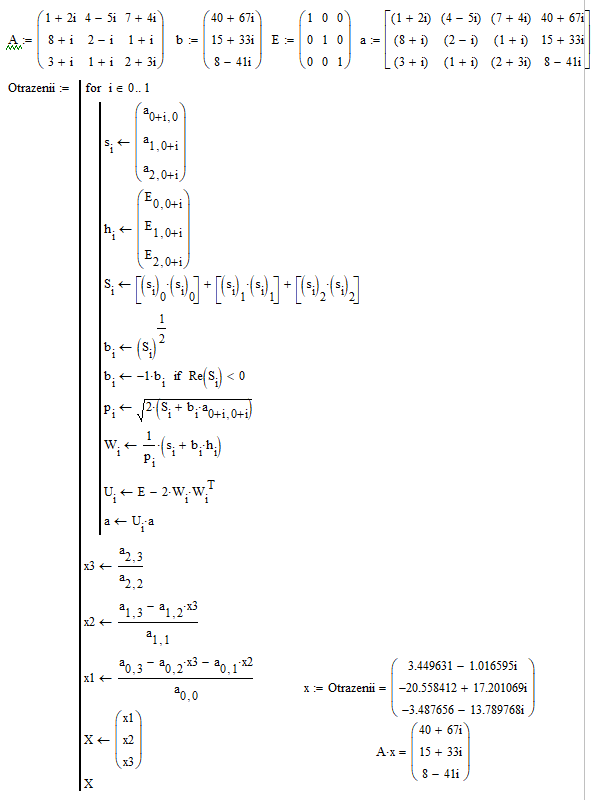


Рисунок 1.-Метод отражений

### 5.2 Метод квадратных корней

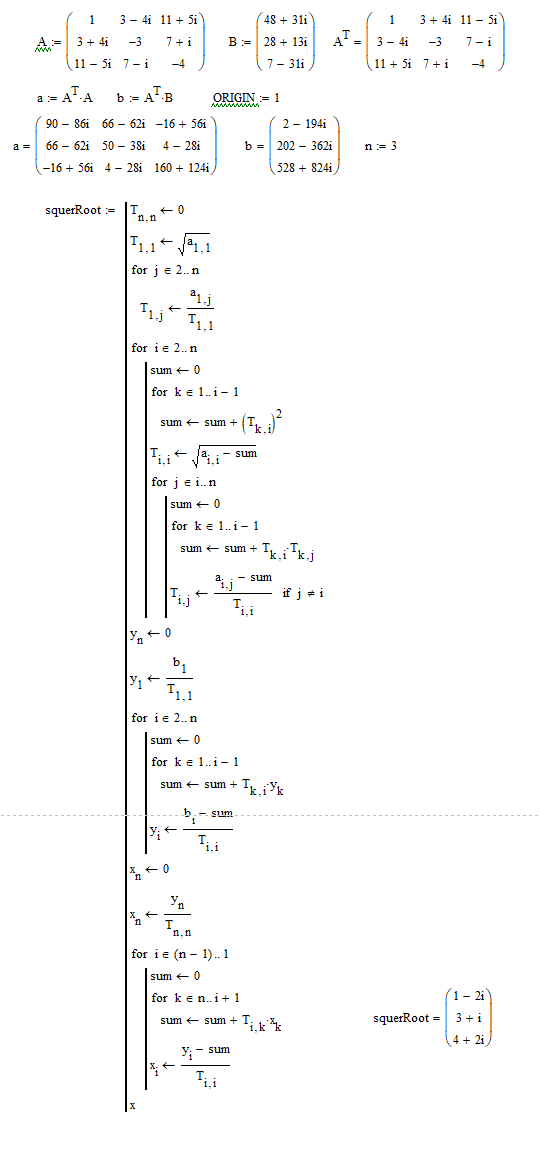


Рисунок 2.-Метод квадратных корней

### 5.3 Метод сопряжённых градиентов

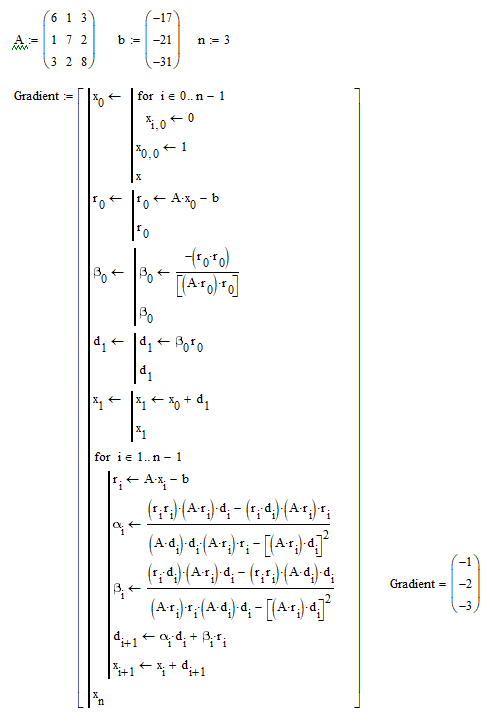


Рисунок 3.-Метод сопряженных градиентов

сопряженных градиентов

### 5.4 Метод ортогонализации

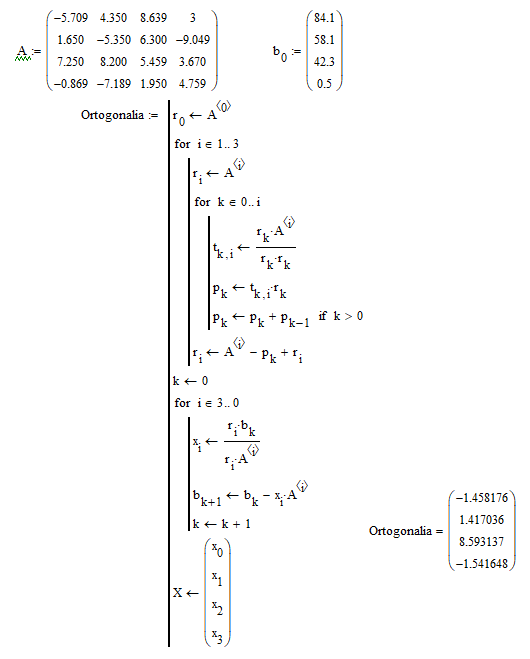


Рисунок 4.-Метод ортогонализации

### 5.5 Метод простой итерации

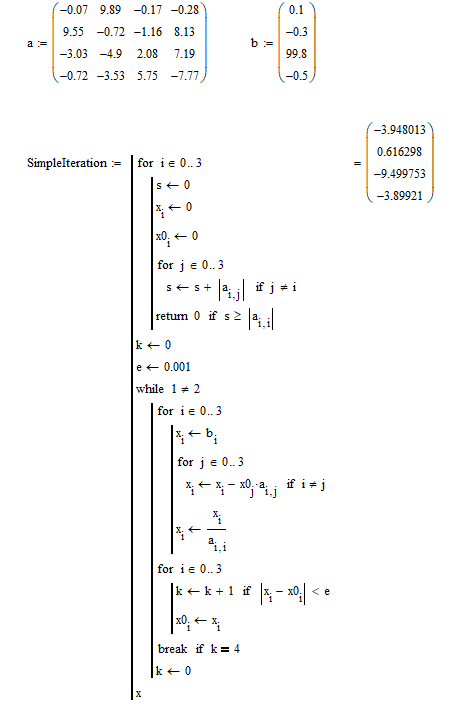


Рисунок 5.- Метод простой итерации

### 5.6 Метод Зейделя

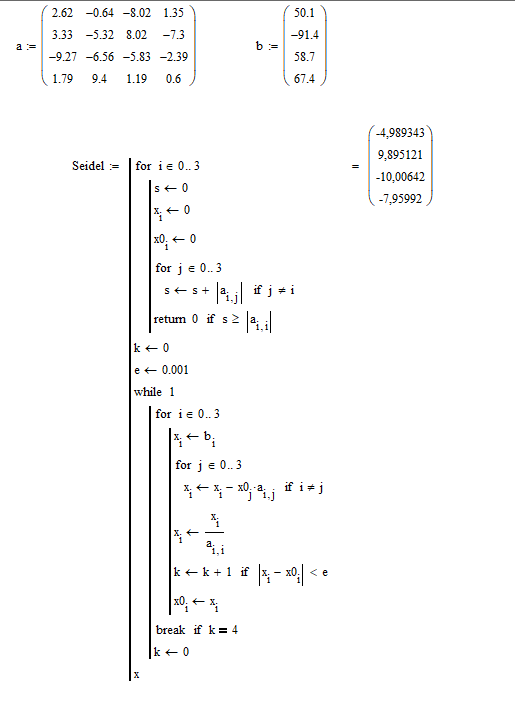
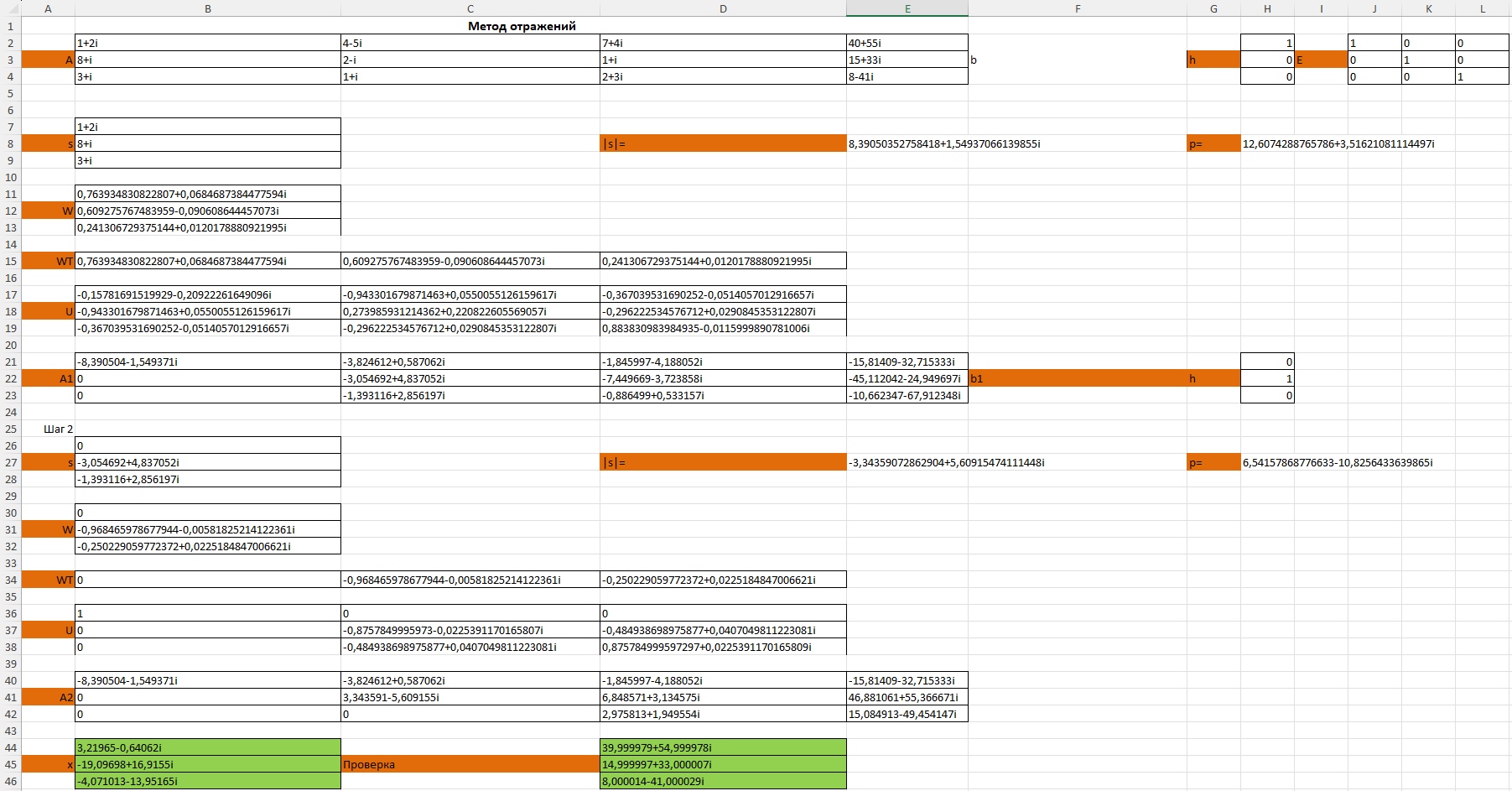


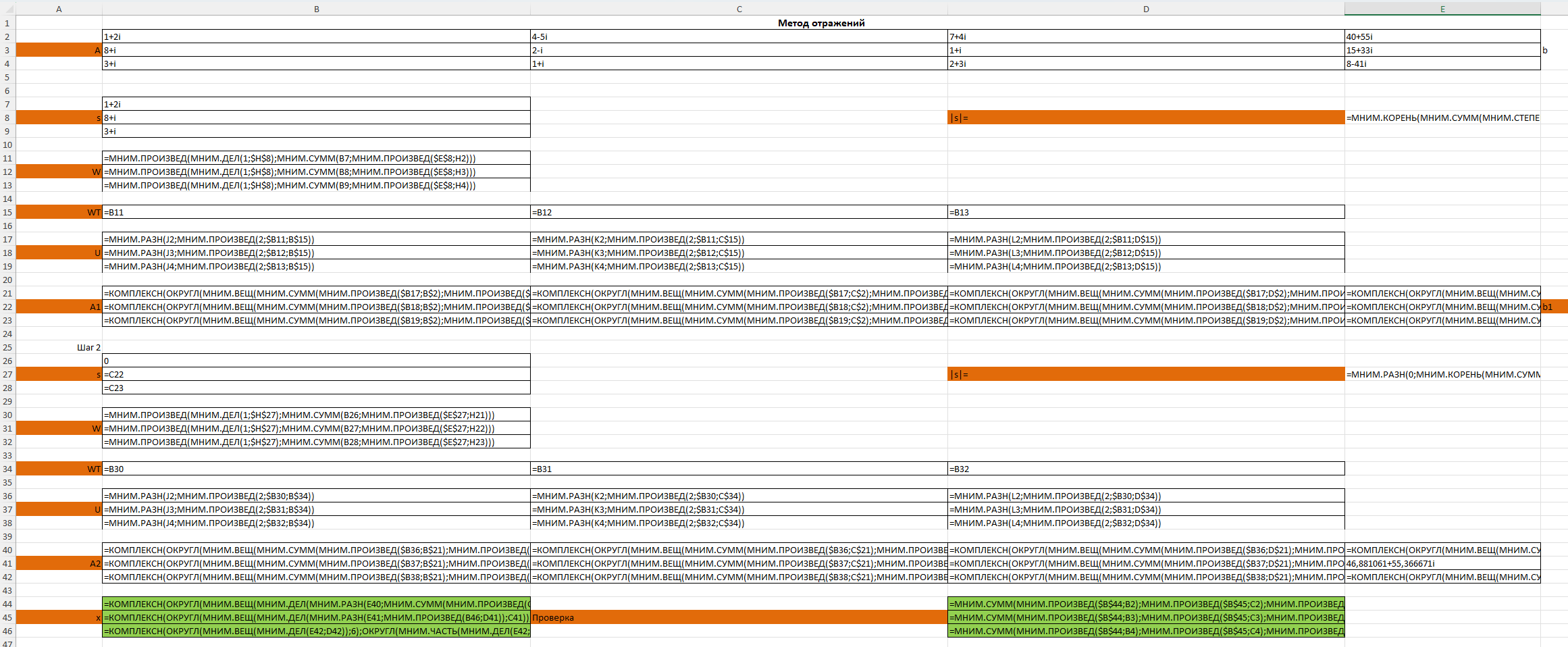
Рисунок 6.- Метод Зейделя

# Реализация задачи в Microsoft Office Excel

## Задание 1

На рисунках 28-29 представлен контроль ручного счета метода отражений в Microsoft Office Excel

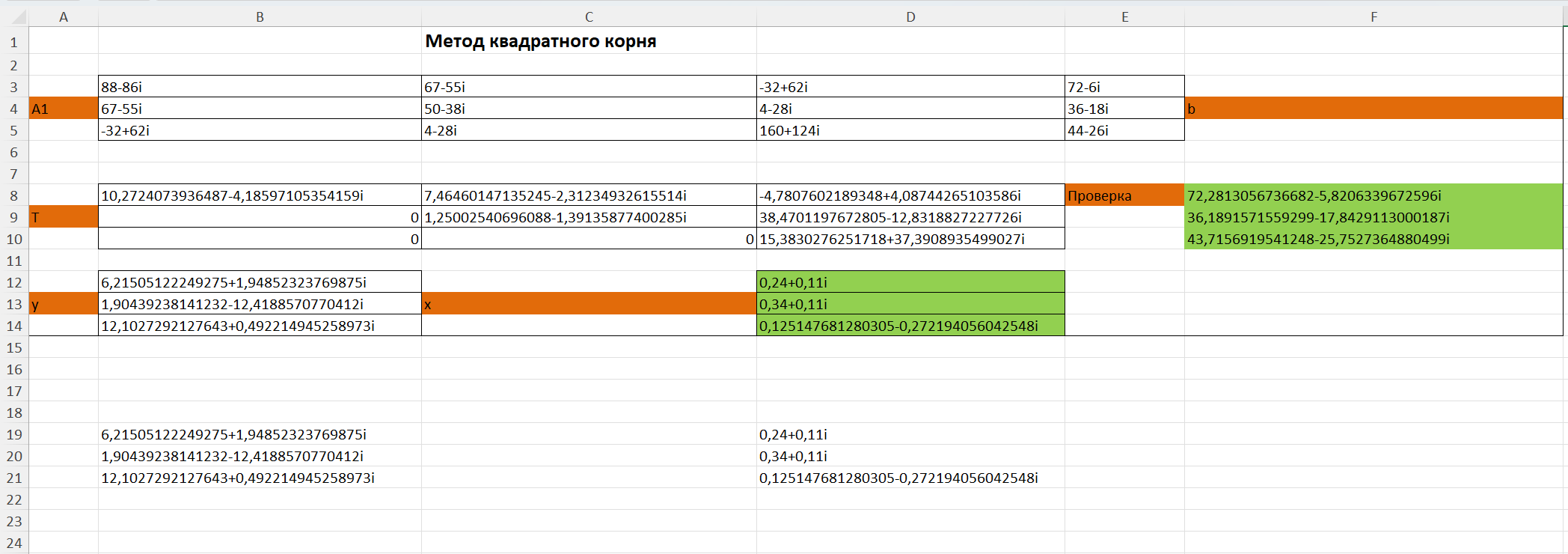
 Рисунок 7 - Контроль ручного счета метода отражений в Excel без формул

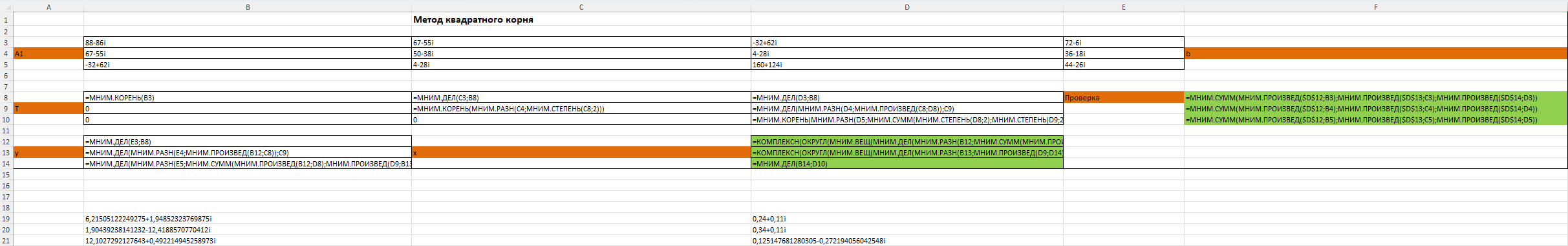


*Рисунок 8- Контроль ручного счета метода отражений в Excel с формулами*

## Задание 2

На рисунках 9-10 представлен контроль ручного счета метода квадратного корня в Microsoft Office Excel

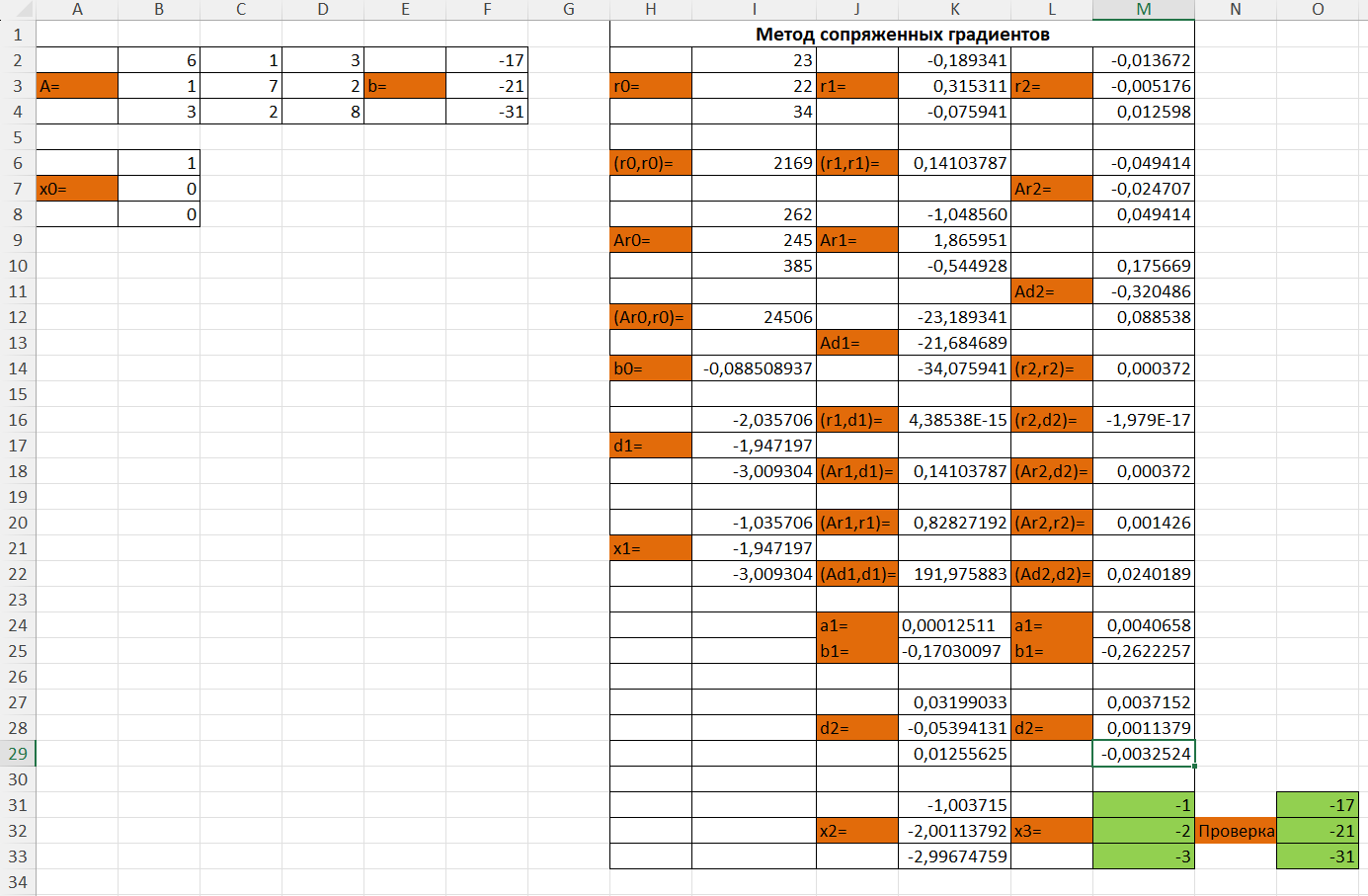
 Рисунок 9 - Контроль ручного счета метода квадратного корня в Excel без формулам



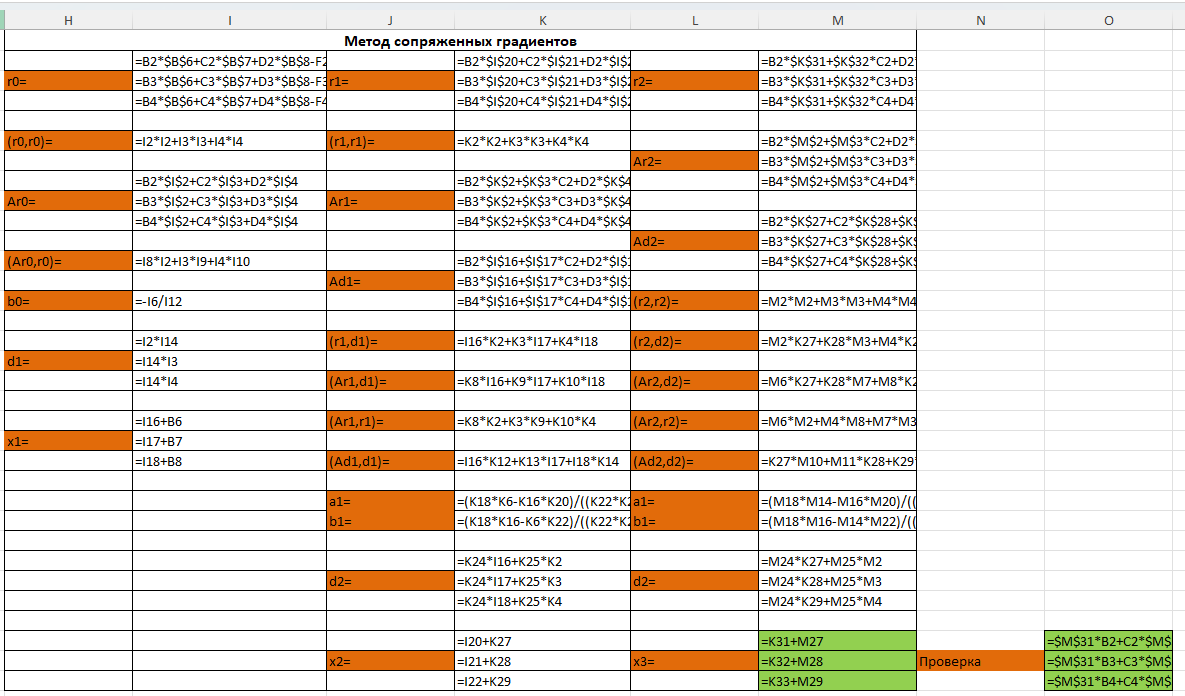
*Рисунок 10 - Контроль ручного счета метода квадратного корня в Excel с формулами*

## Задание 3

На рисунках 11-12 представлен контроль ручного счета метода сопряженных градиентов в Microsoft Office Excel



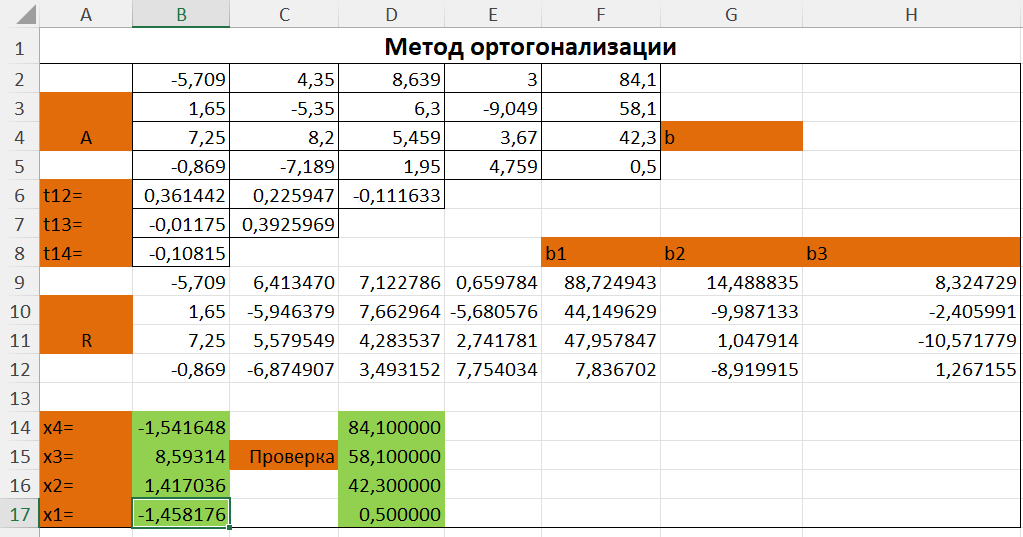
*Рисунок 11– Контроль ручного счета метода сопряженных градиентов в Excel без формул*



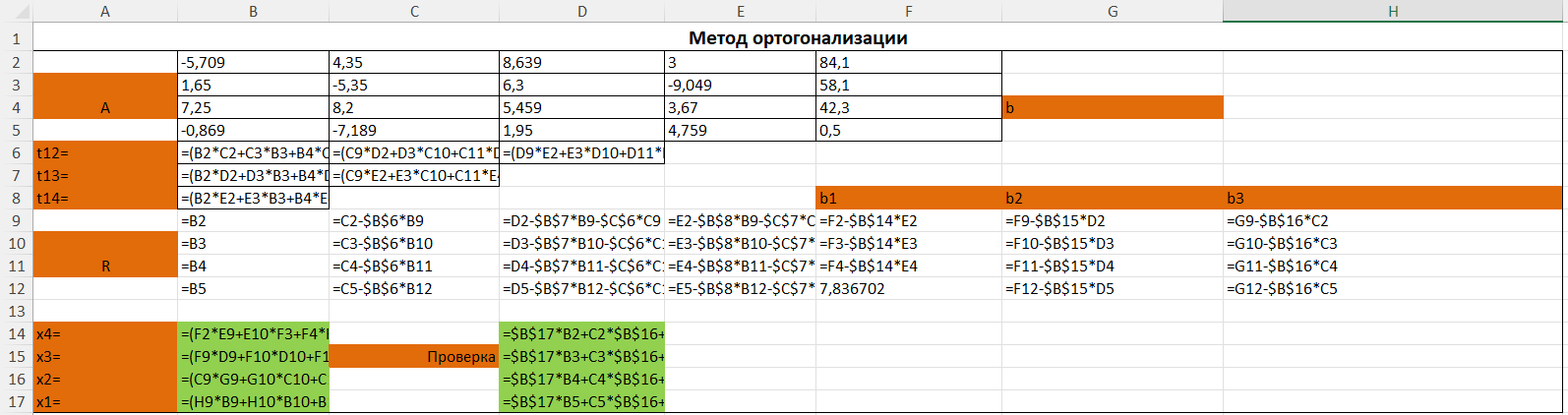
*Рисунок 12- Контроль ручного счета метода сопряженных градиентов в Excel с формулами*

## Задание 4

На рисунках 13-14 представлен контроль ручного счета метода ортогонализации в Microsoft Office Excel



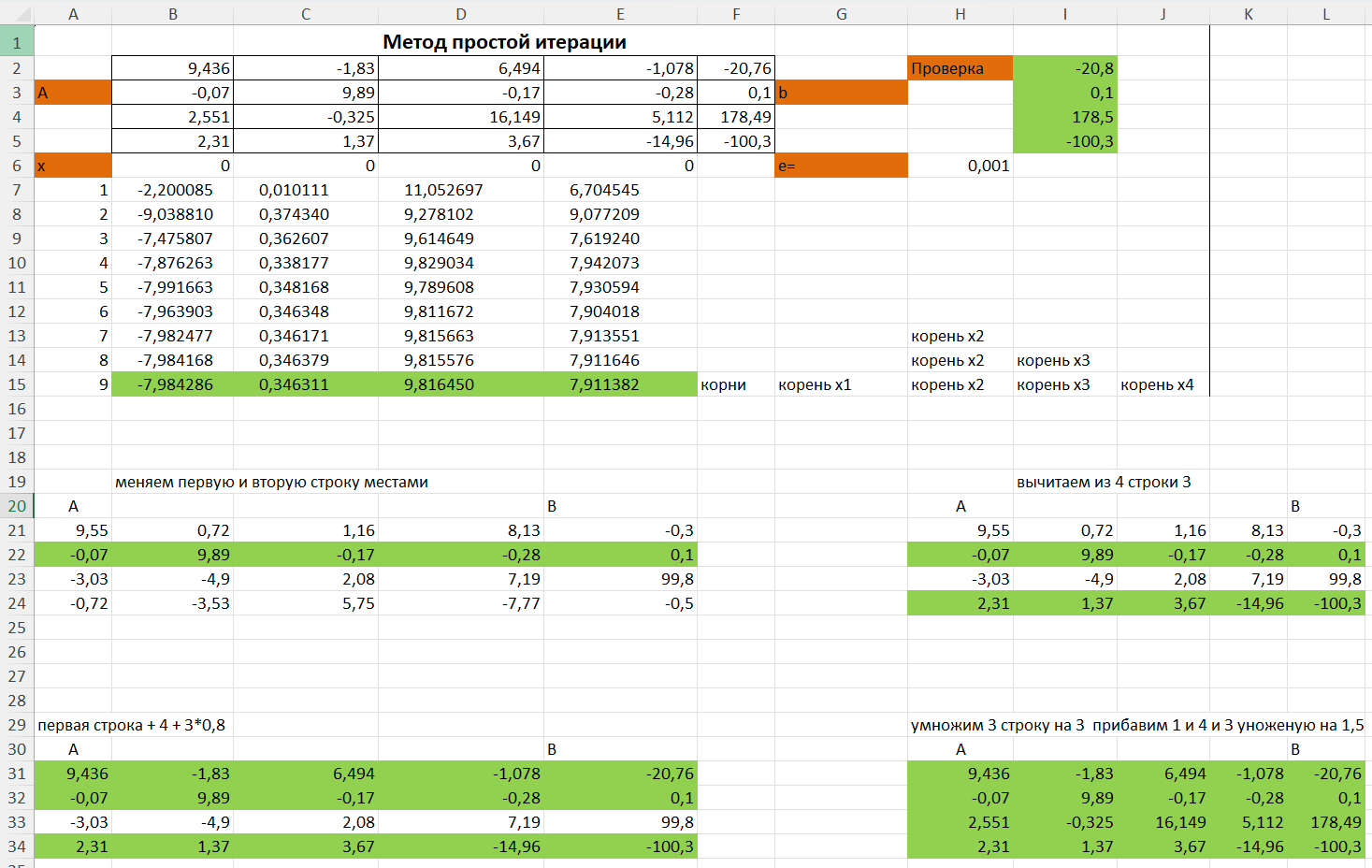
*Рисунок 13 - Контроль ручного счета метода ортогонализации в Excel без формул*



*Рисунок 14 - Контроль ручного счета метода ортогонализации в Excel с формулами*

## Задание 5

На рисунках 15-16 представлен контроль ручного счета метода простой итерации в Microsoft Office Excel

 Рисунок 15 - Контроль ручного счета метода простой итерации в Excel без формул

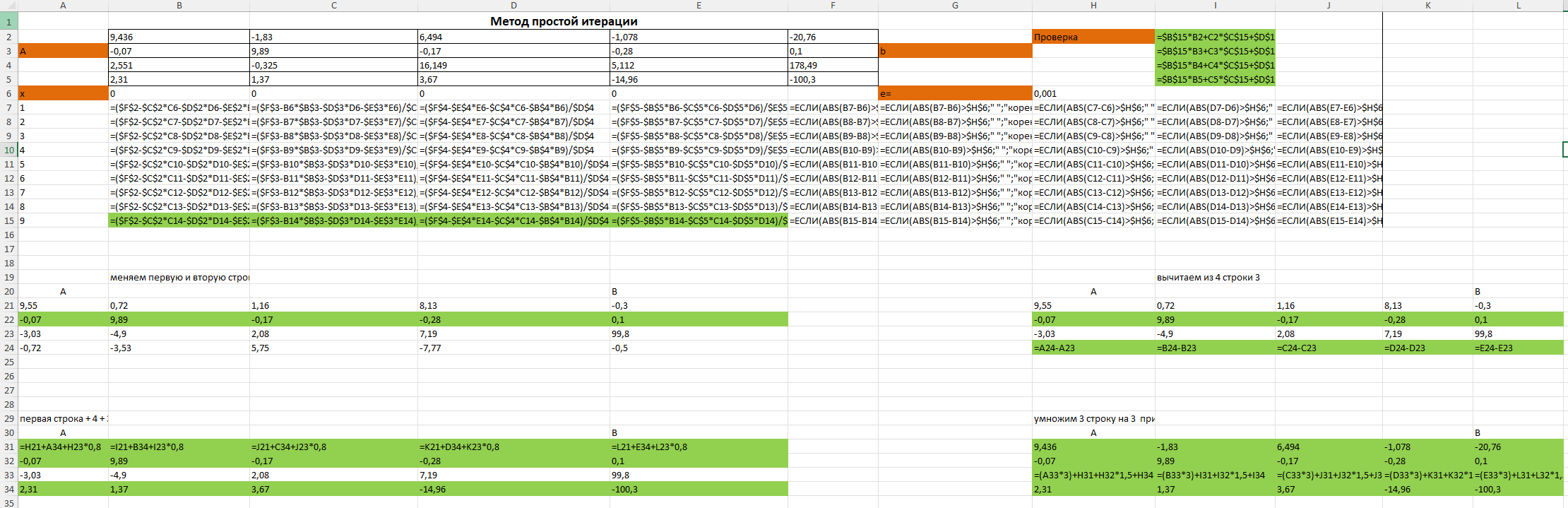
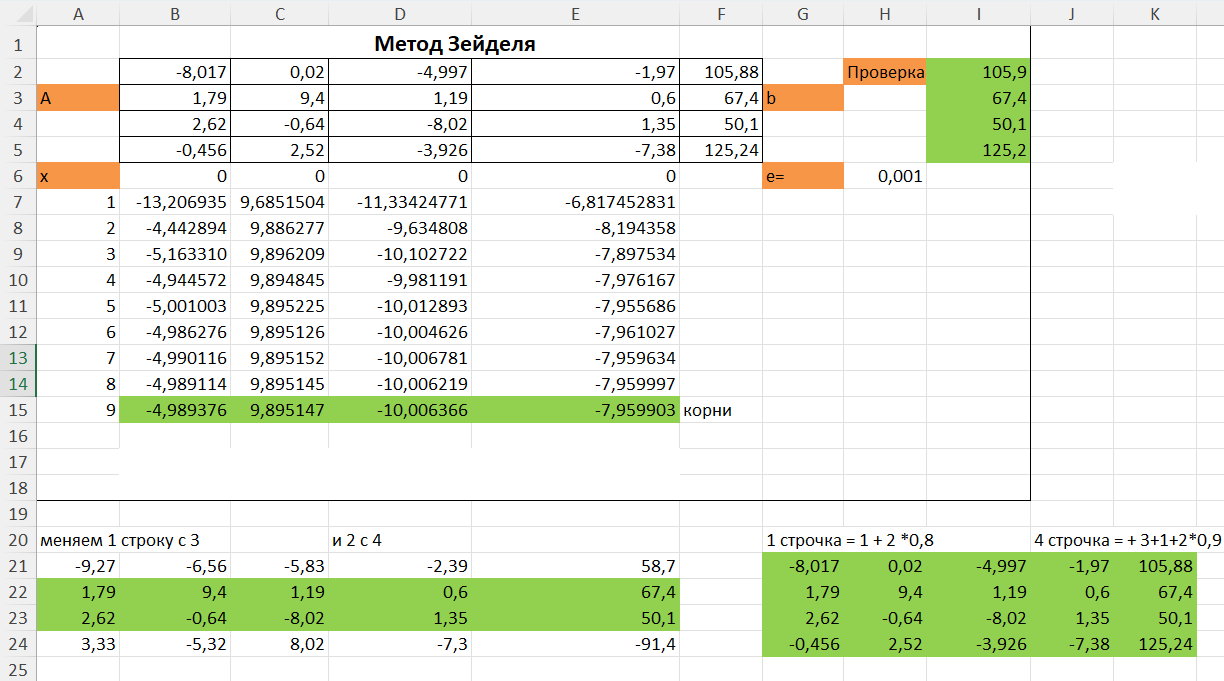


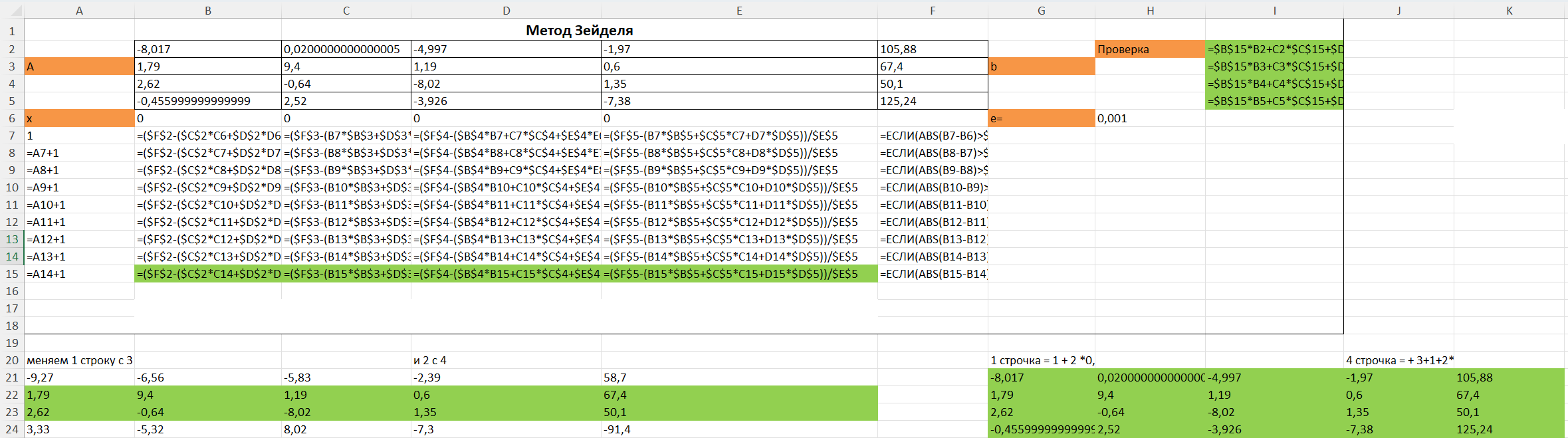
Рисунок 16 - Контроль ручного счета метода простой итерации в Excel с формулами

## Задание 6

На рисунках 17-18 представлен контроль ручного счета метода Зейделя в Microsoft Office Excel



*Рисунок 17 - Контроль ручного счета метода Зейделя в Excel без формул*



*Рисунок 18- Контроль ручного счета метода Зейделя в Excel с формулами*

# 7. Реализация на С++

***7.1 Меню***

Главный класс для пользователя, в котором выбирается нужный метод решения системы линейных алгебраических уравнений.

#include <iostream>

#include <math.h>

#include <cmath>

#include <complex>

#include "gradient.h"

#include "SQRT.h"

#include "Easy.h"

#include "Orthogonalization.h"

#include "Reflections.h"

#include "Zeidel.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

int choice;

int exit = 0;

do {

cout << "Программа решает СЛАУ. Выберите метод, которым хотите решить (введите номер метода):\n";

cout << "1.Метод сопряжённых градиентов.\n";

cout << "2.Метод ортогонализации.\n";

cout << "3.Метод отражений.\n";

cout << "4.Метод квадратных корней.\n";

cout << "5.Метод простой итерации.\n";

cout << "6.Метод Зейделя.\n";

cout << "7.Завершение работы.\n\n";

cout << "Введите число:";

cin >> choice;

// Правильность выбора метода

while (!(choice == 1 || choice == 2 || choice == 3 || choice == 4 || choice == 5 || choice == 6 || choice == 7)) {

cout << "Выберите числа от 1 до 7!\n";

cin >> choice;

//Проверка на число

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> choice;

}

}

switch (choice) {

case 1:

cout << "Вы выбрали метод сопряжёных градиентов.\n";

gradient gradient;

gradient.main();

break;

case 2:

cout << "Вы выбрали метод ортогонализации.\n";

orthogonalization orthogonalization;

orthogonalization.main();

break;

case 3:

cout << "Вы выбрали метод отражений.\n";

reflections reflections;

reflections.main();

break;

case 4:

cout << "Вы выбрали метод квадратных корней.\n\n";

SQRT SQRT;

SQRT.main();

break;

case 5:

cout << "Вы выбрали метод простой итерации.\n\n";

Easy Easy;

Easy.main();

break;

case 6:

cout << "Вы выбрали метод Зейделя.\n\n";

Zeidel Zeidel;

Zeidel.main();

break;

case 7:

cout << "Завершение работы.\n\n";

exit = 1;

break;

}

} while (exit == 0);

return 0; // Добавлено для явного возврата из функции main

}

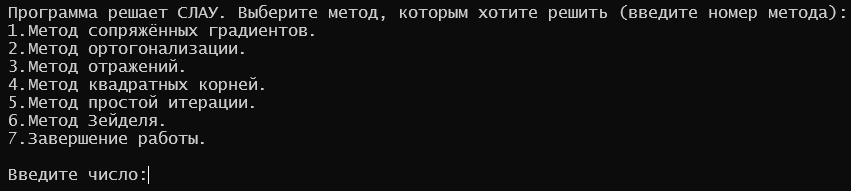


Рисунок 19.-Приветствие и выбор действия в программе в C++

***7.2 Метод квадратных корней***

Класс с реализацией решения системы линейных алгебраических уравнений методом квадратных корней.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

#include <complex>

using namespace std;

class SQRT {

public:

typedef complex<double> Complex;

// Функция приведения СЛАУ к симметричному виду

void symmetricalSLAU(Complex\*\* A, Complex\* B, int n, Complex\*\* a, Complex\* b) {

// Создание транспонированной А

Complex\*\* Atrans = new Complex \* [n];

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

Atrans[i] = new Complex[n];

}

// Получение транспонированной матрицы

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

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

Atrans[j][i] = A[i][j];

}

}

/\*

// Вывод A транспонированной

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

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

cout << Atrans[i][j] << " ";

}

cout << endl;

}

cout << endl;

\*/

// Получение матрицы а= А транспонированная на А

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

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

a[i][j] = 0.0;

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

a[i][j] += Atrans[i][k] \* A[k][j];

}

}

}

cout << "Для решения приведём матрицу А к симметричному виду\n";

// Вывод a

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

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

cout << a[i][j] << " ";

}

cout << endl;

}

cout << endl;

// Получение матрицы b= А транспонированная на B

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

b[i] = 0;

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

b[i] += Atrans[i][j] \* B[j];

}

}

cout << "Делаем преобразования и для матрицы B\n";

// Вывод b

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

cout << b[j] << " ";

}

cout << endl;

}

// Решение методом квадратных корней

void squareRootMeth(int n, Complex\*\* a, Complex\* b, Complex\* X) {

// Создание матрицы U

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

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

{

U[i] = new Complex[n];

}

// Заполнение U нулями

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

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

U[i][j] = 0;

}

}

// Создание и заполнение Y нулями

Complex\* Y = new Complex[n];

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

Y[i] = 0;

}

// Вычисление матрицы U

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

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

Complex sum = 0;

if (j == i) {

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

sum += pow(U[j][k], 2);

}

U[j][j] = sqrt(a[j][j] - sum);

}

else {

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

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

}

U[i][j] = (a[i][j] - sum) / U[j][j];

}

}

// Решаем уравнение Ut\*y=b

Complex sum = 0;

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

sum += U[i][k] \* Y[k];

}

Y[i] = (b[i] - sum) / U[i][i];

}

cout << endl;

// Вывод U

cout << "Матрица U\n";

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

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

cout << U[j][i] << " ";

}

cout << endl;

}

cout << endl;

// Вывод U транспонированной

cout << "Матрица Utrans\n";

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

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

cout << U[i][j] << " ";

}

cout << endl;

}

cout << endl;

// Вывод Y

cout << "Матрица Y\n";

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

cout << "Y[" << j + 1 << "]=" << Y[j] << "\n";

}

cout << endl;

// Решаем уравнение U\*x=y

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

Complex sum = 0;

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

sum += U[k][i] \* X[k];

}

X[i] = (Y[i] - sum) / U[i][i];

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n = 3;

/\*

cout << "Введите количество неизвестных: ";

cin >> n;

// Проверка на ввод числа

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

\*/

// Создание двумерного массива коэффициентов при х системы

Complex\*\* massA = new Complex \* [n];

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

massA[i] = new Complex[n];

}

// Создание преобразованного двумерного массива коэффициентов при х системы

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

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

a[i] = new Complex[n];

}

// Создание нуных массивов

Complex\* massB = new Complex[n];

Complex\* b = new Complex[n];

Complex\* X = new Complex[n];

// Ввод элементов массивов с клавиатуры

/\*

cout << "Введите коэффициенты матрицы A в формате n n:" << endl;

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

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

cout << "A[" << i << "][" << j << "]: ";

double real, imag;

cin >> real >> imag;

A[i][j] = Complex(real, imag);

}

}

\*/

/\*

cout << "Введите коэффициенты вектора B в формате n n:" << endl;

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

cout << "B[" << i << "]: ";

double real, imag;

cin >> real >> imag;

massB[i] = Complex(real, imag);

}

\*/

// Вариант 7

massA[0][0] = Complex(1, 0);

massA[0][1] = Complex(3, -4);

massA[0][2] = Complex(11, 5);

massA[1][0] = Complex(3, 4);

massA[1][1] = Complex(-3, 0);

massA[1][2] = Complex(7, 1);

massA[2][0] = Complex(11, -5);

massA[2][1] = Complex(7, -1);

massA[2][2] = Complex(-4, 0);

massB[0] = Complex(48, 31);

massB[1] = Complex(28, 13);

massB[2] = Complex(7, -31);

symmetricalSLAU(massA, massB, n, a, b);

squareRootMeth(n, a, b, X);

cout << "Решение СЛАУ:" << endl;

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

cout << "X[" << i + 1 << "]: " << X[i] << endl;

}

// Освобождение памяти

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

{

delete[] massA[i];

}

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

{

delete[] a[i];

}

delete[] massA;

delete[] a;

delete[] massB;

delete[] b;

delete[] X;

return 0;

}

};

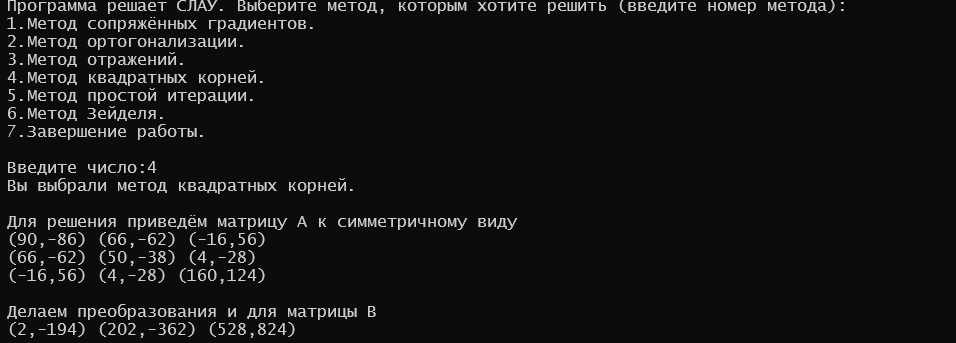


Рисунок 20.-Решение систему уравнений методом квадратных корней в C++

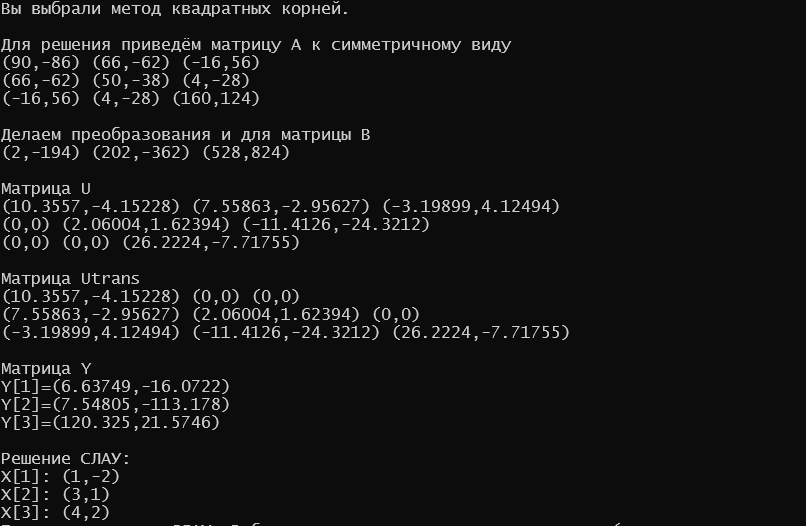


Рисунок 21.-Решение систему уравнений методом квадратных корней в C++

***7.3 Метод сопряжённых градиентов***

Класс с реализацией решения системы линейных алгебраических уравнений методом сопряжённых градиентов.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

using namespace std;

class gradient {

public:

// Скалярное произведение

double vectorMulti(double\* vector1, double\* vector2, int n) {

double result = 0;

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

result += vector1[i] \* vector2[i];

}

return result;

}

// Произведение матрицы на вектор

void matMultVector(double\*\* matrix, double\* vector, int n, double\* result) {

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

result[i] = 0;

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

result[i] += matrix[i][j] \* vector[j];

}

}

}

// Векторное произведение

void vecMultVector(double\* vector1, double\* vector2, int n, double\* result) {

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

result[i] = vector1[i] \* vector2[i];

}

}

// Вычитание векторов

void vecMinVector(double\* vector1, double\* vector2, int n, double\* result) {

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

result[i] = vector1[i] - vector2[i];

}

}

//Вычитание матриц

void matMinMatrix(double\*\* matrix, double num, int n, double\*\* result) {

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

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

result[i][j] = matrix[i][j] \* num;

}

}

}

// Вектор умножить на число

void vecMultNum(double\* vector, double num, int n, double\* result) {

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

result[i] = vector[i] \* num;

}

}

// Сложение векторов

void vecPlusVector(double\* vector1, double\* vector2, int n, double\* result) {

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

result[i] = vector1[i] + vector2[i];

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n = 3;

/\*

cout << "Введите количество неизвестных\n";

cin >> n;

// Проверка на ввод числа

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

\*/

// Объявление используемых переменных

double al, bt, rr, Arr, rDelta, ArDelta, Adeltadelta, e;

double E = 0.01;

double\* r = new double[n];

double\* delta = new double[n];

double\* x = new double[n];

double\* Ax = new double[n];

double\* Ar = new double[n];

double\* Adelta = new double[n];

// Создание двумерного массива коэффициентов при х системы

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

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

{

massA[i] = new double[n];

}

/\*

// Заполнение матрицы

cout << "Введите матрицу А (" << n << "х" << n << "):\n";

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

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

cout << "Введите элемент [" << i + 1 << "][" << j + 1 << "]: ";

cin >> massA[i][j];

// Проверка на ввод значения

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод A[" << i + 1 << "][" << j + 1 << "]: ";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massA[i][j];

}

}

}

\*/

// Создание массива значений

double\* massB = new double[n];

/\*

// Ввод матрицы B

cout << "Введите матрицу В(" << n << "х1):\n";

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

cout << "Введите элемент [" << i + 1 << "]: ";

cin >> massB[i];

// Проверка на ввод значения

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод B[" << i + 1 << "]";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massB[i];

}

}

\*/

// Вариант 7

massA[0][0] = 6;

massA[0][1] = 1;

massA[0][2] = 3;

massA[1][0] = 1;

massA[1][1] = 7;

massA[1][2] = 2;

massA[2][0] = 3;

massA[2][1] = 2;

massA[2][2] = 8;

massB[0] = -17;

massB[1] = -21;

massB[2] = -31;

// первая итерация

cout << "Шаг 1\n";

x[0] = 1;

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

x[i] = 0;

}

matMultVector(massA, x, n, Ax);

vecMinVector(Ax, massB, n, r);

rr = vectorMulti(r, r, n);

cout << "(r0,r0)=" << rr << endl;

matMultVector(massA, r, n, Ar);

Arr = vectorMulti(Ar, r, n);

cout << "(Ar0,r0)=" << Arr << endl;

bt = -(rr / Arr);

cout << "beta0=" << bt << endl;

vecMultNum(r, bt, n, delta);

vecPlusVector(x, delta, n, x);

cout << "x1 =" << endl;

for (int i = 0; i < 3; i++) {

cout << "x1[" << i + 1 << "]=" << x[i] << "\n";

} // Конец первой итерации

int count = 2;

// Цикл следующих итераций

do {

cout << endl << "Шаг " << count << endl;

matMultVector(massA, x, n, Ax);

vecMinVector(Ax, massB, n, r);

matMultVector(massA, r, n, Ar);

matMultVector(massA, delta, n, Adelta);

rr = vectorMulti(r, r, n);

cout << "(r" << count - 1 << ",r" << count - 1 << ")=" << rr << endl;

rDelta = vectorMulti(r, delta, n);

cout << "(r" << count - 1 << ",delta" << count - 1 << ")=" << rDelta << endl;

ArDelta = vectorMulti(Ar, delta, n);

cout << "(r" << count - 1 << ",Ardelta" << count - 1 << ")=" << ArDelta << endl;

Arr = vectorMulti(Ar, r, n);

cout << "(Ar" << count - 1 << ",r" << count - 1 << ")=" << Arr << endl;

Adeltadelta = vectorMulti(Adelta, delta, n);

cout << "(Adelta" << count - 1 << ",delta" << count - 1 << ")=" << Adeltadelta << endl;

al = (((rr \* ArDelta) - (rDelta \* Arr)) / ((Adeltadelta \* Arr) - (ArDelta \* ArDelta)));

cout << "alfa" << count - 1 << "=" << ArDelta << endl;

bt = (((rDelta \* ArDelta) - (rr \* Adeltadelta)) / ((Adeltadelta \* Arr) - (ArDelta \* ArDelta)));

cout << "beta" << count - 1 << "=" << ArDelta << endl;

vecMultNum(delta, al, n, delta);

vecMultNum(r, bt, n, r);

vecPlusVector(delta, r, n, delta);

vecPlusVector(x, delta, n, x);

// Вывод результата по итерации

cout << endl << "Результат " << count << " итерации\n";

for (int i = 0; i < 3; i++) {

cout << "x" << count - 1 << "[" << i + 1 << "]=" << x[i] << endl;

}

// Проверка на достижение точности

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

if (fabs(delta[i]) > fabs(delta[i - 1])) {

e = fabs(delta[i]);

}

else { e = fabs(delta[i - 1]); }

}

cout << endl;

count++;

} while (e > E);

cout << "Результат:\n";

for (int i = 0; i < 3; i++) {

cout << "x[" << i + 1 << "]=" << x[i] << endl;

}

cout << endl;

return 0;

}

};

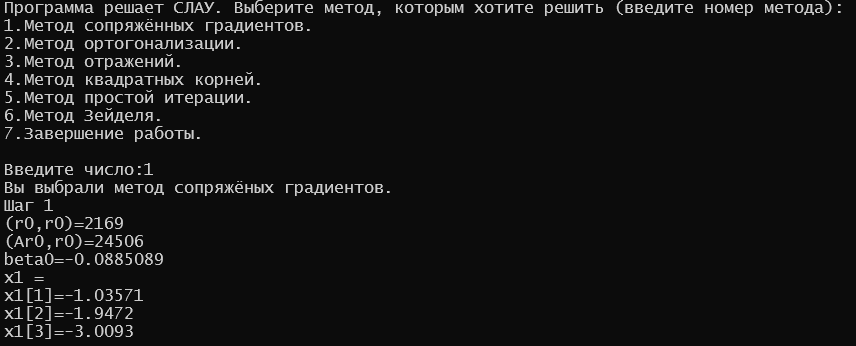


Рисунок 22.-Решение системы уравнений методом сопряжённых градиентов в C++

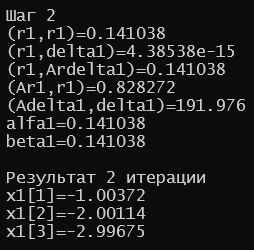


Рисунок 23.-Решение системы уравнений методом сопряжённых градиентов в C++

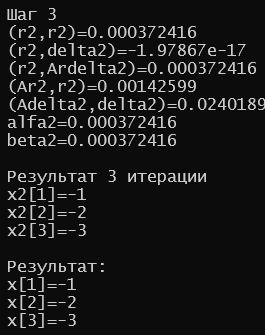


Рисунок 24.-Решение системы уравнений методом сопряжённых градиентов в C++

***7.4 Метод отражений***

Класс с реализацией решения системы линейных алгебраических уравнений методом отражений.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

#include <complex>

#include <math.h>

#include <cmath>

using namespace std;

class reflections {

public:

//Матричное произведение

void matMultMat(complex<double>\*\* matrix1, int n1, int m1, complex<double>\*\* matrix2, int n2, int m2, complex<double>\*\* result) {

if (m1 != n2) {

cout << "Размеры матриц не соответствуют условию умножения (количество столбцов первой не соответствуют количеству строк второй матрицы)";

return;

}

complex <double>\*\* tempm = new complex <double>\*[n2];

for (int i = 0; i < n2; i++)

{

tempm[i] = new complex <double>[m2];

for (int j = 0; j < m2; j++) {

tempm[i][j] = matrix2[i][j];

}

}

for (int i = 0; i < n1; i++) {

for (int j = 0; j < m2; j++) {

complex<double> sum = (0.0, 0.0); //0.0

for (int k = 0; k < m1; k++) {

sum += matrix1[i][k] \* tempm[k][j];

}

result[i][j] = sum;

}

}

}

//Скалярное произведение векторов

complex<double> sqalarVec(complex<double>\* vector1, complex<double>\* vector2, int n) {

complex<double> result = (0.0, 0.0);

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

result += vector1[i] \* vector2[i];

}

return result;

}

//Векторное произведение векторов

void vecMultVec(complex<double>\* vector1, complex<double>\* vector2, int n, complex<double>\*\* result) {

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

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

result[i][j] = vector1[i] \* vector2[j];

}

}

}

//Матрица минус вектор

void matMinVec(complex<double>\*\* matrix, complex<double>\* vector, int n, int m, complex<double>\*\* result) {

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

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

result[i][j] = matrix[i][j] - vector[j];

}

}

}

//Матрица минус матрица

void matMinMat(complex<double>\*\* mat1, complex<double>\*\* mat2, int n, complex<double>\*\* result) {

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

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

result[i][j] = mat1[i][j] - mat2[i][j];

}

}

}

//Вектор минус вектор

void vecMinVec(complex<double>\* vector1, complex<double>\* vector2, int n, complex<double>\* result) {

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

result[i] = vector1[i] - vector2[i];

}

}

//Вектор плюс вектор

void vecPlusVec(complex<double>\* vector1, complex<double>\* vector2, int n, complex<double>\* result) {

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

result[i] = vector1[i] + vector2[i];

}

}

//Вектор умноженный на число

void vecMultNum(complex<double>\* vector, int n, complex<double> num, complex<double>\* result) {

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

result[i] = vector[i] \* num;

}

}

//Матрица умноженная на число

void matMultMat(complex<double>\*\* mat, int n, complex<double> num, complex<double>\*\* result) {

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

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

result[i][j] = mat[i][j] \* num;

}

}

}

//Вырезка столбца матрицы в вектор

void rowVec(complex<double>\*\* mat, int n, int col, complex<double>\* result) {

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

result[i] = mat[i][col];

}

}

//Вырезка строки матрицы в вектор

void colVec(complex<double>\*\* mat, int m, int row, complex<double>\* result) {

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

result[i] = mat[row][i];

}

}

int main() {

complex <double> al, ro, sL, sqrtal;

int n, m, sign;

n = 3;

m = 4;

const double PI = acos(-1.0);

/\*

cout << "Введите разрядность матрицы А: строки и столбцы \n";

cin >> n;

while (!cin) {

cout << "Вы ввели не число! Повторите ввод.\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

cin >> m;

while (!cin) {

cout << "Вы ввели не число! Повторите ввод.\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> m;

}

cout << "Введите матрицу А с комплексными числами в формате (x, y)\n";

\*/

complex <double>\*\* massA = new complex <double>\*[n];

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

{

massA[i] = new complex <double>[m];

}

/\*

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

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

cout << "A[" << i << "][" << j << "] = ";

cin >> A[i][j];

while (!cin) {

cout << "Вы ввели не число! Повторите ввод A[" << i << "][" << j << "]=";

cin.clear();

while (cin.get() != '\n') continue;

cin >> A[i][j];

}

}

cout << "\n";

}

\*/

//Вариант 10

massA[0][0] = complex <double>(1, 2);

massA[0][1] = complex <double>(4, -5);

massA[0][2] = complex <double>(7, 4);

massA[0][3] = complex <double>(40, 67);

massA[1][0] = complex <double>(8, 1);

massA[1][1] = complex <double>(2, -1);

massA[1][2] = complex <double>(1, 1);

massA[1][3] = complex <double>(15, 33);

massA[2][0] = complex <double>(3, 1);

massA[2][1] = complex <double>(1, 1);

massA[2][2] = complex <double>(2, 3);

massA[2][3] = complex <double>(8, -41);

cout << "Ваша матрица имеет вид\n";

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

for (int t = 0; t < m; t++) {

cout << massA[i][t];

}

cout << endl;

}

complex <double>\*\* E = new complex <double>\*[n];

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

{

E[i] = new complex <double>[m];

}

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

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

if (i == j) { E[i][i] = 1; } //complex <double>(1.0, 1.0)

else { E[i][j] = 0; }

}

}

complex <double>\*\* U = new complex <double>\*[n];

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

{

U[i] = new complex <double>[m];

}

complex <double>\*\* W = new complex <double>\*[n];

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

{

W[i] = new complex <double>[m];

}

complex <double>\* x = new complex <double>[n];

complex <double>\* s = new complex <double>[n];

complex <double>\* w = new complex <double>[n];

complex <double>\* L = new complex <double>[n];

complex <double>\* r = new complex <double>[m];

bool absal;

for (int i = 0; i < n - 1; i++) {

cout << endl << "Шаг " << i + 1 << endl;

absal = false;

rowVec(massA, n, i, s);

if (i > 0) {

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

s[k] = complex <double>(0.0, 0.0);

}

}

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

//cout << "s[" << k << "]=" << s[k] << "\n";

}

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

if (j == i) { L[j] = 1; } //complex <double>(1.0, 1.0)

else { L[j] = 0; }

}

al = sqalarVec(s, s, n);

//cout << "al1 = " << al << ", ro = " << ro << endl;

sqrtal = sqrt(al);

//cout << "sqrtal1 = " << sqrtal << ", ro = " << ro << endl;

sL = sqalarVec(s, L, n);

//cout << "sL=" << sL << endl;

if (arg(sL) - PI < 0) {

absal = true;

//cout << "arg(sqrtal) = " << arg(sqrtal) << " arg(sL) = " << arg(sL) << "arg(abs(sL))=" << arg(abs(sL)) << " arg(sL) - Pi = " << arg(sL) - 3.14 << "\n";

}

if (i % 2 != 0) {

sqrtal \*= -1; /\*cout << "sqrtal-1 = " << sqrtal << "\n"; \*/

}

ro = sqrt(2.0 \* al + 2.0 \* sqrtal \* sL);

//cout << "al = " << al << ", ro = " << ro << endl;

ro = pow(ro, -1);

//cout << "sqrtal = " << sqrtal << ", ro2 = " << ro << endl;

vecMultNum(L, n, sqrtal, L);

vecPlusVec(s, L, n, w);

vecMultNum(w, n, ro, w);

/\*

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

cout << "s[" << k << "] = " << s[k] << "\n";

}

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

cout << "L[" << k << "] = " << L[k] << "\n";

}

\*/

cout << "Вектор w:\n";

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

cout << w[k] << "\n";

}

vecMultVec(w, w, n, W);

matMultMat(W, n, 2, W);

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

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

//cout << "W[" << k << "][" << t << "] = " << W[k][t] << endl;

}

}

matMinMat(E, W, n, U);

matMultMat(U, n, n, massA, n, m, massA);

cout << "Вектор U" << i << ":\n";

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

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

cout << U[k][t];

}

cout << endl;

}

cout << "Вектор A" << i + 1 << ":\n";

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

for (int t = 0; t < m; t++) {

cout << massA[k][t];

}

cout << endl;

}

// break;

}

complex <double> sum = complex <double>(0, 0);

x[n - 1] = massA[n - 1][m - 1] / massA[n - 1][m - 2];

for (int i = n - 2; i > -1; i--) {

colVec(massA, m, i, r);

for (int k = 0; k < m; k++) {

//cout << "r[" << k << "] = " << r[k] << "\n";

}

sum = complex <double>(0, 0);

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

sum += x[j] \* r[j];

//cout << "sum = " << sum << "\n";

//cout << "x[j] \* r[j] = " << x[j] \* r[j] << "\n";

}

x[i] = (r[m - 1] - sum) / (r[i]);

//cout << "r[i] = " << r[i] << ", r[m] = " << r[m - 1] << "\n";

//cout << "x[i] = " << x[i] << "\n";

}

cout << "\nРезультат вычисления:\n";

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

cout << "x[" << i + 1 << "] = " << x[i] << "\n";

}

return 0;

}

};

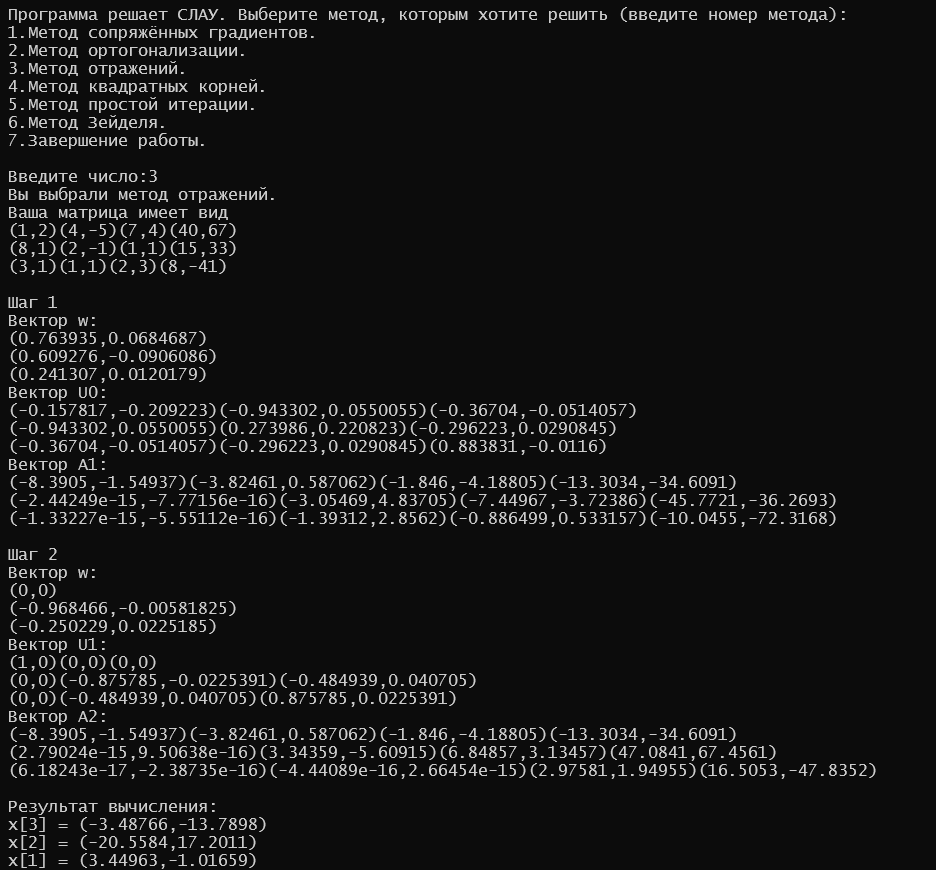


Рисунок 25.-Решение системы уравнений методом отражений в C++

***7.5 Метод ортогонализации***

Класс с реализацией решения системы линейных алгебраических уравнений методом ортогонализации.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

#include <math.h>

using namespace std;

class orthogonalization {

public:

//Векторное произведение

double vectorMult(double\* vector1, double\* vector2, int n) {

double result = 0;

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

result += vector1[i] \* vector2[i];

}

return result;

}

// Сумма векторов

void vectorSum(double\* vector1, double\* vector2, int n, double\* result) {

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

result[i] = vector1[i] + vector2[i];

}

}

// Получение вектора из матрицы

void rowVector(double\*\* mass, int n, int row, double\* vectorRow) {

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

vectorRow[i] = mass[row][i];

}

}

// Вектор делить на число

void vectorDivNum(double\* vector, int n, double num, double\* result) {

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

result[i] = -vector[i] / num;

}

}

// Вектор умножить на число

void vectorMultNum(double\* vector, int n, double num, double\* result) {

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

result[i] = vector[i] \* num;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

double vecMult;

int n, il;

n = 5;

/\*cout << "Введите количество неизвестных:\n";

cin >> n;

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

\*/

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

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

u[i] = new double[n];

}

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

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

b[i] = new double[n];

}

double\* c = new double[n];

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

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

a[i] = new double[n];

}

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

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

cb[i] = new double[n];

}

double\* x = new double[n];

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

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

massA[i] = new double[n];

}

/\*cout << "Введите первую матрицу А (после ввода матрицы, введите строку, состоящую из n-1 нулей и 1 в конце строки)\n";

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

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

cout << "Введите элемент [" << i << "][" << j << "]: ";

cin >> massA[i][j];

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод A[" << i << "][" << j << "]: ";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massA[i][j];

}

}

}

\*/

// Вариант 10

massA[0][0] = -5.709;

massA[0][1] = 4.35;

massA[0][2] = 8.639;

massA[0][3] = 3;

massA[0][4] = 84.1;

massA[1][0] = 1.650;

massA[1][1] = -5.350;

massA[1][2] = 6.3;

massA[1][3] = -9.049;

massA[1][4] = 58.1;

massA[2][0] = 7.250;

massA[2][1] = 8.200;

massA[2][2] = 5.459;

massA[2][3] = 3.670;

massA[2][4] = 42.3;

massA[3][0] = -0.869;

massA[3][1] = -7.189;

massA[3][2] = 1.950;

massA[3][3] = 4.759;

massA[3][4] = 0.5;

massA[4][0] = 0;

massA[4][1] = 0;

massA[4][2] = 0;

massA[4][3] = 0;

massA[4][4] = 1;

//Шаг 1

rowVector(massA, n, 0, u[0]);

vecMult = vectorMult(u[0], u[0], n);

vectorDivNum(u[0], n, sqrt(vecMult), b[0]);

//Шаг 2

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

rowVector(massA, n, i, a[i]);

rowVector(massA, n, i, u[i]);

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

c[l] = -vectorMult(a[i], b[l], n);

vectorMultNum(b[l], n, c[l], cb[l]);

if (l > 0) {

vectorSum(cb[l], cb[l - 1], n, cb[l]);

}

il = l;

}

vectorSum(cb[il], u[i], n, u[i]);

vecMult = vectorMult(u[i], u[i], n);

vectorDivNum(u[i], n, sqrt(vecMult), b[i]);

}

vectorDivNum(b[n - 1], n, b[n - 1][n - 1], x);

cout << "Результат:\n";

for (int i = 0; i < n - 1; i++) {

cout << "x[" << i + 1 << "]= " << x[i] << "\n";

}

return 0;

}

};

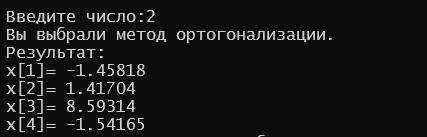


Рисунок 26.-Решение системы уравнений методом ортогонализации в C++

***7.6 Метод простой итерации***

Класс с реализацией решения системы линейных алгебраических уравнений методом простой итерации.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

using namespace std;

class Easy {

public:

// Подсчёт определителя матрицы А

double determinant(double\*\* matrix, int size)

{

double det = 0;

double sign = 1;

// Один элемент

if (size == 1) {

det = matrix[0][0];

}

// Матрица 2х2

else if (size == 2) {

det = (matrix[0][0] \* matrix[1][1])

- (matrix[0][1] \* matrix[1][0]);

}

// Матрица (nхn)

else {

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

// Разложение по строке(столбцу)

double\*\* cofactor = new double\* [size - 1];

for (int j = 0; j < size - 1; j++) {

cofactor[j] = new double[size - 1];

}

int sub\_i = 0, sub\_j = 0;

for (int j = 1; j < size; j++) {

for (int k = 0; k < size; k++) {

if (k == i) {

continue;

}

cofactor[sub\_i][sub\_j] = matrix[j][k];

sub\_j++;

}

sub\_i++;

sub\_j = 0;

}

// Подсчёт значения определителя

det += sign \* matrix[0][i]

\* determinant(cofactor, size - 1);

sign = -sign;

for (int j = 0; j < size - 1; j++) {

delete[] cofactor[j];

}

delete[] cofactor;

}

}

return det;

}

// Приведение к каноническому виду

void canon(double\*\* massA, double\* massB, int n) {

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

double mult = massA[i][i];

// Делим строку на ведущий элемент, чтобы получить 1 на диагонали

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

massA[i][j] /= mult;

}

massB[i] /= mult;

// Обнуляем остальные элементы в столбце

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

if (k != i) {

double factor = massA[k][i];

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

massA[k][j] -= factor \* massA[i][j];

}

massB[k] -= factor \* massB[i];

}

}

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int n, check{ 1 };

string key;

int count = 0;

double delta, eps;

cout << "Введите количество неизвестных:\n";

cin >> n;

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

cout << "Введите точность: " << endl;

cin >> eps;

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> eps;

}

//Выдeление динам. памяти для массива

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

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

{

massA[i] = new double[n];

}

//Выдeление памяти для свободных членов

double\* massB = new double[n];

double\* x = new double[n];

double\* xf = new double[n];

/\*

// Ввод матрицы A

cout << "Введите матрицу А (" << n << "х" << n << "):\n";

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

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

cout << "Введите элемент [" << i << "][" << j << "]: ";

cin >> massA[i][j];

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод A[" << i << "][" << j << "]: ";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massA[i][j];

}

}

}

// Ввод матрицы B

cout << "Введите матрицу В(" << n << "х1):\n";

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

cout << "Введите элемент [" << i << "]: ";

cin >> massB[i];

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод B[" << i << "]";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massB[i];

}

}

// Вывод матрицы A

cout << "Вы ввели матрицу A, размерностью " << n << "х" << n << ":\n";

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

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

cout << massA[i][j] << " ";

}

cout << endl;

}

// Вывод матрицы B

cout << "Вы ввели матрицу B, размерностью " << n << "х1:\n";

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

cout << massB[i] << "\n";

}

\*/

//Вариант 7

massA[0][0] = -0.07;

massA[0][1] = 9.89;

massA[0][2] = -0.17;

massA[0][3] = -0.28;

massA[1][0] = 9.55;

massA[1][1] = -0.72;

massA[1][2] = -1.16;

massA[1][3] = 8.13;

massA[2][0] = -3.03;

massA[2][1] = -4.9;

massA[2][2] = 2.08;

massA[2][3] = 7.19;

massA[3][0] = -0.72;

massA[3][1] = -3.53;

massA[3][2] = 5.75;

massA[3][3] = -7.77;

massB[0] = 0.1;

massB[1] = -0.3;

massB[2] = 99.8;

massB[3] = -0.5;

// Проверка 1 условия сходимости (Определитель матрицы А не равен нулю)

double det = determinant(massA, n);

if (det == 0) {

cout << "Условие 1 не выполняется. Определитель матрицы А=0.\n";

cout << "Решение невозможно!\n";

return 0;

}

// Проверка 2 условия сходимости (Элемент на диагонали по модулю больше суммы оставшихся элементов по модулю)

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

{

double sum = 0;

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

{

if (j != i) sum += fabs(massA[i][j]);

}

if (fabs(massA[i][i]) < fabs(sum))

{

cout << "Условие 2 не выполняется. Вам необходимо вручную привести систему к нужному виду.\n Вы хотите увидеть ответ с приведением к единичной матрице? (y - да, любая клавиша - нет)\n";

cin >> key;

// Исправление под нужное решение

if (key == "y") {

canon(massA, massB, n);

cout << "Приведём систему к виду единичной матрицы для ответа.\n\n";

cout << "Исправленная матрица A\n";

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

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

cout << massA[i][j] << " ";

}

cout << endl;

}

cout << "Исправленная матрица B\n";

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

cout << massB[i] << "\n";

}

cout << "\nРешение вашей системы уравнений имеет вид:\n";

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

cout << "x[" << i + 1 << "]=" << massB[i] << endl;

}

check = 0;

}

else {

check = 0;

}

break;

}

}

if (check != 0) {

// Итераций 0

cout << "Итерация " << count << endl;

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

{

xf[i] = 0;

cout << "x[" << i << "]=" << xf[i] << endl;

}

count = 1;

cout << "Итерация " << count << endl;

// Итерация 1

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

{

xf[i] = massB[i] / massA[i][i];

cout << "x[" << i << "]=" << round(xf[i] \* 10000) / 10000 << endl;

}

count = 2;

// Подсчёт остальных итераций до достижения нужной точности

do {

count++;

cout << "Итерация " << count << endl;

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

{

double sum = 0;

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

// Подсчёт суммы элементов, которые не на диагонали

{

if (j != i) sum += massA[i][j] \* xf[j];

}

x[i] = (massB[i] - sum) / massA[i][i];

cout << "x[" << i << "]=" << round(x[i] \* 10000) / 10000 << endl;

delta = fabs(x[0] - xf[0]);

// Переписывание новой точности

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

{

if (fabs(x[j] - xf[j]) > delta)

delta = fabs(x[j] - xf[j]);

}

xf[i] = x[i];

}

} while (delta > eps);

// Вывод результата

cout << "Достигнута точность:" << delta << "\n";

cout << "Количество итераций:" << count << "\n";

cout << "Результат:\n";

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

{

cout << "x[" << i << "] = " << round(x[i] \* 10000) / 10000 << "\n";

}

// Освобождение памяти

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

{

delete[] massA[i];

}

delete[] massA;

delete[] massB;

delete[] x;

delete[] xf;

};

return 0;

}

};

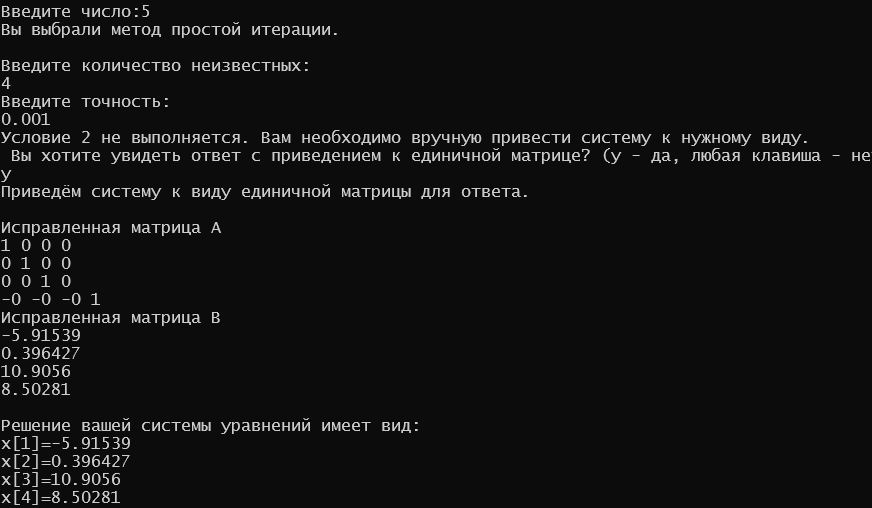


Рисунок 27.-Решение системы уравнений методом простой итерации в C++

***6.7 Метод Зейделя***

Класс с реализацией решения системы линейных алгебраических уравнений методом Зейделя.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

#pragma once

#include <iostream>

class Zeidel {

public:

// Подсчёт определителя матрицы А

double determinant(double\*\* matrix, int size)

{

double det = 0;

double sign = 1;

// Размер матрицы 1х1

if (size == 1) {

det = matrix[0][0];

}

// Размер матрицы 2х2

else if (size == 2) {

det = (matrix[0][0] \* matrix[1][1])

- (matrix[0][1] \* matrix[1][0]);

}

// Размер n\*n

else {

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

// Разложение по строке(столбцу)

double\*\* cofactor = new double\* [size - 1];

for (int j = 0; j < size - 1; j++) {

cofactor[j] = new double[size - 1];

}

int sub\_i = 0, sub\_j = 0;

for (int j = 1; j < size; j++) {

for (int k = 0; k < size; k++) {

if (k == i) {

continue;

}

cofactor[sub\_i][sub\_j] = matrix[j][k];

sub\_j++;

}

sub\_i++;

sub\_j = 0;

}

// Подсчёт значения определителя

det += sign \* matrix[0][i]

\* determinant(cofactor, size - 1);

sign = -sign;

for (int j = 0; j < size - 1; j++) {

delete[] cofactor[j];

}

delete[] cofactor;

}

}

return det;

}

// Приведение к каноническому виду

void canon(double\*\* massA, double\* massB, int n) {

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

double mult = massA[i][i];

// Делим строку на ведущий элемент, чтобы получить 1 на диагонали

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

massA[i][j] /= mult;

}

massB[i] /= mult;

// Обнуляем остальные элементы в столбце

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

if (k != i) {

double factor = massA[k][i];

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

massA[k][j] -= factor \* massA[i][j];

}

massB[k] -= factor \* massB[i];

}

}

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int n, check{ 1 };

string key;

int count = 0;

double delta, eps;

cout << "Введите количество неизвестных:\n";

cin >> n;

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> n;

}

cout << "Введите точность: " << endl;

cin >> eps;

while (!cin)

{

cout << "Введите число!\n";

cin.clear();

while (cin.get() != '\n') continue;

cin >> eps;

}

//Выдeление динам. памяти для массива

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

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

{

massA[i] = new double[n];

}

//Выдeление памяти для свободных членов

double\* massB = new double[n];

double\* x = new double[n];

double\* xf = new double[n];

/\*

// Ввод матрицы A

cout << "Введите матрицу А (" << n << "х" << n << "):\n";

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

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

cout << "Введите элемент [" << i << "][" << j << "]: ";

cin >> massA[i][j];

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод A[" << i << "][" << j << "]: ";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massA[i][j];

}

}

}

// Ввод матрицы B

cout << "Введите матрицу В(" << n << "х1):\n";

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

cout << "Введите элемент [" << i << "]: ";

cin >> massB[i];

while (!cin)

{

cout << "Вы ввели не число! Повторите ввод B[" << i << "]";

cin.clear();

while (cin.get() != '\n') continue;

cin >> massB[i];

}

}

\*/

//Вариант 10

massA[0][0] = 2.62;

massA[0][1] = -0.64;

massA[0][2] = -8.02;

massA[0][3] = 1.35;

massA[1][0] = 3.33;

massA[1][1] = -5.32;

massA[1][2] = 8.02; //

massA[1][3] = -7.3;

massA[2][0] = -9.27;

massA[2][1] = -6.56;

massA[2][2] = -5.83;

massA[2][3] = -2.39;

massA[3][0] = 1.79;

massA[3][1] = 9.4;

massA[3][2] = 1.19;

massA[3][3] = 0.6;

massB[0] = 50.1;

massB[1] = -91.4;

massB[2] = 58.7;

massB[3] = 67.4;

// Вывод матрицы A

cout << "Вы ввели матрицу A, размерностью " << n << "х" << n << ":\n";

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

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

cout << massA[i][j] << " ";

}

cout << endl;

}

// Вывод матрицы B

cout << "Вы ввели матрицу B, размерностью " << n << "х1:\n";

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

cout << massB[i] << "\n";

}

// Проверка 1 условия сходимости (Определитель матрицы А не равен нулю)

double det = determinant(massA, n);

if (det == 0) {

cout << "Условие 1 не выполняется. Определитель матрицы А=0.\n";

cout << "Решение невозможно!\n";

return 0;

}

// Проверка 2 условия сходимости (Элемент на диагонали по модулю больше суммы оставшихся элементов по модулю)

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

{

double sum = 0;

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

{

if (j != i) sum += fabs(massA[i][j]);

}

if (fabs(massA[i][i]) < fabs(sum))

{

cout << "Условие 2 не выполняется. Вам необходимо вручную привести систему к нужному виду.\n Вы хотите увидеть ответ с приведением к единичной матрице? (y - да, любая клавиша - нет)\n";

cin >> key;

// Исправление под нужное решение

if (key == "y") {

canon(massA, massB, n);

cout << "Приведём систему к виду единичной матрицы для ответа.\n\n";

cout << "Исправленная матрица A\n";

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

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

cout << massA[i][j] << " ";

}

cout << endl;

}

cout << "Исправленная матрица B\n";

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

cout << massB[i] << "\n";

}

cout << "\nРешение вашей системы уравнений имеет вид:\n";

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

cout << "x[" << i + 1 << "]=" << massB[i] << endl;

}

check = 0;

}

else {

check = 0;

}

break;

}

}

// Вычисление методом Зейделя

if (check != 0) {

// Итерация 0

cout << "Итерация " << count << endl;

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

xf[i] = 0;

cout << "x[" << i << "]=" << xf[i] << endl;

}

count = 1;

// Итерация 1 и далее

do {

cout << "Итерация " << count << endl;

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

double sum1 = 0;

double sum2 = 0;

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

sum1 += massA[i][j] \* x[j];

}

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

sum2 += massA[i][j] \* xf[j];

}

x[i] = (massB[i] - sum1 - sum2) / massA[i][i];

cout << "x[" << i << "]=" << round(x[i] \* 10000) / 10000 << endl;

}

delta = fabs(x[0] - xf[0]);

// Переписывание новой точности

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

if (fabs(x[i] - xf[i]) > delta) {

delta = fabs(x[i] - xf[i]);

}

}

// Обновление значений переменных для следующей итерации

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

xf[i] = x[i];

}

count++;

} while (delta > eps);

// Вывод результата

cout << "Достигнута точность:" << delta << "\n";

cout << "Количество итераций:" << count << "\n";

cout << "Результат:\n";

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

{

cout << "x[" << i << "] = " << round(x[i] \* 10000) / 10000 << "\n";

}

// Освобождение памяти

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

{

delete[] massA[i];

}

delete[] massA;

delete[] massB;

delete[] x;

delete[] xf;

};

return 0;

}

};

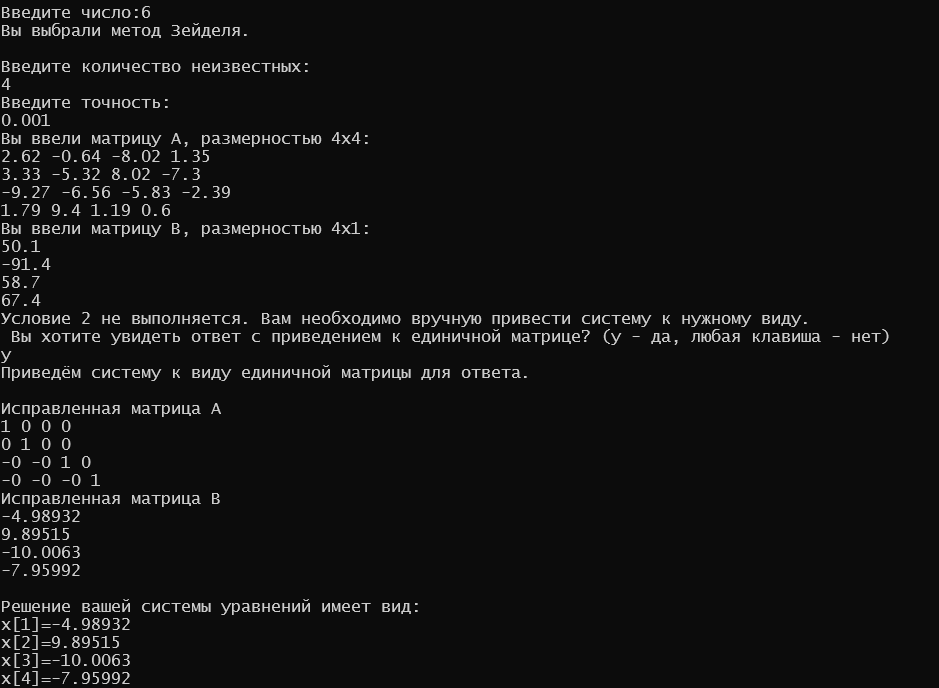


Рисунок 28. -Решение системы уравнений методом Зейделя в C++

# 8. Реализация на Java

***8.1 Меню***

Главный класс для пользователя, в котором выбирается нужный метод решения системы линейных алгебраических уравнений.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Menu {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** choice;

**int** exit = 0;

**do** {

System.***out***.println("Программа решает СЛАУ. Выберите метод, которым хотите решить (введите номер метода):");

System.***out***.println("1. Метод сопряжённых градиентов.");

System.***out***.println("2. Метод ортогонализации.");

System.***out***.println("3. Метод отражений.");

System.***out***.println("4. Метод квадратных корней.");

System.***out***.println("5. Метод простой итерации.");

System.***out***.println("6. Метод Зейделя.");

System.***out***.println("7. Завершение работы.\n");

System.***out***.println("Введите число: ");

choice = scanner.nextInt();

**while**(choice !=1 && choice !=2 && choice !=3 && choice !=4 && choice !=5 && choice !=6 && choice !=7) {

System.***out***.println("Введите число от 1 до 7!");

**while** (**true**) {

**if** (scanner.hasNextInt()) {

choice = scanner.nextInt();

**break**;

} **else** {

System.***out***.println("Введите число!");

scanner.nextLine();

}

}

}

**switch** (choice) {

**case** 1:

System.***out***.println("Вы выбрали метод сопряжёных градиентов.");

System.***out***.println("Дробные значения вводить в формате n,n \n");

Gradient.*main*(args);

**break**;

**case** 2:

System.***out***.println("Вы выбрали метод ортогонализации.");

System.***out***.println("Дробные значения вводить в формате n,n \n");

Orthogonalization.*main*(args);

**break**;

**case** 3:

System.***out***.println("Вы выбрали метод отражений.");

System.***out***.println("Комплексные значения вводить в формате (n,n) \n");

Reflections.*main*(args);

**break**;

**case** 4:

System.***out***.println("Вы выбрали метод квадратных корней.\n");

System.***out***.println("Комплексные значения вводить в формате (n,n) \n");

SQRT.*main*(args);

**break**;

**case** 5:

System.***out***.println("Вы выбрали метод простой итерации.\n");

System.***out***.println("Дробные значения вводить в формате n,n \n");

EasyIter.*main*(args);

**break**;

**case** 6:

System.***out***.println("Вы выбрали метод Зейделя.\n");

System.***out***.println("Дробные значения вводить в формате n,n \n");

Zeidel.*main*(args);

**break**;

**case** 7:

System.***out***.println("Завершение работы.\n");

exit = 1;

**break**;

} } **while** (exit == 0);

// scanner.close();

}

}

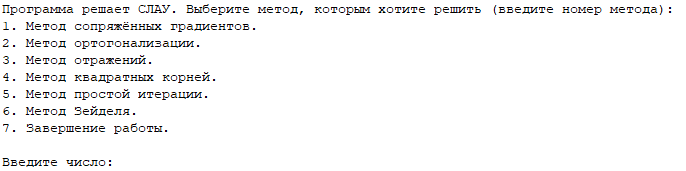


Рисунок 29.-Приветствие и выбор действия в программе в Java

***8.2 Класс комплексных чисел***

Класс с реализацией действий над комплексными числами.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Complex {

**private** **double** real;

**private** **double** imaginary;

//Конструктор комплексного числа

**public** Complex(**double** real, **double** imaginary) {

**this**.real = real;

**this**.imaginary = imaginary;

}

//Возвращение истинной части

**public** **double** getReal() {

**return** real;

}

//Возвращение мнимой части

**public** **double** getImaginary() {

**return** imaginary;

}

//Вставка истинной части

**public** **void** setReal(**double** real) {

**this**.real = real;

}

//Вставка мнимой части

**public** **void** setImaginary(**double** imaginary) {

**this**.imaginary = imaginary;

}

//Сложение

**public** Complex plus(Complex other) {

**double** newReal = **this**.real + other.real;

**double** newImaginary = **this**.imaginary + other.imaginary;

**return** **new** Complex(newReal, newImaginary);

}

//Вычитание

**public** Complex minus(Complex other) {

**double** newReal = **this**.real - other.real;

**double** newImaginary = **this**.imaginary - other.imaginary;

**return** **new** Complex(newReal, newImaginary);

}

//Умножение

**public** Complex multiply(Complex other) {

**double** newReal = **this**.real \* other.real - **this**.imaginary \* other.imaginary;

**double** newImaginary = **this**.real \* other.imaginary + **this**.imaginary \* other.real;

**return** **new** Complex(newReal, newImaginary);

}

//Деление

**public** Complex divide(Complex other) {

**double** denominator = other.real \* other.real + other.imaginary \* other.imaginary;

**double** newReal = (**this**.real \* other.real + **this**.imaginary \* other.imaginary) / denominator;

**double** newImaginary = (**this**.imaginary \* other.real - **this**.real \* other.imaginary) / denominator;

**return** **new** Complex(newReal, newImaginary);

}

//Квадратный корень

**public** Complex sqrt() {

**double** magnitude = Math.*sqrt*(real \* real + imaginary \* imaginary);

**double** angle = Math.*atan2*(imaginary, real);

**double** sqrtMagnitude = Math.*sqrt*(magnitude);

**double** sqrtAngle = angle / 2.0;

**double** realPart = sqrtMagnitude \* Math.*cos*(sqrtAngle);

**double** imaginaryPart = sqrtMagnitude \* Math.*sin*(sqrtAngle);

**return** **new** Complex(realPart, imaginaryPart);

}

//Функция аргумента

**public** **double** arg() {

**return** Math.*atan2*(imaginary, real);

}

//Модуль числа

**public** **double** abs() {

**return** Math.*sqrt*(real \* real + imaginary \* imaginary);

}

//Умножение

**public** Complex multiply(**double** num) {

**double** realPart = real \* num;

**double** imaginaryPart = imaginary \* num;

**return** **new** Complex(realPart, imaginaryPart);

}

// Возведение в степень

**public** Complex pow(**int** exponent) {

**double** denominator = real \* real + imaginary \* imaginary;

**double** realPart = real / denominator;

**double** imaginaryPart = -imaginary / denominator;

**return** **new** Complex(realPart, imaginaryPart);

}

//Чтение введённого комплексного числа

**public** **static** Complex read(Scanner scanner) {

**double** real = scanner.nextDouble();

**double** imaginary = scanner.nextDouble();

**return** **new** Complex(real, imaginary);

}

@Override

**public** String toString() { //Приведение в строку

**if** (real != 0 && imaginary != 0) {

**if** (imaginary > 0) {

**return** real + " + " + imaginary + "i";

} **else** {

**return** real + " - " + (-imaginary) + "i";

}

} **else** **if** (real != 0) {

**return** Double.*toString*(real);

} **else** **if** (imaginary != 0) {

**return** imaginary + "i";

} **else** {

**return** "0";

}

}

}

***8.3 Метод сопряжённых градиентов***

Класс с реализацией решения системы линейных алгебраических уравнений методом сопряжённых градиентов.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Gradient {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** n = 3;

// Ввод с клавиатуры

/\*while (!scanner.hasNextInt()) {

System.out.println("re-enter");

scanner.next();

}

n = scanner.nextInt();

\*/

// Объявление используемых переменных

**double** al, bt, rr, Arr, rDelta, ArDelta, Adeltadelta, e;

**double** E = 0.0001;

**double**[] r = **new** **double**[n];

**double**[] delta = **new** **double**[n];

**double**[] x = **new** **double**[n];

**double**[] Ax = **new** **double**[n];

**double**[] Ar = **new** **double**[n];

**double**[] Adelta = **new** **double**[n];

**double**[][] massA = **new** **double**[n][n];

**double**[] massB = **new** **double**[n];

// Вариант 7

massA[0][0] = 6;

massA[0][1] = 1;

massA[0][2] = 3;

massA[1][0] = 1;

massA[1][1] = 7;

massA[1][2] = 2;

massA[2][0] = 3;

massA[2][1] = 2;

massA[2][2] = 8;

massB[0] = -17;

massB[1] = -21;

massB[2] = -31;

/\*

System.out.println("Введите матрицу А");

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

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

System.out.print("A[" + i + "][" + j + "] = ");

while (!scanner.hasNextDouble()) {

System.out.println("Повторите ввод");

scanner.next();

}

A[i][j] = scanner.nextDouble();

}

System.out.println();

}

System.out.println("Введите матрицу В:");

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

System.out.print("B[" + i + "] = ");

while (!scanner.hasNextDouble()) {

System.out.println("Повторите ввод");

scanner.next();

}

B[i] = scanner.nextDouble();

System.out.println();

}

\*/

// первая итерация

System.***out***.println("Шаг 1");

x[0] = 1;

**for** (**int** i = 1; i < n; i++) {

x[i] = 0;

}

*matMultVector*(massA, x, n, Ax);

*vecMinVector*(Ax, massB, n, r);

rr = *vectorMulti*(r, r, n);

System.***out***.println("(r0,r0)=" + rr);

*matMultVector*(massA, r, n, Ar);

Arr = *vectorMulti*(Ar, r, n);

System.***out***.println("(Ar0,r0)=" + Arr);

bt = -(rr / Arr);

System.***out***.println("beta0=" + bt);

*vecMultNum*(r, bt, n, delta);

*vecPlusVector*(x, delta, n, x);

System.***out***.println("Результат итерации");

**for** (**int** i = 0; i < 3; i++) {

System.***out***.println("x1[" + (i + 1) + "]=" + x[i]);

} // Конец первой итерации

**int** count = 2;

// Цикл следующих итераций

**do** {

System.***out***.println();

System.***out***.println("Шаг " + count);

*matMultVector*(massA, x, n, Ax);

*vecMinVector*(Ax, massB, n, r);

*matMultVector*(massA, r, n, Ar);

*matMultVector*(massA, delta, n, Adelta);

rr = *vectorMulti*(r, r, n);

System.***out***.println("(r" + (count - 1) + ",r" + (count - 1) + ")=" + rr);

rDelta = *vectorMulti*(r, delta, n);

System.***out***.println("(r" + (count - 1) + ",delta" + (count - 1) + ")=" + rDelta);

ArDelta = *vectorMulti*(Ar, delta, n);

System.***out***.println("(r" + (count - 1) + ",Ardelta" + (count - 1) + ")=" + ArDelta);

Arr = *vectorMulti*(Ar, r, n);

System.***out***.println("(Ar" + (count - 1) + ",r" + (count - 1) + ")=" + Arr);

Adeltadelta = *vectorMulti*(Adelta, delta, n);

System.***out***.println("(Adelta" + (count - 1) + ",delta" + (count - 1) + ")=" + Adeltadelta);

al = (((rr \* ArDelta) - (rDelta \* Arr)) / ((Adeltadelta \* Arr) - (ArDelta \* ArDelta)));

System.***out***.println("alfa" + (count - 1) + "=" + ArDelta);

bt = (((rDelta \* ArDelta) - (rr \* Adeltadelta)) / ((Adeltadelta \* Arr) - (ArDelta \* ArDelta)));

System.***out***.println("beta" + (count - 1) + "=" + ArDelta);

*vecMultNum*(delta, al, n, delta);

*vecMultNum*(r, bt, n, r);

*vecPlusVector*(delta, r, n, delta);

*vecPlusVector*(x, delta, n,x);

System.***out***.println("Результат итерации");

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

System.***out***.println("x" + count + "[" + (i + 1) + "]=" + x[i]);

}

count++;

} **while** (Math.*abs*(rr) >= E);

System.***out***.println();

System.***out***.println("Результат решения:");

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

System.***out***.println("x[" + (i + 1) + "]=" + x[i]);

}

}

// Скалярное произведение

**public** **static** **double** vectorMulti(**double**[] vector1, **double**[] vector2, **int** n) {

**double** result = 0;

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

result += vector1[i] \* vector2[i];

}

**return** result;

}

// Произведение матрицы на вектор

**public** **static** **void** matMultVector(**double**[][] matrix, **double**[] vector, **int** n, **double**[] result) {

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

result[i] = 0;

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

result[i] += matrix[i][j] \* vector[j];

}

}

}

// Векторное произведение

**public** **static** **void** vecMultVector(**double**[] vector1, **double**[] vector2, **int** n, **double**[] result) {

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

result[i] = vector1[i] \* vector2[i];

}

}

//Вычитание матриц

**public** **static** **void** vecMinVector(**double**[] vector1, **double**[] vector2, **int** n, **double**[] result) {

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

result[i] = vector1[i] - vector2[i];

}

}

// Вектор умножить на число

**public** **static** **void** vecMultNum(**double**[] vector, **double** num, **int** n, **double**[] result) {

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

result[i] = vector[i] \* num;

}

}

// Сложение векторов

**public** **static** **void** vecPlusVector(**double**[] vector1, **double**[] vector2, **int** n, **double**[] result) {

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

result[i] = vector1[i] + vector2[i];

}

}

}

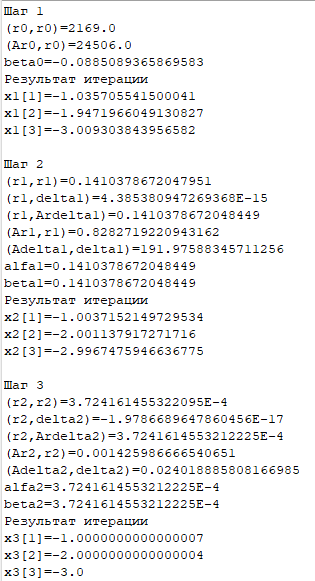


Рисунок 30.-Решение системы уравнений методом сопряжённых градиентов в Java

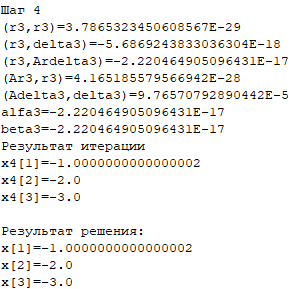


Рисунок 31.-Решение системы уравнений методом сопряжённых градиентов в Java

***8.4 Метод квадратных корней***

Класс с реализацией решения системы линейных алгебраических уравнений методом квадратных корней.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** SQRT {

//Ввод комплексных чисел

**public** **static** Complex read(Scanner scanner) {

**double** real = scanner.nextDouble();

**double** imaginary = scanner.nextDouble();

**return** **new** Complex(real, imaginary);

}

//Произведение матриц

**public** **static** **void** matMultMat(Complex[][] matrix1, **int** n, Complex[][] matrix2, Complex[][] result) {

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

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

result[i][j] = **new** Complex(0.0, 0.0);

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

result[i][j] = result[i][j].plus(matrix1[i][k].multiply(matrix2[k][j]));

}

}

}

}

//Произведение матрицы на вектор

**public** **static** **void** matMultVec(Complex[][] matrix, **int** n, Complex[] vector, Complex[] result) {

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

result[i] = **new** Complex(0.0, 0.0);

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

result[i] = result[i].plus(matrix[i][j].multiply(vector[j]));

}

}

}

//Приведение к симметричной матрице

**public** **static** **void** symmetricMatrix(Complex[][] massA, **int** n, Complex[] massB) {

Complex[][] massATrans = **new** Complex[n][n];

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

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

massATrans[j][i] = massA[i][j];

}

}

Complex[][] massA2 = **new** Complex[n][n];

Complex[] massB2 = **new** Complex[n];

*matMultMat*(massATrans, n, massA, massA2);

*matMultVec*(massATrans, n, massB, massB2);

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

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

massA[i][j] = massA2[i][j];

}

}

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

massB[i] = massB2[i];

}

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

**for** (**int** t = 0; t < n; t++) {

System.***out***.println("A[" + k + "][" + t +"] = " + massA[k][t]);

}

}

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

System.***out***.println("B[" + k + "] = " + massB[k]);

}

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** n;

//n = scanner.nextInt();

n=3;

Complex u = **new** Complex(0,0);

Complex sumY = **new** Complex(0,0);

Complex[][] massA = **new** Complex[n][n];

/\*

//Ввод с клавиатуры

System.out.println("Введите матрицу A");

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

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

System.out.print("A[" + i + "][" + j + "]= ");

massA[i][j] = read(scanner);

}

System.out.println();

}

\*/

Complex[] massB = **new** Complex[n];

/\*

//Ввод с клавиатуры

System.out.println("Введите матрицу B");

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

System.out.print("B[" + i + "]= ");

massB[i] = read(scanner);

}

\*/

//Вариант 7

massA[0][0] = **new** Complex(1, 0);

massA[0][1] = **new** Complex(3, -4);

massA[0][2] = **new** Complex(11, 5);

massA[1][0] = **new** Complex(3, 4);

massA[1][1] = **new** Complex(-3, 0);

massA[1][2] = **new** Complex(7, 1);

massA[2][0] = **new** Complex(11, -5);

massA[2][1] = **new** Complex(7, -1);

massA[2][2] = **new** Complex(-4, 0);

massB[0] = **new** Complex(48, 31);

massB[1] = **new** Complex(28, 13);

massB[2] = **new** Complex(7, -31);

Complex[][] U = **new** Complex[n][n];

//Инициализация матрицы U

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

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

U[i][j] = **new** Complex(0.0 ,0.0);

}

}

//Приведение к симметричному виду

System.***out***.print("Симметричная матрица");

*symmetricMatrix*(massA, n, massB);

Complex[] y = **new** Complex[n];

Complex[] x = **new** Complex[n];

//Объявление вектора с y

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

y[i] = **new** Complex(0.0, 0.0);

}

//Вычисление Utr\*y=B

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

**for** (**int** j = 0; j < i + 1; j++) {

u = **new** Complex(0.0, 0.0);

//Диагональные элементы матрицы U

**if** (i == j) {

**for** (**int** k = 0; k < j; k++) {

u = u.plus(U[j][k].multiply(U[j][k])); // U[j][k]\*U[j][k];

}

U[i][i] = massA[i][i].minus(u).sqrt();

}

**else** {

//Остальные элементы строки матрицы U

**for** (**int** k = 0; k < j; k++) {

u = u.plus(U[i][k].multiply(U[j][k]));

}

U[i][j] = (massA[i][j].minus(u)).divide(U[j][j]);

}

}

sumY = **new** Complex(0.0 ,0.0);

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

sumY = sumY.plus(U[i][j].multiply(y[j]));

}

y[i] = (massB[i].minus(sumY)).divide(U[i][i]);

}

//Вывод матрицы U

System.***out***.print("Матрица U");

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

**for** (**int** j = 0; j < i + 1; j++) {

System.***out***.println("U[" + i + "][" + j + "]= " + U[i][j]);

}

}

System.***out***.println("Матрица y:");

//Вывод y

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

System.***out***.println("y[" + i + "] = " + y[i]);

}

// Подсчёт U\*x=y

**for** (**int** i = n - 1; i > -1; i--) {

Complex sum = **new** Complex(0.0 ,0.0);

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

sum = sum.plus(U[k][i].multiply(x[k]));

}

x[i] = (y[i].minus(sum)).divide(U[i][i]);

}

//Результат

System.***out***.println("Матрица х (результат):");

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

System.***out***.println("x[" + i + "] = " + x[i]);

}

}

}

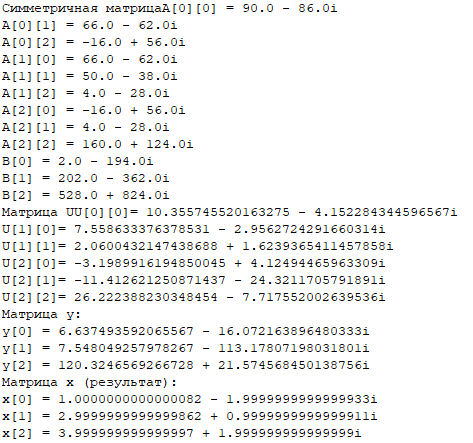


Рисунок 32.-Решение системы уравнений методом квадратных корней в Java

***8.5 Метод ортогонализации***

Класс с реализацией решения системы линейных алгебраических уравнений методом ортогонализации.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Orthogonalization {

//Векторное произведение

**public** **static** **double** vectorMult(**double**[] vector1, **double**[] vector2, **int** n) {

**double** result = 0;

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

result += vector1[i] \* vector2[i];

}

**return** result;

}

// Сумма векторов

**public** **static** **void** vectorSum(**double**[] vector1, **double**[] vector2, **int** n, **double**[] result) {

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

result[i] = vector1[i] + vector2[i];

}

}

// Получение вектора из матрицы

**public** **static** **void** rowМector(**double**[][] matrix, **int** n, **int** row, **double**[] vectorrow) {

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

vectorrow[i] = matrix[row][i];

}

}

// Вектор делить на число

**public** **static** **void** vectorDivNum(**double**[] vector, **int** n, **double** num, **double**[] result) {

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

result[i] = vector[i] / num;

}

}

// Вектор умножить на число

**public** **static** **void** vectorMultNum(**double**[] vector, **int** n, **double** num, **double**[] result) {

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

result[i] = vector[i] \* num;

}

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**double** vecMult;

**int** n, il;

il = 0;

//System.out.println("Введите размер матрицы");

//n = scanner.nextInt();

n=5;

**double**[][] u = **new** **double**[n][n];

**double**[][] b = **new** **double**[n][n];

**double**[] c = **new** **double**[n];

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

**double**[][] cb = **new** **double**[n][n];

**double**[] x = **new** **double**[n];

**double**[][] massA = **new** **double**[n][n];

/\* Ввод матрицы А с клавиатуры

System.out.println("Введите матрицу A (после ввода матрицы, введите строку, состоящую из n-1 нулей и 1 в конце строки)");

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

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

System.out.print("A[" + i + "][" + j + "] = ");

A[i][j] = scanner.nextDouble();

while (!scanner.hasNextDouble()) {

System.out.println("Повторите ввод");

scanner.next();

}

}

}

\*/

//Вариант 7

massA[0][0] = -5.709;

massA[0][1] = 4.35;

massA[0][2] = 8.639;

massA[0][3] = 3;

massA[0][4] = 84.1;

massA[1][0] = 1.650;

massA[1][1] = -5.350;

massA[1][2] = 6.3;

massA[1][3] = -9.049;

massA[1][4] = 58.1;

massA[2][0] = 7.250;

massA[2][1] = 8.200;

massA[2][2] = 5.459;

massA[2][3] = 3.670;

massA[2][4] = 42.3;

massA[3][0] = -0.869;

massA[3][1] = -7.189;

massA[3][2] = 1.950;

massA[3][3] = 4.759;

massA[3][4] = 0.5;

massA[4][0] = 0;

massA[4][1] = 0;

massA[4][2] = 0;

massA[4][3] = 0;

massA[4][4] = 1;

System.***out***.println("Ваша матрица имеет вид:");

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

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

System.***out***.print(massA[i][j] + " ");

}

System.***out***.println(" ");

}

// Шаг 1

*rowМector*(massA, n, 0, u[0]);

vecMult = *vectorMult*(u[0], u[0], n);

*vectorDivNum*(u[0], n, Math.*sqrt*(vecMult), b[0]);

// Шаг 2

**for** (**int** i = 1; i < n; i++) {

*rowМector*(massA, n, i, a[i]);

*rowМector*(massA, n, i, u[i]);

**for** (**int** l = 0; l < i; l++) {

c[l] = -*vectorMult*(a[i], b[l], n);

*vectorMultNum*(b[l], n, c[l], cb[l]);

**if** (l > 0) {

*vectorSum*(cb[l], cb[l - 1], n, cb[l]);

}

il = l;

}

*vectorSum*(cb[il], u[i], n, u[i]);

vecMult = *vectorMult*(u[i], u[i], n);

*vectorDivNum*(u[i], n, Math.*sqrt*(vecMult), b[i]);

}

*vectorDivNum*(b[n - 1], n, b[n - 1][n - 1], x);

System.***out***.println("Результат:");

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

System.***out***.println("x" + (i + 1) + " = " + x[i]);

}

}

}

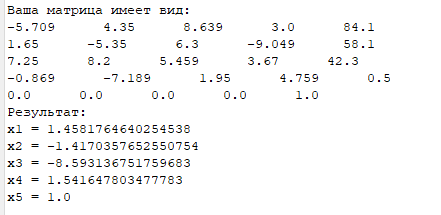


Рисунок 33.-Решение системы уравнений методом ортогонализации в Java

***8.6 Метод отражений***

Класс с реализацией решения системы линейных алгебраических уравнений методом отражений.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Reflections {

//Ввод комплексных чисел

**public** **static** Complex read(Scanner scanner) {

**double** real = scanner.nextDouble();

**double** imaginary = scanner.nextDouble();

**return** **new** Complex(real, imaginary);

}

//Матричное произведение

**public** **static** **void** matMultMat(Complex[][] matrix1, **int** n1, **int** m1, Complex[][] matrix2, **int** n2, **int** m2, Complex[][] result) {

**if** (m1 != n2) {

System.***out***.println("Размеры матриц не соответствуют условию умножения (количество столбцов первой не соответствуют количеству строк второй матрицы)");

**return**;

}

Complex[][] tempm = **new** Complex[n2][m2];

**for** (**int** i = 0; i < n2; i++) {

**for** (**int** j = 0; j < m2; j++) {

tempm[i][j] = matrix2[i][j];

}

}

**for** (**int** i = 0; i < n1; i++) {

**for** (**int** j = 0; j < m2; j++) {

Complex sum = **new** Complex(0.0, 0.0);

**for** (**int** k = 0; k < m1; k++) {

sum = sum.plus(matrix1[i][k].multiply(tempm[k][j]));

}

result[i][j] = sum;

}

}

}

//Скалярное произведение векторов

**public** **static** Complex sqalarVec(Complex[] vector1, Complex[] vector2, **int** n) {

Complex result = **new** Complex(0.0, 0.0);

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

result = result.plus(vector1[i].multiply(vector2[i]));

}

**return** result;

}

//Векторное произведение векторов

**public** **static** **void** vecMultVec(Complex[] vector1, Complex[] vector2, **int** n, Complex[][] result) {

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

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

result[i][j] = vector1[i].multiply(vector2[j]);

}

}

}

//Матрица минус вектор

**public** **static** **void** matMinVec(Complex[][] matrix, Complex[] vector, **int** n, **int** m, Complex[][] result) {

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

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

result[i][j] = matrix[i][j].minus(vector[j]);

}

}

}

//Матрица минус матрица

**public** **static** **void** matMinMat(Complex[][] mat1, Complex[][] mat2, **int** n, Complex[][] result) {

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

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

result[i][j] = mat1[i][j].minus(mat2[i][j]);

}

}

}

//Вектор минус вектор

**public** **static** **void** vecMinVec(Complex[] vector1, Complex[] vector2, **int** n, Complex[] result) {

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

result[i] = vector1[i].minus(vector2[i]);

}

}

//Вектор плюс вектор

**public** **static** **void** vecPlusVec(Complex[] vector1, Complex[] vector2, **int** n, Complex[] result) {

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

result[i] = vector1[i].plus(vector2[i]);

}

}

//Вектор умноженный на число

**public** **static** **void** vecMultNum(Complex[] vector, **int** n, Complex num, Complex[] result) {

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

result[i] = vector[i].multiply(num);

}

}

//Матрица умноженная на число

**public** **static** **void** matMultNum(Complex[][] mat, **int** n, **int** num, Complex[][] result) {

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

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

result[i][j] = mat[i][j].multiply(num);

}

}

}

//Вырезка столбца матрицы в вектор

**public** **static** **void** colVec(Complex[][] mat, **int** n, **int** col, Complex[] result) {

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

result[i] = mat[i][col];

}

}

//Вырезка строки матрицы в вектор

**public** **static** **void** rowVec(Complex[][] mat, **int** m, **int** row, Complex[] result) {

**for** (**int** i = 0; i < m; i++) {

result[i] = mat[row][i];

}

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** n = 0, m = 0;

**boolean** validInput = **false**;

n = 3;

m = 4;

/\*

while (!validInput) {

try {

n = Integer.parseInt(scanner.next());

m = Integer.parseInt(scanner.next());

validInput = true;

} catch (NumberFormatException e) {

System.out.println("Повторите ввод ");

}

}

\*/

//System.out.println("Введите матрицу А:");

Complex[][] massA = **new** Complex[n][m];

/\*

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

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

System.out.print("A[" + i + "][" + j + "] = ");

A[i][j] = read(scanner);

}

System.out.println();

}

\*/

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

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

massA[i][j] = **new** Complex(1.0, 1.0);

}

//System.out.println();

}

//Вариант 7

massA[0][0] = **new** Complex(1, 2);

massA[0][1] = **new** Complex(4, -5);

massA[0][2] = **new** Complex(7, 4);

massA[0][3] = **new** Complex(40, 67);

massA[1][0] = **new** Complex(8, 1);

massA[1][1] = **new** Complex(2, -1);

massA[1][2] = **new** Complex(1, 1);

massA[1][3] = **new** Complex(15, 33);

massA[2][0] = **new** Complex(3, 1);

massA[2][1] = **new** Complex(1, 1);

massA[2][2] = **new** Complex(2, 3);

massA[2][3] = **new** Complex(8, -41);

System.***out***.println("Ваша матрица имеет вид:");

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

**for** (**int** t = 0; t < m; t++) {

System.***out***.print(massA[i][t] + " ");

}

System.***out***.println();

}

Complex[][] E = **new** Complex[n][m];

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

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

**if** (i == j) {

E[i][i] = **new** Complex(1.0, 0.0);

} **else** {

E[i][j] = **new** Complex(0.0, 0.0);

}

}

}

Complex[][] U = **new** Complex[n][m];

Complex[][] W = **new** Complex[n][m];

Complex[] x = **new** Complex[n];

Complex[] s = **new** Complex[n];

Complex[] w = **new** Complex[n];

Complex[] L = **new** Complex[n];

Complex[] r = **new** Complex[m];

**boolean** absal;

**for** (**int** i = 0; i < n - 1; i++) {

System.***out***.println();

System.***out***.println("Шаг " + (i+1));

absal = **false**;

*colVec*(massA, n, i, s);

**if** (i > 0) {

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

s[k] = **new** Complex(0.0, 0.0);

}

}

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

**if** (j == i) {

L[j] = **new** Complex(1.0, 0.0);

} **else** {

L[j] = **new** Complex(0.0, 0.0);

}

}

Complex al = *sqalarVec*(s, s, n);

Complex sqrtal = al.sqrt();

Complex sL = *sqalarVec*(s, L, n);

**if** (sL.arg() - 3.14159 < 0) {

absal = **true**;

}

**if** (absal) {

Complex one = **new** Complex(-1.0, 0.0);

sqrtal = sqrtal.multiply(one);

}

Complex ro = al.multiply(2.0).plus(sqrtal.multiply(2.0).multiply(sL)).sqrt();

ro = ro.pow(-1);

//System.out.println("al = " + al + ", ro = " + ro);

*vecMultNum*(s, n, ro, s);

*vecMultNum*(L, n, sqrtal, L);

*vecMultNum*(L, n, ro, L);

*vecPlusVec*(s, L, n, w);

System.***out***.println("Вектор w:");

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

System.***out***.println(w[k]);

}

*vecMultVec*(w, w, n, W);

*matMultNum*(W, n, 2, W);

*matMinMat*(E, W, n, U);

*matMultMat*(U, n, n, massA, n, m, massA);

System.***out***.println("Вектор U:");

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

**for** (**int** t = 0; t < n; t++) {

System.***out***.print(U[k][t] + " ");

}

System.***out***.println();

}

System.***out***.println("Вектор A:");

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

**for** (**int** t = 0; t < m; t++) {

System.***out***.print(massA[k][t] +" ");

}

System.***out***.println();

}

}

Complex sum = **new** Complex(0, 0);

x[n - 1] = massA[n - 1][m - 1].divide(massA[n - 1][m - 2]);

**for** (**int** i = n - 2; i >= 0; i--) {

*rowVec*(massA, m, i, r);

sum = **new** Complex(0, 0);

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

sum = sum.minus(x[j].multiply(r[j]));

}

x[i] = r[m - 1].plus(sum).divide(r[i]);

}

System.***out***.println();

System.***out***.println("Результат вычисления:");

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

System.***out***.println("x[" + (i + 1) + "]= " + x[i]);

}

}

}

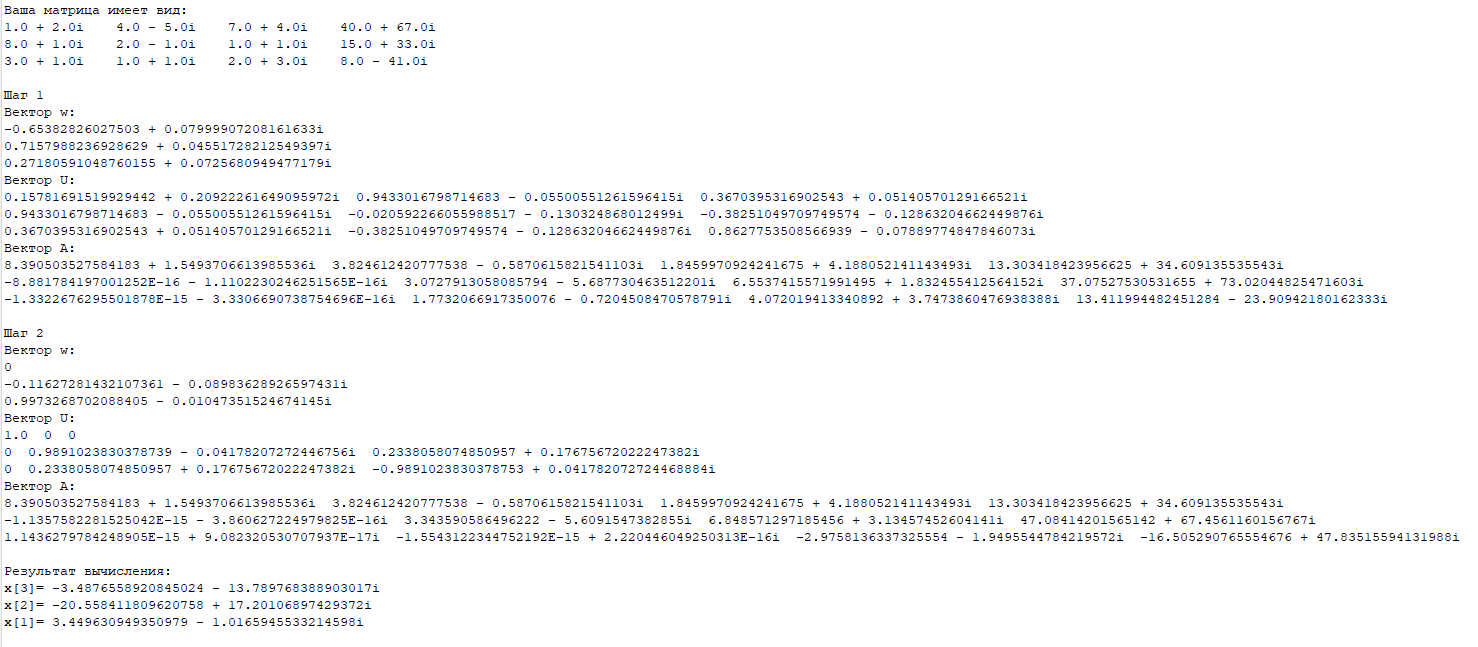


Рисунок 34.-Решение системы уравнений методом отражений в Java

***8.7 Метод простой итерации***

Класс с реализацией решения системы линейных алгебраических уравнений методом простой итерации.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** EasyIter {

// Подсчёт определителя матрицы А

**public** **static** **double** determinant(**double**[][] matrix, **int** size) {

**double** det = 0;

**double** sign = 1;

// Один элемент

**if** (size == 1) {

det = matrix[0][0];

}

// Матрица 2х2

**else** **if** (size == 2) {

det = (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]);

}

// Матрица (nхn)

**else** {

**for** (**int** i = 0; i < size; i++) {

// Разложение по строке(столбцу)

**double**[][] cofactor = **new** **double**[size - 1][size - 1];

**int** sub\_i = 0, sub\_j = 0;

**for** (**int** j = 1; j < size; j++) {

**for** (**int** k = 0; k < size; k++) {

**if** (k == i) {

**continue**;

}

cofactor[sub\_i][sub\_j] = matrix[j][k];

sub\_j++;

}

sub\_i++;

sub\_j = 0;

}

// Подсчёт значения определителя

det += sign \* matrix[0][i] \* *determinant*(cofactor, size - 1);

sign = -sign;

}

}

**return** det;

}

// Приведение к каноническому виду

**public** **static** **void** canon(**double**[][] massA, **double**[] massB, **int** n) {

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

**double** mult = massA[i][i];

// Делим строку на ведущий элемент, чтобы получить 1 на диагонали

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

massA[i][j] /= mult;

}

massB[i] /= mult;

// Обнуляем остальные элементы в столбце

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

**if** (k != i) {

**double** factor = massA[k][i];

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

massA[k][j] -= factor \* massA[i][j];

}

massB[k] -= factor \* massB[i];

}

}

}

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** n, check = 1;

String key;

**int** count = 0;

**double** delta, eps;

n=4;

/\*

System.out.println("Введите количество неизвестных:");

n = scanner.nextInt();

while (n <= 0) {

System.out.println("Введите положительное число:");

n = scanner.nextInt();

}

\*/

eps=0.001;

/\*

System.out.println("Введите точность:");

eps = scanner.nextDouble();

while (eps <= 0) {

System.out.println("Введите положительное число:");

eps = scanner.nextDouble();

}

\*/

//Выдeление динам. памяти для массива

**double**[][] massA = **new** **double**[n][n];

//Выдeление памяти для свободных членов

**double**[] massB = **new** **double**[n];

**double**[] x = **new** **double**[n];

**double**[] xf = **new** **double**[n];

//Вариант 10

massA[0][0] = -0.07;

massA[0][1] = 9.89;

massA[0][2] = -0.17;

massA[0][3] = -0.28;

massA[1][0] = 9.55;

massA[1][1] = -0.72;

massA[1][2] = -1.16;

massA[1][3] = 8.13;

massA[2][0] = -3.03;

massA[2][1] = -4.9;

massA[2][2] = 2.08;

massA[2][3] = 7.19;

massA[3][0] = -0.72;

massA[3][1] = -3.53;

massA[3][2] = 5.75;

massA[3][3] = -7.77;

massB[0] = 0.1;

massB[1] = -0.3;

massB[2] = 99.8;

massB[3] = -0.5;

// Проверка 1 условия сходимости (Определитель матрицы А не равен нулю)

**double** det = *determinant*(massA, n);

**if** (det == 0) {

System.***out***.println("Условие 1 не выполняется. Определитель матрицы А=0.");

System.***out***.println("Решение невозможно!");

**return**;

}

// Проверка 2 условия сходимости (Элемент на диагонали по модулю больше суммы оставшихся элементов по модулю)

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

**double** sum = 0;

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

**if** (j != i) sum += Math.*abs*(massA[i][j]);

}

**if** (Math.*abs*(massA[i][i]) < Math.*abs*(sum)) {

System.***out***.println("Условие 2 не выполняется. Вам необходимо вручную привести систему к нужному виду.");

System.***out***.println("Вы хотите увидеть ответ с приведением к единичной матрице? (y - да, любая клавиша - нет)");

key = scanner.next();

// Исправление под нужное решение

**if** (key.equals("y")) {

*canon*(massA, massB, n);

System.***out***.println("Приведём систему к виду единичной матрицы для ответа.\n");

System.***out***.println("Исправленная матрица A");

**for** (**int** h = 0; h < n; h++) {

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

System.***out***.print(massA[h][j] + " ");

}

System.***out***.println();

}

System.***out***.println("Исправленная матрица B");

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

System.***out***.println(massB[k]);

}

System.***out***.println("\nРешение вашей системы уравнений имеет вид:");

**for** (**int** l = 0; l < n; l++) {

System.***out***.println("x[" + (l + 1) + "]=" + massB[l]);

}

check = 0;

} **else** {

check = 0;

}

**break**;

}

}

**if** (check != 0) {

// Итераций 0

System.***out***.println("Итерация " + count);

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

xf[i] = 0;

System.***out***.println("x[" + i + "]=" + xf[i]);

}

System.***out***.println("Итерация " + count);

// Итерация 1

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

xf[i] = massB[i] / massA[i][i];

System.***out***.println("x[" + i + "]=" + Math.*round*(xf[i] \* 10000) / 10000);

}

count = 2;

// Подсчёт остальных итераций до достижения нужной точности

eps = 0.001;

delta =0;

**do** {

count++;

System.***out***.println("Итерация " + count);

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

**double** sum = 0;

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

// Подсчёт суммы элементов, которые не на диагонали

**if** (j != i) sum += massA[i][j] \* xf[j];

}

x[i] = (massB[i] - sum) / massA[i][i];

System.***out***.println("x[" + i + "]=" + Math.*round*(x[i] \* 10000) / 10000);

delta = Math.*abs*(x[0] - xf[0]);

// Переписывание новой точности

**for** (**int** j = 1; j < n; j++) {

**if** (Math.*abs*(x[j] - xf[j]) > delta)

delta = Math.*abs*(x[j] - xf[j]);

}

xf[i] = x[i];

}

} **while** (delta > eps);

// Вывод результата

System.***out***.println("Достигнута точность:" + delta);

System.***out***.println("Количество итераций:" + count);

System.***out***.println("Результат:");

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

System.***out***.println("x[" + i + "] = " + Math.*round*(x[i] \* 10000) / 10000);

}

// Освобождение памяти

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

massA[i] = **null**;

}

massA = **null**;

massB = **null**;

x = **null**;

xf = **null**;

}

}

}

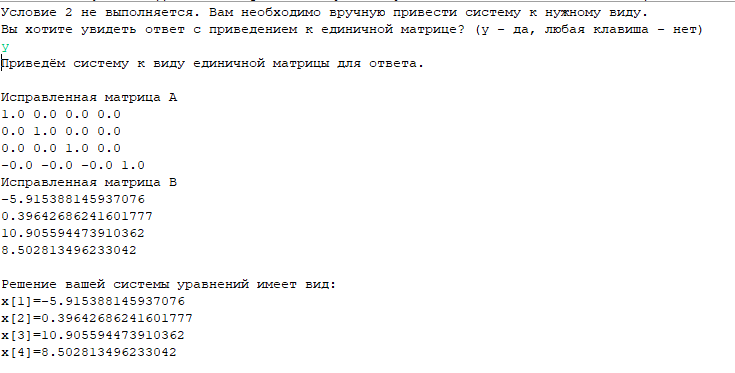


Рисунок 35.-Решение системы уравнений методом простых итераций в Java

***8.8 Метод Зейделя***

Класс с реализацией решения системы линейных алгебраических уравнений методом Зейделя.

Для удобства программа решает заданную систему линейных уравнений, но также возможен ввод значений с клавиатуры.

**package** Lab2;

**import** java.util.Scanner;

**public** **class** Zeidel {

// Подсчёт определителя матрицы А

**public** **static** **double** determinant(**double**[][] matrix, **int** size) {

**double** det = 0;

**double** sign = 1;

// Размер матрицы 1х1

**if** (size == 1) {

det = matrix[0][0];

}

// Размер матрицы 2х2

**else** **if** (size == 2) {

det = (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]);

}

// Размер n\*n

**else** {

**for** (**int** i = 0; i < size; i++) {

// Разложение по строке(столбцу)

**double**[][] cofactor = **new** **double**[size - 1][size - 1];

**int** sub\_i = 0, sub\_j = 0;

**for** (**int** j = 1; j < size; j++) {

**for** (**int** k = 0; k < size; k++) {

**if** (k == i) {

**continue**;

}

cofactor[sub\_i][sub\_j] = matrix[j][k];

sub\_j++;

}

sub\_i++;

sub\_j = 0;

}

// Подсчёт значения определителя

det += sign \* matrix[0][i] \* *determinant*(cofactor, size - 1);

sign = -sign;

}

}

**return** det;

}

// Приведение к каноническому виду

**public** **static** **void** canon(**double**[][] massA, **double**[] massB, **int** n) {

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

**double** mult = massA[i][i];

// Делим строку на ведущий элемент, чтобы получить 1 на диагонали

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

massA[i][j] /= mult;

}

massB[i] /= mult;

// Обнуляем остальные элементы в столбце

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

**if** (k != i) {

**double** factor = massA[k][i];

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

massA[k][j] -= factor \* massA[i][j];

}

massB[k] -= factor \* massB[i];

}

}

}

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

**int** n, check = 1;

String key;

**int** count = 0;

**double** delta, eps;

n=4;

/\*

System.out.println("Введите количество неизвестных:");

n = scanner.nextInt();

while (n <= 0) {

System.out.println("Введите положительное число:");

n = scanner.nextInt();

}

\*/

eps=0.001;

/\*

System.out.println("Введите точность:");

eps = scanner.nextDouble();

while (eps <= 0) {

System.out.println("Введите положительное число:");

eps = scanner.nextDouble();

}

\*/

//Выдeление динам. памяти для массива

**double**[][] massA = **new** **double**[n][n];

//Выдeление памяти для свободных членов

**double**[] massB = **new** **double**[n];

**double**[] x = **new** **double**[n];

**double**[] xf = **new** **double**[n];

//Вариант 10

massA[0][0] = 2.62;

massA[0][1] = -0.64;

massA[0][2] = -8.02;

massA[0][3] = 1.35;

massA[1][0] = 3.33;

massA[1][1] = -5.32;

massA[1][2] = 8.02; //

massA[1][3] = -7.3;

massA[2][0] = -9.27;

massA[2][1] = -6.56;

massA[2][2] = -5.83;

massA[2][3] = -2.39;

massA[3][0] = 1.79;

massA[3][1] = 9.4;

massA[3][2] = 1.19;

massA[3][3] = 0.6;

massB[0] = 50.1;

massB[1] = -91.4;

massB[2] = 58.7;

massB[3] = 67.4;

// Вывод матрицы A

System.***out***.println("Вы ввели матрицу A, размерностью " + n + "х" + n + ":");

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

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

System.***out***.print(massA[i][j] + " ");

}

System.***out***.println();

}

// Вывод матрицы B

System.***out***.println("Вы ввели матрицу B, размерностью " + n + "х1:");

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

System.***out***.println(massB[i]);

}

// Проверка 1 условия сходимости (Определитель матрицы А не равен нулю)

**double** det = *determinant*(massA, n);

**if** (det == 0) {

System.***out***.println("Условие 1 не выполняется. Определитель матрицы А=0.");

System.***out***.println("Решение невозможно!");

**return**;

}

// Проверка 2 условия сходимости (Элемент на диагонали по модулю больше суммы оставшихся элементов по модулю)

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

**double** sum = 0;

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

**if** (j != i) sum += Math.*abs*(massA[i][j]);

}

**if** (Math.*abs*(massA[i][i]) < Math.*abs*(sum)) {

System.***out***.println("Условие 2 не выполняется. Вам необходимо вручную привести систему к нужному виду.");

System.***out***.println("Вы хотите увидеть ответ с приведением к единичной матрице? (y - да, любая клавиша - нет)");

key = scanner.next();

// Исправление под нужное решение

**if** (key.equals("y")) {

*canon*(massA, massB, n);

System.***out***.println("Приведём систему к виду единичной матрицы для ответа.\n");

System.***out***.println("Исправленная матрица A");

**for** (**int** il = 0; il < n; il++) {

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

System.***out***.print(massA[il][j] + " ");

}

System.***out***.println();

}

System.***out***.println("Исправленная матрица B");

**for** (**int** ik = 0; ik < n; ik++) {

System.***out***.println(massB[ik]);

}

System.***out***.println("\nРешение вашей системы уравнений имеет вид:");

**for** (**int** in = 0; in < n; in++) {

System.***out***.println("x[" + (in + 1) + "]=" + massB[in]);

}

check = 0;

} **else** {

check = 0;

}

**break**;

}

}

// Вычисление методом Зейделя

**if** (check != 0) {

// Итерация 0

System.***out***.println("Итерация " + count);

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

xf[i] = 0;

System.***out***.println("x[" + i + "]=" + xf[i]);

}

count = 1;

// Итерация 1 и далее

**do** {

System.***out***.println("Итерация " + count);

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

**double** sum1 = 0;

**double** sum2 = 0;

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

sum1 += massA[i][j] \* x[j];

}

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

sum2 += massA[i][j] \* xf[j];

}

x[i] = (massB[i] - sum1 - sum2) / massA[i][i];

System.***out***.println("x[" + i + "]=" + Math.*round*(x[i] \* 10000) / 10000);

}

delta = Math.*abs*(x[0] - xf[0]);

// Переписывание новой точности

**for** (**int** i = 1; i < n; i++) {

**if** (Math.*abs*(x[i] - xf[i]) > delta) {

delta = Math.*abs*(x[i] - xf[i]);

}

}

// Обновление значений переменных для следующей итерации

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

xf[i] = x[i];

}

count++;

} **while** (delta > eps);

// Вывод результата

System.***out***.println("Достигнута точность:" + delta);

System.***out***.println("Количество итераций:" + count);

System.***out***.println("Результат:");

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

System.***out***.println("x[" + i + "] = " + Math.*round*(x[i] \* 10000) / 10000);

}

// Освобождение памяти

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

massA[i] = **null**;

}

massA = **null**;

massB = **null**;

x = **null**;

xf = **null**;

}

}

}

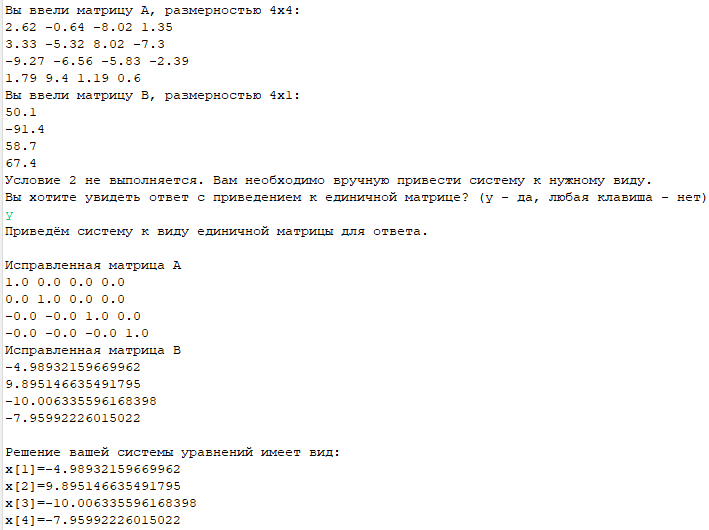


Рисунок 36.-Решение системы уравнений методом Зейделя в Java

# 9. Результат и анализ реализаций

Таблица 2

Таблица результатов решения системы линейных алгебраических уравнений

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Метод | Отражений | Квадратных корней | Сопряженных градиентов | Ортогонализации | Простой итерации | Зейделя |
| Ручной счёт +  excel |  |  |  |  |  |  |
| Mathcad |  |  |  |  |  |  |
| C++ |  |  |  |  |  |  |
| Java |  |  |  |  |  |  |

Сравнив результаты, можно увидеть, что с помощью использованных инструментов можно достичь необходимой точности, в данном случае она равна 0,000001 и получить почти одинаковые ответы.

# 10. Вывод

По итогам лабораторной работы можно выделить следующие характеристики методов решения систем линейных алгебраических уравнений:

1. Метод простых итераций:
   1. Этот метод основан на последовательном уточнении приближенного решения систем линейных алгебраических уравнений.
   2. Сходимость метода зависит от ненулевого определителя матрицы коэффициентов при х, а также диагональных элементов матрицы коэффициентов при х.
   3. Метод простых итераций может быть медленным для систем с большим числом уравнений или плохо обусловленных матриц.
2. Метод Зейделя:
   1. Этот метод является модификацией метода простых итераций, где итерации выполняются по очереди для каждого уравнения системы.
   2. Метод Зейделя может обеспечить более быструю сходимость по сравнению с методом простых итераций.
   3. Для некоторых матриц метод Зейделя может не сойтись или сойтись медленно.
3. Метод ортогонализации:
   1. Метод может быть применен к системам симметричных матриц.
   2. Метод ортогонализации может быть эффективным для больших разреженных систем, но требует большого объема вычислений для плотных матриц.
4. Метод отражений:
   1. Метод эффективен для решения систем симметричных матриц и матриц с малым числом ненулевых элементов.
   2. Метод отражений требует большего количества операций, чем некоторые другие методы, но может быть численно стабильным.
5. Метод сопряженных градиентов:
   1. Этот метод применяется для решения систем линейных алгебраических уравнений с симметричными положительно определенными матрицами.
   2. Метод сопряженных градиентов обладает хорошей сходимостью для матриц с небольшим числом обусловленности и может быть эффективным для больших систем.
6. Метод квадратных корней:
   1. Метод применим к системам симметричных положительно определенных матриц.
   2. Метод квадратных корней может быть эффективным для больших систем симметричных матриц.

Разные методы могут быть эффективными в разных сценариях, в зависимости от структуры и свойств матрицы, размера системы, требуемой точности решения и доступных вычислительных ресурсов. Некоторые методы могут обладать лучшей сходимостью для определенных типов матриц, в то время как другие методы могут быть более универсальными, но менее эффективными.

# 10. Использованная литература

**1.** **Численное решение задач экономики с использованием EXCEL, C++ и MATLAB [Электронные текстовые данные] : Учеб.пособие / Л.Ю. Катаева [и др.]; НГТУ им.Р.Е.Алексеева. - Н.Новгород : [Изд-во НГТУ], 2020. - 230 с. : ил. - Прил.:c.188-230. - Библиогр.:с.187.:**

[**https://fdp.nntu.ru/books/Chisl\_reshenie\_zadach\_economiki/Chisl\_reshenie\_zadach\_economiki/assets/basic-html/index.html#189в**](https://fdp.nntu.ru/books/Chisl_reshenie_zadach_economiki/Chisl_reshenie_zadach_economiki/assets/basic-html/index.html#189%D0%B2)**.**

**2. Численные методы : Курс лекций / В.А. Срочко. - СПб.; М.; Краснодар : Лань, 2010. - 202 с. - (Учебники для вузов. Специальная литература). - Библиогр.:с.200. - ISBN 978-5-8114-1014-9 : 180-00.**

[**https://studfile.net/preview/5793014/page:4/**](https://studfile.net/preview/5793014/page:4/)

**3. Численные методы линейной алгебры : Учеб.пособие / Г.С. Шевцов, О.Г. Крюкова, Б.И. Мызникова. - 2-е изд.,испр.и доп. - СПб.; М.; Краснодар : Лань, 2011. - 495 с. - (Учебники для вузов. Специальная литература). - Предм.указ.:с.491-495. - Библиогр.:с.489-490. - ISBN 978-5-8114-1246-4 : 465-00.:**

[**https://dpm.pstu.ru/images/R/Z/shevcov\_lineynaya\_algebra.pdf**](https://dpm.pstu.ru/images/R/Z/shevcov_lineynaya_algebra.pdf)