МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

Курсовой проект

по дисциплине «Вычислительная математика»

на тему «Решение систем линейных алгебраических уравнений методами с неполной релаксации»

Направление подготовки
02.03.01 «Математика и компьютерные науки»

Выполнила студентка гр.
Б9122-02.03.01сцт

_____ Винницкая Дина Сергеевна
(*подпись*) *(Ф.И.О.)*

Проверил доцент, к.ф.-м.н.

Колобов А.Г._____
(подпись)

«_____»_____2024г.

г. Владивосток
2024

Оглавление

1. **Введение**

Решение систем линейных алгебраических уравнений (СЛАУ) является одной из ключевых задач вычислительной математики и находит широкое применение в самых разных областях: от физики и инженерии до экономики и информатики. Современные методы решения СЛАУ варьируются от прямых методов, таких как метод Гаусса, до итерационных подходов, включающих метод неполной релаксации. Последний представляет собой эффективный инструмент для обработки крупных и разреженных систем, что делает его особенно актуальным в современных вычислительных задачах.

Цель работы – изучить теоретические аспекты метода неполной релаксации, его преимуществ и ограничений, а также проведение сравнительного анализа с другими методами решения СЛАУ. Дополнительно планируется проверить метод на устойчивость, точность и вычислительную сложность.

Задачи исследования:

- Изучить теоретические основы метода неполной релаксации для решения СЛАУ.
- Разработать и реализовать алгоритм решения СЛАУ методом неполной релаксации.
- Провести тестирование алгоритма на различных типах матриц и оценить его точность и устойчивость.
- Сравнить метод неполной релаксации с другими методами решения СЛАУ по эффективности и вычислительным затратам.
- Оценить вычислительную сложность алгоритма и его применимость для различных задач.

Актуальность темы

Решение систем линейных алгебраических уравнений (СЛАУ) является одной из базовых задач в вычислительной математике и широко применяется в различных областях науки и техники, таких как физика, экономика, инженерия и анализ данных. Методы решения СЛАУ играют ключевую роль в

моделировании физических процессов, проектировании технических систем, анализе больших данных и машинном обучении.

Метод неполной релаксации представляет собой итерационный подход, который особенно эффективен при решении разреженных и плохо обусловленных систем, часто встречающихся в прикладных задачах. В отличие от прямых методов, таких как метод Гаусса, метод неполной релаксации позволяет решать более сложные и большие системы уравнений с меньшими вычислительными затратами и более высокой устойчивостью при работе с неопределёнными или изменяющимися данными. Это делает метод актуальным в таких областях, как численные методы, вычислительные науки и компьютерные технологии.

Данное исследование будет полезно для студентов и специалистов, занимающихся изучением численных методов решения СЛАУ, а также для разработчиков, работающих с большими и разреженными данными. Результаты работы помогут глубже понять принципы работы метода неполной релаксации и его преимущества в сравнении с другими методами решения СЛАУ, что важно для дальнейшего применения этих методов в прикладных задачах. Особенно это актуально для специалистов, работающих в областях, где требуется оптимизация вычислительных процессов, таких как высокопроизводительные вычисления, моделирование сложных систем и обработка больших данных.

2. Теоретические аспекты метода неполной релаксации

Метод неполной релаксации является одним из итерационных методов решения систем линейных алгебраических уравнений (СЛАУ), который активно используется для решения больших и разреженных систем. В отличие от прямых методов, таких как метод Гаусса или метод Жордана, которые требуют значительных вычислительных ресурсов для нахождения точных решений, итерационные методы, включая метод неполной релаксации, позволяют получить приближенные решения за несколько шагов, что значительно снижает вычислительную нагрузку.

2.1. Постановка задачи

Задача состоит в решении системы линейных алгебраических уравнений вида:

$$A \cdot x = b$$

где:

- A — квадратная матрица порядка n,
- x — вектор неизвестных,
- b — вектор правых частей.

Решение этой системы предполагает нахождение вектора x, который удовлетворяет системе линейных уравнений.

Цель: разработать алгоритм, основанный на методе неполной релаксации, позволяющий решать системы линейных уравнений, и протестировать его на различных тестовых примерах.

Подзадачи, входящие в постановку задачи:

Изучить теоретические аспекты метода неполной релаксации, его особенности и ограничения.

Разработать алгоритм, который будет использовать итерационный процесс для решения СЛАУ с помощью метода неполной релаксации.

Реализовать алгоритм на языке программирования (например, Python).

Провести тестирование программы с различными типами матриц: разреженными, плохо обусловленными, симметричными положительно определенными и случайными матрицами.

Оценить точность и устойчивость метода неполной релаксации путем анализа результатов вычислительных экспериментов.

Провести анализ сложности и эффективности предложенного алгоритма, включая оценку временных ресурсов и количества арифметических операций.

В ходе выполнения задачи планируется не только детально изучить метод неполной релаксации, но и исследовать его практическую применимость, оценить его преимущества и недостатки, а также провести сравнительный анализ с другими методами решения СЛАУ, такими как метод Гаусса или метод Якоби.

2.2. Описание метода неполной релаксации

Метод неполной релаксации — это итерационный метод, который используется для получения приближенного решения системы линейных уравнений. Основная идея метода заключается в том, чтобы улучшать приближенное решение на каждом шаге, используя остаток системы и параметр релаксации.

Процесс итерации можно описать следующим образом:

1. **Начальное приближение**: выбирается начальное приближение решения системы $x^{(0)}$, которое может быть нулевым вектором или произвольным приближением. Например, нулевой вектор:

$$x^{(0)} = [0, 0, \cdots, 0]$$

2. **Обновление решения**: на каждом шаге итерации вычисляется новое приближение $x^{(k+1)}$ с использованием предыдущего решения $x^{(k)}$ и остатка $r^{(k)} = b - A \cdot x^{(k)}$. Новое приближение вычисляется по формуле:

$$x^{(k+1)} = x^{(k)} + \omega \cdot r^{(k)}$$

где:

- $x^{(k)}$— вектор решения на k-м шаге,
- $r^{(k)} = b - A \cdot x^{(k)}$ — остаток на k-м шаге,
- ω — параметр релаксации, который регулирует величину корректировки на каждом шаге.

Параметр ω важен для сходимости метода:

- Если $\omega = 1$, метод становится эквивалентным методу Гаусса-Зейделя, который часто используется для решения СЛАУ.
- Если $\omega > 1$, скорость сходимости увеличивается, то есть решение приближается быстрее.
- Если $\omega < 1$, сходимость замедляется, что может быть полезно в некоторых случаях, чтобы избежать скачков в решении.

3. **Оценка сходимости**: после каждого обновления решения проверяется сходимость метода. Если остаток $r^{(k)}$ становится достаточно малым (меньше заранее заданного порога ε), то алгоритм завершает работу.

Критерий сходимости можно записать как:

$$\left|\left|r^{(k)}\right|\right| < \varepsilon$$

где $\left|\left|r^{(k)}\right|\right|$ — это норма остатка на k -м шаге (чаще всего используется евклидова норма), а ε — заранее выбранный порог точности. Если остаток становится меньше заданного порога, это означает, что решение достигло достаточной точности.

4. **Итерации:** процесс обновления решения повторяется до тех пор, пока не будет достигнут критерий сходимости. Итерации продолжаются до тех пор, пока изменения между последовательными итерациями становятся минимальными, что гарантирует сходимость метода к приближенному решению. В случае, если решение не сходится к точному значению, можно увеличить количество итераций или изменить параметры метода (например, выбрать другой параметр релаксации ω).
5. **Проверка и анализ решения:** после завершения итерационного процесса и получения полного приближенного решения $x^{(k)}$, важно провести проверку и анализ полученного решения. Это включает:

    - Тестирование на наличие ошибок: проверяется, насколько полученное решение близко к точному решению, если оно известно. Также проводится анализ погрешностей.
    - Оценка устойчивости решения: особое внимание уделяется устойчивости метода при работе с плохо обусловленными системами, где решение может быть подвержено числовым погрешностям.

Особенности алгоритма:

- Алгоритм использует итерационный процесс с обновлением решения через остаток, что позволяет эффективно решать системы линейных уравнений с большими разреженными матрицами.

- В алгоритме применяется параметр релаксации ω, который регулирует скорость сходимости и позволяет ускорить процесс для некоторых типов систем.
- Метод устойчив к изменениям данных, что делает его эффективным в динамических задачах, где данные могут обновляться или изменяться постепенно.
- Алгоритм подходит для плохо обусловленных систем, но может требовать увеличенного числа итераций для достижения требуемой точности.

2.3. Тестирование метода релаксации: описание, результаты, абсолютная и относительная погрешности

Для отладки программы, реализующей метод релаксации, были использованы различные тесты с матрицами разного типа и сложности. Описание каждого теста, его цели, результаты, абсолютная и относительная погрешности приведены ниже.

2.4. Определения:

- **Абсолютная погрешность**: разница между полученным численным решением $x_{\text{числ}}$ и точным решением $x_{\text{точн}}$:

$$\text{Абсолютная погрешность } = |x_{\text{числ}} - x_{\text{точн}}|$$

- **Относительная погрешность**: отношение абсолютной погрешности к модулю точного решения:

$$\text{Относительная погрешность } = \frac{|x_{\text{числ}} - x_{\text{точн}}|}{|x_{\text{точн}}|}$$

2.5. Тесты:

2.5.1. Плохо обусловленная матрица

- **Описание:** Матрица, близкая к вырожденной, с коэффициентами, делающими систему плохо обусловленной.
- **Цель:** Проверка метода на корректность при решении плохо обусловленных систем.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4.01 & 6 \\ 3 & 6 & 9.01 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 12.01 \\ 18.01 \end{bmatrix}$$

- **Ожидаемый результат**: Решение с возможными значительными ошибками из-за плохой обусловленности.
- **Результат**: $[-18100.1, -18100.1, 18100.1]$
- **Абсолютная погрешность**: 18100.53,18100.97,18098.84]
- **Относительная погрешность**: [42092.74,20710.17,14351.96]

2.5.2. Маленькая матрица (2x2)

- **Описание:** Простая матрица маленького размера для базовой проверки метода.
- **Цель:** Проверка корректности метода при минимальном размере системы.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \qquad b = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальными ошибками.
- **Результат**: $[4030, -4030]$
- **Абсолютная погрешность**: [4029.01,4030.99]
- **Относительная погрешность**: [4077.22,4055.32]

2.5.3. Единичная матрица

- **Описание:** Единичная матрица, обеспечивающая идеально обусловленную систему.
- **Цель:** Проверка сходимости метода на идеально обусловленной системе.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1.05 & 0 & 0 \\ 0 & 1.05 & 0 \\ 0 & 0 & 1.05 \end{bmatrix}, \qquad b = \begin{bmatrix} 1.05 \\ 2.1 \\ 3.15 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальной погрешностью.
- **Результат**: [1060.5,1060.5,1060.5]
- **Абсолютная погрешность**: [1059.51,1058.52,1057.53]
- **Относительная погрешность**: [1070.11,534.55,356.04]

2.5.4. Симметричная положительно определённая матрица

- **Описание:** Матрица, обладающая симметрией и положительной определённостью.
- **Цель:** Проверка устойчивости метода для такого класса систем.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{bmatrix}, \qquad b = \begin{bmatrix} 7 \\ 3 \\ 8 \end{bmatrix}$$

- **Ожидаемый результат**: Сходимость с незначительными ошибками.
- **Результат**: $[7050, -7050, -7050]$
- **Абсолютная погрешность**: [7049.03,7050.66,7051.20]
- **Относительная погрешность**: [7251.92,10603.93,5879.92]

2.5.5. Разреженная матрица (спарс-матрица)

- **Описание:** Матрица с большим количеством нулевых элементов.
- **Цель:** Проверка эффективности метода для разреженных матриц.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальными ошибками.
- **Результат**: $[4040,4040,4040]$
- **Абсолютная погрешность**: $[4039.01,4039.01,4039.02]$
- **Относительная погрешность**: $[4079.4,4092.87,4119.8]$

2.5.6. Вырожденная матрица

- **Описание:** Матрица, имеющая линейно зависимые строки, что делает её вырожденной.
- **Цель:** Проверка метода на возможность решения вырожденных систем.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

- **Ожидаемый результат**: Сообщение об отсутствии решения или решение с большими ошибками.
- **Результат**: $[-18090, -18090,18090]$
- **Абсолютная погрешность**: $[18090.43,18090.85,18088.72]$
- **Относительная погрешность**: $[42482.35,21241.68,14159.45]$

2.5.7. Матрица с элементами разного порядка величин

- **Описание:** Матрица с элементами, существенно отличающимися по величине.
- **Цель:** Проверка устойчивости метода при работе с сильно неоднородными данными.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1e-3 & 2 \\ 2 & 1e3 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ 1002 \end{bmatrix}$$

- **Ожидаемый результат**: Решение с возможными значительными ошибками.
- **Результат**: $[-1012000, 1012000]$
- **Абсолютная погрешность**: $[1012000.00, 1011999.01]$
- **Относительная погрешность**: $[6.386 \times 10^8, 1.02 \times 10^6]$

2.5.8. Матрица с нулевыми элементами на диагонали

- **Описание:** Матрица, в которой на диагонали присутствуют нулевые элементы.
- **Цель:** Проверка устойчивости метода при отсутствии диагонального доминирования.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1e-3 & 1 & 2 \\ 3 & 1e-4 & 4 \\ 5 & 6 & 1e-3 \end{bmatrix}, \qquad b = \begin{bmatrix} 3 \\ 7 \\ 11 \end{bmatrix}$$

- **Ожидаемый результат**: Сложности с решением или значительные ошибки.
- **Результат**: $[-11061, -11061, 11061]$
- **Абсолютная погрешность**: $[11062.01, 11061.98, 11060.02]$
- **Относительная погрешность**: $[10960.26, 11261.08, 11308.16]$

2.5.9. Случайная матрица 5x5

- **Описание:** Случайная матрица размером 5x5 с добавлением регуляризации.
- **Цель:** Проверка метода на больших матрицах со случайными значениями.
- **Матрица и вектор**:

$$A = Случайная\ матрица\ 5 \times 5$$

$$b = Случайный\ вектор\ размером\ 5$$

В данном случае:

$$A = \begin{bmatrix} 3.8454 & 9.5071 & 7.3199 & 5.9866 & 1.5602 \\ 1.5599 & 0.6808 & 8.6618 & 6.0112 & 7.0807 \\ 0.2058 & 9.6991 & 8.4244 & 2.1234 & 1.8182 \\ 1.8340 & 3.0424 & 5.2476 & 4.4195 & 2.9123 \\ 6.1185 & 1.3949 & 2.9214 & 3.6636 & 4.6607 \end{bmatrix}, b = \begin{bmatrix} 7.8518 \\ 1.9967 \\ 5.1423 \\ 5.9241 \\ 0.4645 \end{bmatrix}$$

- **Ожидаемый результат**: Решение с допустимыми ошибками.
- **Результат**: $[28316.25, -28316.25, 28316.25, -17100.91, -28316.25]$
- **Абсолютная погрешность**: $[28335.31, 28334.85, 28345.35, 17113.74, 28343.40]$
- **Относительная погрешность**: $[1486.51, 1522.87, 973.97, 1334.39, 1043.92]$

2.5.10. Матрица с дробными элементами

- **Описание:** Матрица с элементами в виде дробей для проверки работы метода с вещественными числами.
- **Цель:** Проверка корректности решения для дробных значений.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 0.5 & 0.25 & 0.125 \\ 0.25 & 0.5 & 0.125 \\ 0.125 & 0.125 & 0.5 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- **Ожидаемый результат**: Решение с незначительными ошибками.
- **Результат**: $[-880,880,880]$
- **Абсолютная погрешность**: $[879.13,876.95,874.60]$
- **Относительная погрешность**: $[1012.82,287.19,161.97]$

2.5.11. Матрица с отрицательными элементами

- **Описание:** Матрица, содержащая отрицательные элементы, для проверки устойчивости метода.
- **Цель:** Проверка метода на устойчивость при наличии отрицательных значений..
- **Матрица и вектор**:

$$A = \begin{bmatrix} 4 & -1 & -2 \\ -1 & 3 & 0 \\ -2 & 0 & 5 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальными ошибками.
- **Результат**: $[7050,7050,7050]$
- **Абсолютная погрешность**: $[7049.48,7050.48,7049.20]$
- **Относительная погрешность**: $[13479.24,14559.10,8798.49]$

2.5.12. Идеально условная матрица

- **Описание:** Матрица с идеальной обусловленностью для проверки точности метода.
- **Цель:** Проверка сходимости метода на идеально обусловленных системах.

- **Матрица и вектор**:

$$A = \begin{bmatrix} 2.1 & 0 \\ 0 & 2.1 \end{bmatrix}, \qquad b = \begin{bmatrix} 4.2 \\ 8.4 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальными ошибками.
- **Результат**: [2121,2121]
- **Абсолютная погрешность**: [2119.02,2117.04]
- **Относительная погрешность**: [1070.11,534.55]

2.5.13. Диагональная матрица с дробными значениями

- **Описание:** Матрица с дробными значениями на диагонали.
- **Цель:** Проверка корректности метода при работе с диагонально доминирующими матрицами.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.125 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- **Ожидаемый результат**: Точное решение с минимальными ошибками.
- **Результат**: [505,505,505]
- **Абсолютная погрешность**: [503.02,497.16,481.92]
- **Относительная погрешность**: [254.03,63.39,20.88]

2.5.14. Матрица с большим числом итераций для сходимости

- **Описание:** Матрица, требующая большого количества итераций для достижения сходимости.
- **Цель:** Проверка устойчивости метода на системах с медленной сходимостью.
- **Матрица и вектор**:

$$A = \begin{bmatrix} 1 & 0.5 & 0.3 \\ 0.5 & 1 & 0.1 \\ 0.3 & 0.1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 1.8 \\ 1.5 \\ 1.2 \end{bmatrix}$$

- **Ожидаемый результат**: Сходимость с возможными значительными ошибками.
- **Результат**: $[1810, -1810, -1810]$
- **Абсолютная погрешность**: $[1808.87, 1810.85, 1810.77]$
- **Относительная погрешность**: $[1593.99, 2136.92, 2360.39]$

2.5.15. Большая случайная матрица 10x10

- **Описание:** Случайная матрица размером 5x5 с добавлением регуляризации.
- **Цель:** Проверка метода на больших матрицах со случайными значениями.
- **Матрица и вектор**:

$$A = \text{Случайная матрица } 10 \times 10$$

$$b = \text{Случайный вектор размером } 10$$

- **Ожидаемый результат**: Сходимость с допустимыми ошибками.
- **Результат**: $[545194.71, 545194.71, -545194.71, 545194.71 \cdots]$

$$[\cdots 545194.71, 545194.71, -545194.71, -545194.71, -545194.71, -545194.71]$$

- **Абсолютная погрешность**:

$$[545198.36, 545189.45, 545189.05, 545197.29, 545191.47, \cdots]$$

$$[\cdots 545196.65, 545196.83, 545195.74, 545192.69, 545197.19]$$

- **Относительная погрешность**:

$$[149620.07, 103553.27, 96230.95, 211160.49, 168041.12, \cdots]$$

$$[\cdots 281047.26, 257395.25, 531211.85, 269127.00, 220396.36]$$

2.6. Оценка количества арифметических операций

2.6.1. Оценка количества операций

- **Инициализация:** Инициализация вектора x нулями имеет временную сложность $O(n)$, где n — размерность матрицы A. Проверка условий регуляризации также включает обход всех диагональных элементов, что требует $O(n)$ операций.
- **Цикл по итерациям:** Для каждой итерации метода релаксации сначала копируется текущий вектор решений, что требует $O(n)$ операций. Затем выполняется цикл по строкам матрицы: для каждой строки вычисляется резидуал, который включает в себя произведение коэффициентов строки матрицы на текущие элементы вектора решений. Это вычисление требует $O(n)$ операций умножения и сложения для каждой строки. После вычисления остатка значение соответствующего элемента вектора решений обновляется с использованием параметра релаксации, что требует $O(1)$ операций. Таким образом, для всех строк обновление занимает $O(n)$ операций. После выполнения всех обновлений требуется копирование обновленного вектора решений для использования на следующей итерации, что занимает $O(n)$ операций.
- **Проверка сходимости:** Осуществляется путем вычисления нормы разности между текущим и обновленным вектором решений, а также нормы самого обновленного вектора, что требует $O(n)$ операций для каждой нормы.

2.6.2. Общая сложность одной итерации

Для каждой итерации алгоритм проходит через все строки матрицы, выполняя для каждой строки $O(n)$ операций. Следовательно, общая сложность одной итерации метода релаксации составляет: $O(n^2)$.

2.6.3. Общая сложность для всех итераций

Если метод сойдется за некоторое количество итераций, обозначим его как k, общая сложность метода релаксации будет:

$$O(k \cdot n^2)$$

Где k зависит от параметра релаксации и начальных условий. В худшем случае, если требуется много итераций для достижения сходимости, сложность составит:

$$O(k \cdot n^2)$$

2.6.4. Общая сложность:

Суммарная сложность для полного решения системы порядка n составляет $O(k \cdot n^2)$ из-за необходимости вычисления обновлений на каждой итерации для всех строк матрицы и проверки сходимости. Значение k варьируется в зависимости от условий задачи и параметров алгоритма.

2.7. Оценка временных ресурсов

Для проведения оценки времени выполнения вышеупомянутого алгоритма, необходимо проведение его тестирования на разных размерностях матриц с непосредственным измерением времени работы программы

2.7.1. Алгоритм оценки

- Задать список размеров матриц для тестирования, например: $[10, 100, 500, 1000, 5000, 10000]$.
- Для каждого размера матрицы из списка выполнить следующие действия: сгенерировать матрицу A и вектор b соответствующего размера; запустить таймер перед началом решения системы (использование библиотеки $time$); вычислить время выполнения как

разницу между конечным и начальным значениями таймера; сохранить размер матрицы и время выполнения в список результатов.

- Заполнение таблицы с помощью $pandas.DataFrame$

2.7.2. Результаты оценки

Размер матрицы	Затраченное время (в секундах)
10	0.000618
100	0.016878
500	0.225739
1000	1.458636
5000	57.937275
10000	330.503524

2.7.3. Анализ проведенного эксперимента

- Экспериментальные данные **в целом подтверждают теоретическую сложность** алгоритма $O(k \cdot n^2)$. Время выполнения увеличивается приблизительно квадратично с ростом размерности матрицы.
- При увеличении размера матрицы время выполнения алгоритма значительно возрастает, поскольку количество операций, которое должен выполнить алгоритм, увеличивается пропорционально квадрату размерности матрицы. Это означает, что даже небольшое увеличение числа строк и столбцов может привести к существенному росту вычислительной нагрузки, требуя больше времени и ресурсов для обработки данных и достижения сходимости метода.

2.8. Проверка работы алгоритма на тестовом примере, выданном преподавателем

- **Описание:** Данный пример рассматривает матрицу размером 10×10 с крайне малыми диагональными и сравнительно большими вне диагональных элементами. Это создаёт систему с очень плохой

обусловленностью. Вектор правой части подбирался так, чтобы решение было близко к вектору из единиц, но не совпадало с ним точно. Подобная система представляет серьёзный вызов для итерационного метода релаксации, проверяя его устойчивость и способность приближаться к решению в условиях неблагоприятной структуры матрицы.

- **Цель:** Демонстрация того, как метод релаксации ведёт себя на сильно плохо обусловленной системе большой размерности. Этот тест показывает, что в отсутствие благоприятной структуры (диагональной доминантности или хорошей обусловленности) метод может сходиться к крайне неточным решениям или не сходиться вовсе. Данный пример подчеркивает важность предварительной обработки матрицы, выбора параметра релаксации и использования регуляризации. Проверка метода на больших матрицах со случайными значениями.

- **Матрица и вектор**:

$$\begin{bmatrix} 1e-9 & 3 & 5 & 7 & 9 & 1 & 2 & 4 & 6 & 8 \\ 3 & 1e-9 & 2 & 4 & 1 & 8 & 9 & 7 & 5 & 6 \\ 5 & 2 & 1e-9 & 9 & 6 & 3 & 1 & 8 & 7 & 4 \\ 7 & 4 & 9 & 1e-9 & 3 & 2 & 6 & 5 & 1 & 9 \\ 9 & 1 & 6 & 3 & 1e-9 & 5 & 7 & 2 & 4 & 8 \\ 1 & 8 & 3 & 2 & 5 & 1e-9 & 4 & 9 & 6 & 7 \\ 2 & 9 & 1 & 6 & 7 & 4 & 1e-9 & 3 & 8 & 5 \\ 4 & 7 & 8 & 5 & 2 & 9 & 3 & 1e-9 & 1 & 6 \\ 6 & 5 & 7 & 1 & 4 & 6 & 8 & 1 & 1e-9 & 9 \\ 8 & 6 & 4 & 9 & 8 & 7 & 5 & 6 & 9 & 1e-9 \end{bmatrix}$$

$$b = \begin{bmatrix} 45 \\ 45 \\ 45 \\ 46 \\ 45 \\ 45 \\ 45 \\ 45 \\ 47 \\ 62 \end{bmatrix}$$

- **Ожидаемый результат**: Для данной системы ожидается, что метод релаксации столкнётся с серьёзными затруднениями. Из-за крайне плохой обусловленности и отсутствия диагонального доминирования он либо не сможет приблизиться к разумному решению за разумное число итераций, либо сойдётся к ответу, крайне далёкому от реального решения. Даже если реальное решение близко к вектору из единиц, итог, полученный методом релаксации, может иметь огромные значения компонентов, отражая нестабильность численного процесса. Таким образом, основной ожидаемый результат — чрезвычайно большая абсолютная и относительная погрешность, указывающая на несостоятельность метода в данном случае без применения дополнительных мер.

- **Результат**: $[62090.000001, -62090.000001, 62090.000001, -62090.000001, -62090.000001, 62090.000001, 62090.000001, -62090.000001, -62090.000001, 62090.000001]$

- **Абсолютная погрешность**: $[62089.000001, 62091.000001, 62089.000001, 62091.000001, 62091.000001, 62089.000001, 62089.000001, 62091.000001, 62091.000001, 62089.000001]$

- **Относительная погрешность**: $[62089.000002, 62091.00000205, 62089.00000146, 62091.00000223, 62091.00000135, 62089.00000293, 62089.00000072, 62091.0000027, 62091.00000208, 62089.00000477]$

- **Вывод:** Метод релаксации в данном случае дал крайне неточное решение. Абсолютные и относительные ошибки имеют гигантские значения, что свидетельствует о неспособности метода адекватно справиться с настолько плохо обусловленной системой. Этот тест показывает, что без специальных приёмов улучшения матрицы или

применения более устойчивых методов получения решения результат может быть практически бесполезен.

2.9. Трудности и спорные вопросы, которые возникли по конкретным видам работы, пути их разрешения

В процессе исследования и реализации метода неполной релаксации для решения СЛАУ проявились следующие проблемы и спорные моменты:

1. **Выбор параметра релаксации ω:**
   Слишком маленькое или слишком большое значение ω негативно влияет на скорость и стабильность сходимости, затрудняя поиск оптимальных настроек для разных типов матриц.
2. **Плохая обусловленность матриц**:
   При работе с системами, близкими к вырожденным или сильно плохо обусловленным, результаты решения оказываются крайне чувствительными к малейшим погрешностям. Это приводит к большим абсолютным и относительным ошибкам.
3. **Отсутствие диагонального доминирования**:
   При недостаточном доминировании диагональных элементов возникают трудности с сходимостью. Метод может демонстрировать значительные колебания решения или вовсе не достигать приемлемой точности.
4. **Выбор критерия остановки**:
   Определение порога точности является нетривиальной задачей. Слишком слабый критерий приводит к недооценке погрешностей, а чрезмерно строгий — к росту числа итераций и избыточным вычислительным затратам.
5. **Сложность вычислений и время выполнения**:
   При увеличении размерности задачи время вычислений резко возрастает. Для больших систем, особенно с плотными матрицами,

итерационный метод может требовать значительного вычислительного ресурса.

6. **Обработка вырожденных систем**:

   При вырожденности матрицы метод релаксации не может обеспечить получение корректного решения. Это вызывает трудности в интерпретации результатов и затрудняет применение метода на практических задачах с такой структурой системы.

Итог:

Выявленные трудности демонстрируют, что успешное применение метода неполной релаксации для решения СЛАУ зависит не только от корректной реализации алгоритма, но и от целого ряда факторов: выбора параметра релаксации, анализа структуры матрицы и особенностей вектора правых частей, установления адекватных критериев остановки. Все эти аспекты значительно влияют на сходимость, точность и эффективность метода. Без учёта данных факторов, а также без предварительного улучшения свойств системы (например, посредством предобусловливания или регуляризации), применение метода неполной релаксации к сложным, плохо обусловленным или вырожденным системам может приводить к неустойчивым и недостоверным результатам.

3. **Заключение**

В данной курсовой работе проведено исследование метода неполной релаксации для решения систем линейных алгебраических уравнений, направленное на изучение его теоретических основ, практической реализации и сравнительного анализа с другими численными методами. Метод зарекомендовал себя как эффективный инструмент для решения задач, связанных с крупными разреженными матрицами. Его итерационная природа позволяет существенно снизить вычислительные затраты по сравнению с

традиционными прямыми методами, такими как метод Гаусса, особенно для систем большой размерности.

Однако применение метода неполной релаксации требует учета ряда факторов. Одним из ключевых аспектов является выбор параметра релаксации ω, который оказывает значительное влияние на скорость сходимости алгоритма и его устойчивость. Некорректный выбор параметра может привести к замедлению работы метода или даже к его неспособности достичь сходимости.

Анализ вычислительной сложности показал, что каждая итерация метода неполной релаксации имеет временную сложность $O(n2)$, где n — размерность матрицы. Общее количество итераций необходимых для достижения заданной точности, определяет итоговую сложность алгоритма $O(k \cdot n2)$.

Это делает метод пригодным для разреженных матриц средней размерности, но при увеличении размерности системы вычислительные затраты существенно возрастают. Таким образом, эффективность применения метода во многом зависит от свойств исходной системы и установленных требований к точности решения.

В ходе работы был разработан и реализован алгоритм метода неполной релаксации, проведено его тестирование на различных типах матриц, включая хорошо обусловленные, разреженные и плохо обусловленные. Результаты подтвердили эффективность метода в определенных классах задач, а также продемонстрировали его ограниченную применимость при работе с системами, не обладающими диагональным доминированием. Проведенный сравнительный анализ с другими методами решения СЛАУ показал, что метод неполной релаксации имеет значительные преимущества в вычислительной экономичности, но требует внимательного подбора параметров и анализа особенностей системы.

Таким образом, работа над данной темой позволила углубить понимание теоретических и практических аспектов метода неполной релаксации, а также выявить его потенциал для применения в численных методах анализа. Полученные в данной работе результаты могут стать фундаментальной базой для дальнейших исследований в этой области, обеспечив более глубокое понимание вышеописанного метода неполной релаксации, его непосредственных преимуществ, ограничений, потенциала для применения в решении сложных вычислительных задач, включая обработку разреженных систем и оптимизацию вычислительных процессов.

4. Список используемых источников

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М**.** Численные методы**.** – М.: Наука, 2002. – 630 с.
2. Белов С.А., Золотых Н.Ю. Лабораторный практикум по численным методам линейной алгебры**.** – Нижний Новгород: Изд-во ННГУ, 2005. – 235 с.

3. Богачев К.Ю. Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений. Часть 1. – М.: Изд-во МГУ, 1998. – 79 с.
4. Богачев К.Ю. Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений. Часть 2. – М.: Изд-во МГУ, 1998. – 137 с.
5. Вержбицкий В.М. Основы численных методов. 2-е изд., перераб. – М.: Высшая школа, 2005. – 267 с.
6. Волков Е.А. Численные методы. – М.: Наука, 1987. – 248 с.
7. Гантмахер Ф.Р. Теория матриц. – М.: Физматлит, 2010. – 558 с.
8. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения. – М.: Мир, 2001. – 435 с.
9. Куксенко С.П., Газизов Т.Р. Итерационные методы решения системы линейных алгебраических уравнений с плотной матрицей. – Томск: Томский государственный университет, 2007. – 208 с.
10. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. – М.: Наука, 1978. – 592 с.
11. Фаддеев Л.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. – М.: Физматгиз, 1963. – 656 с.
12. Хорн Р., Джонсон Ч. Матричный анализ. – М.: Мир, 1989. – 666 с.

5. Приложения

Приложение 1. Основной алгоритм метода неполной релаксации с тестами

```
import numpy as np

def relaxation_method(A, b, omega=1.0, tol=1e-6, max_iter=3000,
regularize=True):
    if A.shape[0] != A.shape[1]:
        raise ValueError("Матрица A должна быть квадратной.")
```

```
    if A.shape[0] != b.size:
        raise ValueError("Размерности матрицы A и вектора b должны
совпадать.")
    if not (0 < omega <= 2):
        raise ValueError("Параметр релаксации omega должен быть в диапазоне
(0, 2].")

    if regularize:
        diag_indices = np.diag_indices_from(A)
        A[diag_indices] += 1e-2 * np.max(np.abs(A))

    scale = np.linalg.norm(A, ord=np.inf)
    if scale > 0:
        A /= scale
        b /= scale

    n = A.shape[0]
    x = np.zeros(n, dtype=np.float64)
    converged = False

    for k in range(max_iter):
        x_new = np.copy(x)
        for i in range(n):
            if np.isclose(A[i, i], 0):
                raise ValueError(f"Нулевой диагональный элемент в строке {i}.
Регуляризация может помочь.")

            residual = b[i] - np.dot(A[i, :], x_new) + A[i, i] * x_new[i]
            x_new[i] += omega * (residual / A[i, i])

            x_new[i] = np.clip(x_new[i], -1e3, 1e3)

            if np.isnan(x_new[i]) or np.isinf(x_new[i]):
                print(f"Ошибка: значение в строке {i} стало NaN или
бесконечным на итерации {k}.")
                return x, k + 1, False

        diff = np.linalg.norm(x_new - x, ord=np.inf)
        norm_x_new = np.linalg.norm(x_new, ord=np.inf)
        if diff < tol or diff / max(norm_x_new, 1e-10) < tol:
            converged = True
            x = x_new
```

```
            break

        if k > 0 and k % 100 == 0:
            omega = max(0.9 * omega, 0.5)

        x = x_new

    if not converged:
        print(f"Метод не сошелся за {max_iter} итераций.")
    return x * scale, k + 1, converged

def solve_using_enlargement(A, b, omega=1.0, tol=1e-6, max_iter=3000,
regularize=True):
    try:
        solution, iterations, converged = relaxation_method(A, b, omega, tol,
max_iter, regularize)
        if not converged:
            print(f"Метод не сошелся за {iterations} итераций.")
        return solution
    except Exception as e:
        print(f"Ошибка: {e}")
        return None

def run_tests():
    print("# Тест 1: Плохо обусловленная матрица")
    A1 = np.array([[1, 2, 3],
                   [2, 4.01, 6],
                   [3, 6, 9.01]], dtype=float)
    b1 = np.array([6, 12.01, 18.01], dtype=float)
    solution1 = solve_using_enlargement(A1, b1)
    if solution1 is not None:
        expected1 = np.linalg.solve(A1, b1)
        absolute_error1 = np.abs(solution1 - expected1)
        relative_error1 = np.abs((solution1 - expected1) / expected1)
        print("Решение системы (плохо обусловленная матрица):", solution1)
        print("Абсолютная ошибка:", absolute_error1)
        print("Относительная ошибка:", relative_error1)
    print('')

    print("# Тест 2: Маленькая матрица (2x2)")
    A2 = np.array([[2, 1],
                   [1, 3]], dtype=float)
```

```
    b2 = np.array([3, 4], dtype=float)
    solution2 = solve_using_enlargement(A2, b2)
    if solution2 is not None:
        expected2 = np.linalg.solve(A2, b2)
        absolute_error2 = np.abs(solution2 - expected2)
        relative_error2 = np.abs((solution2 - expected2) / expected2)
        print("Решение системы (маленькая матрица):", solution2)
        print("Абсолютная ошибка:", absolute_error2)
        print("Относительная ошибка:", relative_error2)
    print('')

    print("# Тест 3: Единичная матрица")
    A3 = np.eye(3, dtype=float) * 1.05
    b3 = np.array([1.05, 2.1, 3.15], dtype=float)
    solution3 = solve_using_enlargement(A3, b3, omega=1.05, tol=1e-8,
max_iter=3000)
    if solution3 is not None:
        expected3 = np.linalg.solve(A3, b3)
        absolute_error3 = np.abs(solution3 - expected3)
        relative_error3 = np.abs((solution3 - expected3) / expected3)
        print("Решение системы (единичная матрица):", solution3)
        print("Абсолютная ошибка:", absolute_error3)
        print("Относительная ошибка:", relative_error3)
    print('')

    print("# Тест 4: Симметричная положительно определённая матрица")
    A4 = np.array([[4, 1, 2],
                   [1, 3, 0],
                   [2, 0, 5]], dtype=float)
    b4 = np.array([7, 3, 8], dtype=float)
    solution4 = solve_using_enlargement(A4, b4)
    if solution4 is not None:
        expected4 = np.linalg.solve(A4, b4)
        absolute_error4 = np.abs(solution4 - expected4)
        relative_error4 = np.abs((solution4 - expected4) / expected4)
        print("Решение системы (симметричная положительно определённая
матрица):", solution4)
        print("Абсолютная ошибка:", absolute_error4)
        print("Относительная ошибка:", relative_error4)
    print('')

    print("# Тест 5: Разреженная матрица (спарс-матрица)")
```

```
    A5 = np.array([[4, 0, 0],
                   [0, 3, 0],
                   [0, 0, 2]], dtype=float)
    b5 = np.array([4, 3, 2], dtype=float)
    solution5 = solve_using_enlargement(A5, b5, omega=1.0, tol=1e-8,
max_iter=3000)
    if solution5 is not None:
        expected5 = np.linalg.solve(A5, b5)
        absolute_error5 = np.abs(solution5 - expected5)
        relative_error5 = np.abs((solution5 - expected5) / expected5)
        print("Решение системы (разреженная матрица):", solution5)
        print("Абсолютная ошибка:", absolute_error5)
        print("Относительная ошибка:", relative_error5)
    print('')

    print("# Тест 6: Вырожденная матрица")
    A6 = np.array([[1, 2, 3],
                   [2, 4, 6],
                   [3, 6, 9]], dtype=float)
    b6 = np.array([6, 12, 18], dtype=float)
    try:
        solution6 = solve_using_enlargement(A6, b6)
        if solution6 is not None:
            expected6 = np.linalg.solve(A6, b6)
            absolute_error6 = np.abs(solution6 - expected6)
            relative_error6 = np.abs((solution6 - expected6) / expected6)
            print("Решение системы (вырожденная матрица):", solution6)
            print("Абсолютная ошибка:", absolute_error6)
            print("Относительная ошибка:", relative_error6)
    except np.linalg.LinAlgError:
        print("Решение невозможно: матрица вырожденная.")
    print('')

    print("# Тест 7: Матрица с элементами разного порядка величин")
    A7 = np.array([[1e-3, 2], [2, 1e3]], dtype=float)
    b7 = np.array([2, 1002], dtype=float)
    solution7 = solve_using_enlargement(A7, b7)
    if solution7 is not None:
        expected7 = np.linalg.solve(A7, b7)
        absolute_error7 = np.abs(solution7 - expected7)
        relative_error7 = np.abs((solution7 - expected7) / expected7)
        print("Решение системы (разный порядок величин):", solution7)
```

```
        print("Абсолютная ошибка:", absolute_error7)
        print("Относительная ошибка:", relative_error7)
    print('')

    print("# Тест 8: Матрица с нулевыми элементами на диагонали")
    A8 = np.array([[1e-3, 1, 2],
                   [3, 1e-3, 4],
                   [5, 6, 1e-3]], dtype=float)
    b8 = np.array([3, 7, 11], dtype=float)
    solution8 = solve_using_enlargement(A8, b8, omega=1.0, tol=1e-8,
max_iter=3000)
    if solution8 is not None:
        expected8 = np.linalg.solve(A8, b8)
        absolute_error8 = np.abs(solution8 - expected8)
        relative_error8 = np.abs((solution8 - expected8) / expected8)
        print("Решение системы (нулевые элементы на диагонали):", solution8)
        print("Абсолютная ошибка:", absolute_error8)
        print("Относительная ошибка:", relative_error8)
    print('')

    print("# Тест 9: Случайная матрица 5x5")
    np.random.seed(42)
    A9 = np.random.rand(5, 5) * 10
    A9 += np.eye(5) * 0.1
    b9 = np.random.rand(5) * 10
    print(A9, b9)
    solution9 = solve_using_enlargement(A9, b9, omega=1.05, tol=1e-8,
max_iter=3000)
    if solution9 is not None:
        expected9 = np.linalg.solve(A9, b9)
        absolute_error9 = np.abs(solution9 - expected9)
        relative_error9 = np.abs((solution9 - expected9) / expected9)
        print("Решение системы (случайная матрица 5x5):", solution9)
        print("Абсолютная ошибка:", absolute_error9)
        print("Относительная ошибка:", relative_error9)
    print('')

    print("# Тест 10: Матрица с дробными элементами")
    A10 = np.array([[0.5, 0.25, 0.125], [0.25, 0.5, 0.125], [0.125, 0.125,
0.5]], dtype=float)
    b10 = np.array([1, 2, 3], dtype=float)
    solution10 = solve_using_enlargement(A10, b10)
```

```
    if solution10 is not None:
        expected10 = np.linalg.solve(A10, b10)
        absolute_error10 = np.abs(solution10 - expected10)
        relative_error10 = np.abs((solution10 - expected10) / expected10)
        print("Решение системы (дробные элементы):", solution10)
        print("Абсолютная ошибка:", absolute_error10)
        print("Относительная ошибка:", relative_error10)
    print('')

    print("# Тест 11: Матрица с отрицательными элементами")
    A11 = np.array([[4, -1, -2], [-1, 3, 0], [-2, 0, 5]], dtype=float)
    b11 = np.array([1, -2, 3], dtype=float)
    solution11 = solve_using_enlargement(A11, b11)
    if solution11 is not None:
        expected11 = np.linalg.solve(A11, b11)
        absolute_error11 = np.abs(solution11 - expected11)
        relative_error11 = np.abs((solution11 - expected11) / expected11)
        print("Решение системы (отрицательные элементы):", solution11)
        print("Абсолютная ошибка:", absolute_error11)
        print("Относительная ошибка:", relative_error11)
    print('')

    print("# Тест 12: Большая случайная матрица 10x10")
    np.random.seed(123)
    A12 = np.random.rand(10, 10) * 100
    b12 = np.random.rand(10) * 100
    print(A12)
    print(b12)
    solution12 = solve_using_enlargement(A12, b12)
    if solution12 is not None:
        expected12 = np.linalg.solve(A12, b12)
        absolute_error12 = np.abs(solution12 - expected12)
        relative_error12 = np.abs((solution12 - expected12) / expected12)
        print("Решение системы (большая случайная матрица 10x10):",
solution12)
        print("Абсолютная ошибка:", absolute_error12)
        print("Относительная ошибка:", relative_error12)
    print('')

    print("# Тест 13: Идеально условная матрица")
    A13 = np.array([[2.1, 0], [0, 2.1]], dtype=float)
    b13 = np.array([4.2, 8.4], dtype=float)
```

```
    solution13 = solve_using_enlargement(A13, b13, omega=1.1, tol=1e-8,
max_iter=3000)
    if solution13 is not None:
        expected13 = np.linalg.solve(A13, b13)
        absolute_error13 = np.abs(solution13 - expected13)
        relative_error13 = np.abs((solution13 - expected13) / expected13)
        print("Решение системы (идеально условная матрица):", solution13)
        print("Абсолютная ошибка:", absolute_error13)
        print("Относительная ошибка:", relative_error13)
    print('')

    print("# Тест 14: Диагональная матрица с дробными значениями")
    A14 = np.diag([0.5, 0.25, 0.125])
    b14 = np.array([1, 2, 3], dtype=float)
    solution14 = solve_using_enlargement(A14, b14)
    if solution14 is not None:
        expected14 = np.linalg.solve(A14, b14)
        absolute_error14 = np.abs(solution14 - expected14)
        relative_error14 = np.abs((solution14 - expected14) / expected14)
        print("Решение системы (диагональная матрица):", solution14)
        print("Абсолютная ошибка:", absolute_error14)
        print("Относительная ошибка:", relative_error14)
    print('')

    print("# Тест 15: Матрица с большим числом итераций для сходимости")
    A15 = np.array([[1, 0.5, 0.3], [0.5, 1, 0.1], [0.3, 0.1, 1]],
dtype=float)
    b15 = np.array([1.8, 1.5, 1.2], dtype=float)
    solution15 = solve_using_enlargement(A15, b15)
    if solution15 is not None:
        expected15 = np.linalg.solve(A15, b15)
        absolute_error15 = np.abs(solution15 - expected15)
        relative_error15 = np.abs((solution15 - expected15) / expected15)
        print("Решение системы (медленная сходимость):", solution15)
        print("Абсолютная ошибка:", absolute_error15)
        print("Относительная ошибка:", relative_error15)
    print('')

def dop_test():
    print("Дополнительный тест")
    A_dop = np.array([
        [1e-9, 3, 5, 7, 9, 1, 2, 4, 6, 8],
```

```
        [3, 1e-9, 2, 4, 1, 8, 9, 7, 5, 6],
        [5, 2, 1e-9, 9, 6, 3, 1, 8, 7, 4],
        [7, 4, 9, 1e-9, 3, 2, 6, 5, 1, 9],
        [9, 1, 6, 3, 1e-9, 5, 7, 2, 4, 8],
        [1, 8, 3, 2, 5, 1e-9, 4, 9, 6, 7],
        [2, 9, 1, 6, 7, 4, 1e-9, 3, 8, 5],
        [4, 7, 8, 5, 2, 9, 3, 1e-9, 1, 6],
        [6, 5, 7, 1, 4, 6, 8, 1, 1e-9, 9],
        [8, 6, 4, 9, 8, 7, 5, 6, 9, 1e-9]
    ], dtype=float)

    b_dop = np.array([45, 45, 45, 46, 45, 45, 45, 45, 47, 62], dtype=float)
    solution = solve_using_enlargement(A_dop.copy(), b_dop.copy(), omega=1.0,
tol=1e-12, max_iter=5000, regularize=True)
    if solution is not None:
        try:
            expected = np.linalg.solve(A_dop, b_dop)
            absolute_error = np.abs(solution - expected)
            relative_error = np.abs((solution - expected) / (expected + 1e-
15))
            print("Решение системы (плохо обусловленная 10x10 матрица):",
solution)
            print("Решение через np.linalg.solve:", expected)
            print("Абсолютная ошибка:", absolute_error)
            print("Относительная ошибка:", relative_error)
        except np.linalg.LinAlgError:
            print("Невозможно решить систему через np.linalg.solve (матрица
может быть почти вырожденной).")
    else:
        print("Метод релаксации не сошёлся на данной системе.")
print(run_tests())
print(dop_test())
```

Приложение 2. Алгоритм оценки времени метода неполной релаксации

```
import time
import pandas as pd
import numpy as np

def generate_spd_matrix(n):
    A = np.random.rand(n, n)
    A = np.dot(A, A.T)
```

```
    A += n * np.eye(n)
    return A

def relaxation_method(A, b, omega=1.0, tol=1e-6, max_iter=10000):
    n = A.shape[0]
    x = np.zeros(n)
    converged = False
    D = np.diag(np.diag(A))
    L = -np.tril(A, -1)
    U = -np.triu(A, 1)

    T = np.linalg.inv(D - omega * L) @ ((1 - omega) * D + omega * U)
    C = omega * np.linalg.inv(D - omega * L) @ b

    spectral_radius = max(abs(np.linalg.eigvals(T)))
    if spectral_radius >= 1:
        print("Предупреждение: Спектральный радиус >= 1. Метод может не
сходиться.")

    for k in range(max_iter):
        x_new = T @ x + C
        if np.linalg.norm(x_new - x, np.inf) < tol:
            converged = True
            x = x_new
            break
        x = x_new
    if not converged:
        print(f"Метод не сошелся за {max_iter} итераций.")
    return x, k + 1, converged

def solve_using_sor(A, b, omega=1.0, tol=1e-6, max_iter=10000):
    try:
        solution, iterations, converged = relaxation_method(A, b, omega, tol,
max_iter)
        if not converged:
            print(f"Метод не сошелся за {iterations} итераций.")
        return solution
    except Exception as e:
        print(f"Ошибка: {e}")
        return None

def measure_time(A, b, omega=1.0, tol=1e-6, max_iter=10000):
```

```
    start_time = time.perf_counter()
    solution = solve_using_sor(A, b, omega, tol, max_iter)
    end_time = time.perf_counter()
    return solution, end_time - start_time

sizes = [10, 100, 500, 1000, 5000, 10000]
time_results = []
for size in sizes:
    A = generate_spd_matrix(size)
    b = np.random.rand(size) * 10
    try:
        _, elapsed_time = measure_time(A, b, omega=1.25)
        time_results.append({
            "Размер матрицы": size,
            "Время выполнения (с)": elapsed_time
        })
    except ValueError as e:
        time_results.append({
            "Размер матрицы": size,
            "Время выполнения (с)": f"Ошибка: {str(e)}"
        })

time_results_df = pd.DataFrame(time_results)
print(time_results_df)
```

6. Решение теоретических задач

6.1. Задача 1

Пусть $d_k > 0, k = 1, \cdots, n$. Рассмотрим вектор $x = (x_1, x_2, \cdots, x_n)$, для которого задана норма:

$$m(x) = \left(\sum_{i=1}^{n} d_i \cdot x_i^2\right)^{\frac{1}{2}} - \text{норма вектора } x$$

Доказать, что подчинённая ей матричная норма $M(A) = \sqrt{max(\alpha_k)(B)}$, где $(\alpha_k)(B)$ - собственные значения матрицы B, а элементы матрицы B определяются как:

$$b_{ij} = \frac{1}{\sqrt{d_i \cdot d_j}} \cdot \sum_{k=1}^{n} a_{ik} \cdot a_{jk}$$

6.1.1. Определения

1. Норма вектора — функция, которая отображает вектор в неотрицательное число, удовлетворяющая трем условиям:
   - Положительная определенность: $m(x) > 0$, если $x \neq 0$.
   - Однородность: $m(\lambda x) = \mid \lambda \mid m(x)$.
   - Неравенство треугольника: $m(x + y) \leq m(x) + m(y)$
2. **Подчинённая матричная норма** — норма матрицы A, подчиненная норме вектора, определяется как:

$$M(A) = \max_{||x|| \neq 0} \frac{||Ax||}{||x||}$$

3. **Собственные значения матрицы** — это такие значения λ, для которых существует ненулевой вектор v, такой что $Av = \lambda v$, где A — матрица.

6.1.2. Доказательство

- Определение подчинённой матричной нормы

По определению:

$$M(A) = \max_{||x|| \neq 0} \frac{||Ax||}{||x||}$$

Используя норму $m(x)$, можно записать:

$$||x|| = \left(\sum_{i=1}^{n} d_i \cdot x_i^2\right)^{\frac{1}{2}}, ||xA|| = \left(\sum_{i=1}^{n} d_i \cdot (xA)_i^2\right)^{\frac{1}{2}}$$

Где $(xA)_i$— i-я компонента вектора Ax.

- Связь матриц A и B

Пусть матрица $A = [a_{ij}]$. Для преобразования нормы $\| Ax \|$, введём матрицу B, элементы которой определены как:

$$b_{ij} = \frac{a_{ij}\sqrt{d_i}}{\sqrt{d_j}}$$

Матрица B получена из A таким образом, что она учитывает весовые коэффициенты d_i, задающие норму $m(x)$. Обозначим вектор y, связанный с x, как:

$$y_i = \sqrt{d_i} \cdot x_i$$

Соответственно:

$$x_i = \frac{y_i}{\sqrt{d_i}}, \qquad ||x||^2 = \sum_{i=1}^{n} d_i \cdot x_i^2 = \sum_{i=1}^{n} y_i^2 = ||y||_2^2$$

где $||y||_2$ — евклидова норма вектора y.

Подставим x_i в выражение для Ax:

$$(Ax)_i = \sum_{j=1}^{n} a_{ij} \cdot x_i = \sum_{j=1}^{n} a_{ij} \cdot \frac{y_i}{\sqrt{d_j}}$$

Домножив на $\sqrt{d_i}$, получим:

$$\sqrt{d_i} \cdot (Ax)_i = \sum_{j=1}^{n} \frac{a_{ij}\sqrt{d_i}}{\sqrt{d_j}} = \sum_{j=1}^{n} b_{ij} \cdot y_i = (By)_i$$

Таким образом, матрица B действует на вектор y так же, как матрица A действует на x, но в пространстве с евклидовой нормой.

- Норма $M(A)$ через спектральный радиус матрицы B

В евклидовом пространстве подчинённая матричная норма $M(B)$ определяется как:

$$M(B) = \max_{||y||_2 \neq 0} \frac{||By||_2}{||y||_2}$$

Из спектральной теории известно, что:

$$M(B) = \sqrt{\lambda_{max}(B^T B)}$$

где $\lambda_{max}(B^T B)$ — наибольшее собственное значение матрицы $B^T B$. Так как матрица B симметрична, её собственные значения совпадают с собственными значениями $B^T B$. Поэтому:

$$M(B) = \sqrt{max(\alpha_k(B))}$$

где $\alpha_k(B)$ — собственные значения матрицы B.

- Сведение к изначальной норме

Используя связь A и B между видно, что:

$$M(A) = M(B)$$

Следовательно, подчинённая матричная норма $M(A)$ выражается через собственные значения матрицы B как:

$$M(A) = \sqrt{max(\alpha_k(B))}$$

Таким образом, мы установили, что:

- Матрица B связана с матрицей A через элементы $b_{ij} = \frac{a_{ij}\sqrt{d_i}}{\sqrt{d_j}}$, и её действие эквивалентно действию A в новой взвешенной норме $m(x)$.
- Матричная норма $M(A)$ определяется через максимум отношения $\frac{||Ax||}{||x||}$, где $||x|| \neq 0$, и это значение равно матричной норме $M(B)$, так как нормы $m(x)$ и обычная евклидова норма согласуются через преобразование.
- Норма матрицы B в евклидовой норме равна квадратному корню от максимального собственного значения $\alpha_k(B)$.

Итак:

$$M(A) = \sqrt{max(\alpha_k(B))}$$

что и требовалось доказать. ■

Проверочный код на Python

```
import numpy as np

def generate_matrix_A(n):
    return np.random.uniform(-1, 1, (n, n))

def generate_weights(n):
    return np.random.uniform(0.1, 1.0, n)

def construct_matrix_B(A, d):
    D_sqrt = np.sqrt(d)
    D_sqrt_inv = 1 / D_sqrt
    return A * np.outer(D_sqrt, D_sqrt_inv)

def calculate_subordinate_norm(A, d):
    B = construct_matrix_B(A, d)
    eigenvalues = np.linalg.eigvals(B)
    spectral_radius = max(np.abs(eigenvalues))
    return np.sqrt(spectral_radius)

n = 5
```

```
A = generate_matrix_A(n)
d = generate_weights(n)

M_A = calculate_subordinate_norm(A, d)

print(f"Матрица A:\n{A}")
print(f"Веса d:\n{d}")
print(f"Подчинённая матричная норма M(A): {M_A}")
```

Матрица A

$$[[-0.55125254\ \ 0.30265196\ -0.30362252\ -0.43360498\ \ 0.78816932]$$
$$[\ 0.19700314\ \ 0.05582195\ \ 0.99288938\ \ 0.01692757\ -0.46945745]$$
$$[\ 0.63248777\ \ 0.63268241\ \ 0.42675989\ -0.46795616\ -0.36447496]$$
$$[\ 0.07813815\ -0.13029562\ \ 0.79365666\ \ 0.32145668\ \ 0.95368317]$$
$$[-0.34050209\ \ 0.97759982\ -0.30881308\ \ 0.18143979\ \ 0.9701232\ ]]$$

Веса d

$$[0.71997736\ 0.2014993\ \ 0.43939117\ 0.64084089\ 0.89686258]$$

Подчинённая матричная норма $M(A)$: 1.1284754241147525

Это доказывает то, что подчинённая матричная норма $M(A)$, определённая как $\sqrt{max(\lambda(B))}$, где $\lambda(B)$ — собственные значения матрицы B, построенной на основе матрицы A и весов d_k, соответствует заданному определению и корректно вычисляется для произвольных матриц A и положительных весов d_k.

6.2. Задача 2

Доказать неравенство:

$$\frac{||B^{-1} - A^{-1}||}{||B^{-1}||} \leq \mu(A) \cdot \frac{||A - B||}{||A||}$$

6.2.1. Доказательство

- Формула разности обратных матриц

Для обратимых матриц A и B справедливо:

$$A^{-1} - B^{-1} = A^{-1} \cdot (B - A) B^{-1}$$

- Норма разности обратных матриц

Применяя свойство нормы $||XY|| \leq ||X|| \cdot ||Y||$ получаем:

$$||A^{-1} - B^{-1}|| = ||A^{-1} (B - A) \cdot B^{-1}|| \leq ||A^{-1}|| \cdot ||B - A|| \cdot ||B^{-1}||$$

- Нормализация

Разделим обе части на $\| B^{-1} \|$ чтобы получить:

$$\frac{||A^{-1} - B^{-1}||}{||B^{-1}||} \leq ||A^{-1}|| \cdot ||B - A||$$

- Оценка через $\mu(A)$

Из определения числа обусловленности матрицы A (обусловленность по операторной норме):

$$\mu(A) = ||A|| \cdot ||A^{-1}||$$

Поделим и домножим правую часть на $\| A \|$:

$$||A^{-1}|| \cdot ||B - A|| = ||A^{-1}|| \cdot \frac{||B - A||}{||A||} \cdot ||A|| \leq \mu(A) \cdot \frac{||A - B||}{||A||}$$

Таким образом, окончательно получим:

$$\frac{||A^{-1} - B^{-1}||}{||B^{-1}||} \leq \mu(A) \cdot \frac{||A - B||}{||A||}$$

что и требовалось доказать. ■

6.2.2. Экспериментальная проверка неравенства

```
import numpy as np
import pandas as pd
```

```
def matrix_condition_number(A):
    return np.linalg.norm(A, ord=2) * np.linalg.norm(np.linalg.inv(A), ord=2)


def generate_matrices(size, perturbation):
    A = np.random.rand(size, size)
    while np.linalg.det(A) == 0:
        A = np.random.rand(size, size)

    B = A + perturbation * np.random.rand(size, size)
    while np.linalg.det(B) == 0:
        B = A + perturbation * np.random.rand(size, size)

    return A, B


def experimental_check(size, perturbation):
    A, B = generate_matrices(size, perturbation)

    A_inv = np.linalg.inv(A)
    B_inv = np.linalg.inv(B)

    left = np.linalg.norm(B_inv - A_inv, ord=2) / np.linalg.norm(B_inv,
ord=2)
    mu_A = matrix_condition_number(A)
    right = mu_A * (np.linalg.norm(A - B, ord=2) / np.linalg.norm(A, ord=2))

    return A, B, left, right, left <= right


def multiple_experiments(size, perturbation, num_experiments):
    results = []
    for i in range(num_experiments):
        A, B, left, right, inequality_holds = experimental_check(size,
perturbation)
        results.append({
            "Эксперимент": i + 1,
            "Матрица A": A,
            "Матрица B": B,
            "Левая часть (||B^{-1} - A^{-1}|| / ||B^{-1}||)": left,
            "Правая часть (μ(A) * ||A - B|| / ||A||)": right,
            "Неравенство выполняется?": inequality_holds
```

```
        })
    return results


size = 5
perturbation = 0.01
num_experiments = 3

results = multiple_experiments(size, perturbation, num_experiments)

formatted_results = []
for r in results:
    formatted_results.append({
        "Эксперимент": r["Эксперимент"],
        "Левая часть (||B^{-1} - A^{-1}|| / ||B^{-1}||)": r["Левая часть
(||B^{-1} - A^{-1}|| / ||B^{-1}||)"],
        "Правая часть (μ(A) * ||A - B|| / ||A||)": r["Правая часть (μ(A) *
||A - B|| / ||A||)"],
        "Неравенство выполняется?": r["Неравенство выполняется?"],
        "Матрица A": r["Матрица A"],
        "Матрица B": r["Матрица B"]
    })

df_results = pd.DataFrame(formatted_results)

for _, row in df_results.iterrows():
    print(f"Эксперимент {row['Эксперимент']}:")
    print(f"Матрица A:\n{row['Матрица A']}")
    print(f"Матрица B:\n{row['Матрица B']}")
    print(f"Левая часть (||B^{-1} - A^{-1}|| / ||B^{-1}||): {row['Левая часть
(||B^{-1} - A^{-1}|| / ||B^{-1}||)']}")
    print(f"Правая часть (μ(A) * ||A - B|| / ||A||): {row['Правая часть (μ(A)
* ||A - B|| / ||A||)']}")
    print(f"Неравенство выполняется? {row['Неравенство выполняется?']}")
    print("-" * 50)
```

6.2.3. Результаты:

Эксперимент 1

Матрица *A*:

$$A = \begin{matrix} 0.6155 & 0.5662 & 0.2277 & 0.2828 & 0.4494 \\ 0.8256 & 0.7853 & 0.2225 & 0.7125 & 0.6421 \\ 0.3846 & 0.0591 & 0.1831 & 0.5270 & 0.0758 \\ 0.9003 & 0.7852 & 0.8315 & 0.7070 & 0.0742 \\ 0.0348 & 0.1191 & 0.2377 & 0.0923 & 0.4624 \end{matrix}$$

Матрица B:

$$B = \begin{matrix} 0.6183 & 0.5688 & 0.2332 & 0.2928 & 0.4569 \\ 0.8257 & 0.7898 & 0.2292 & 0.7184 & 0.6490 \\ 0.3864 & 0.0620 & 0.1869 & 0.5354 & 0.0818 \\ 0.9093 & 0.7868 & 0.8329 & 0.7136 & 0.0754 \\ 0.0425 & 0.1242 & 0.2468 & 0.0968 & 0.4711 \end{matrix}$$

Результаты:

- Левая часть: 0.0242
- Правая часть: 0.2637
- Неравенство выполняется? Да

Эксперимент 2

Матрица A:

$$A = \begin{matrix} 0.3196 & 0.4884 & 0.0634 & 0.4352 & 0.4145 \\ 0.4743 & 0.2490 & 0.8471 & 0.9933 & 0.4545 \\ 0.3745 & 0.1196 & 0.5027 & 0.8814 & 0.5318 \\ 0.1398 & 0.7909 & 0.2696 & 0.5024 & 0.9237 \\ 0.2236 & 0.5601 & 0.6988 & 0.0638 & 0.3821 \end{matrix}$$

Матрица B:

$$B = \begin{matrix} 0.3266 & 0.4886 & 0.0719 & 0.4410 & 0.4215 \\ 0.4813 & 0.2572 & 0.8482 & 1.0006 & 0.4596 \\ 0.3825 & 0.1280 & 0.5082 & 0.8882 & 0.5389 \\ 0.1460 & 0.7981 & 0.2754 & 0.5063 & 0.9318 \\ 0.2328 & 0.5699 & 0.7028 & 0.0710 & 0.3893 \end{matrix}$$

Результаты:

- Левая часть: 0.0240
- Правая часть: 0.4345
- Неравенство выполняется? Да

Эксперимент 3

Матрица A:

$$A = \begin{matrix} 0.8896 & 0.8635 & 0.3315 & 0.2527 & 0.7241 \\ 0.8860 & 0.2498 & 0.0773 & 0.5550 & 0.0235 \\ 0.3025 & 0.2176 & 0.4020 & 0.6967 & 0.9309 \\ 0.1101 & 0.4805 & 0.6168 & 0.2258 & 0.4847 \\ 0.8824 & 0.7969 & 0.1263 & 0.3669 & 0.1446 \end{matrix}$$

Матрица B:

$$B = \begin{matrix} 0.8964 & 0.8677 & 0.3371 & 0.2589 & 0.7297 \\ 0.8938 & 0.2524 & 0.0866 & 0.5550 & 0.0264 \\ 0.3090 & 0.2213 & 0.4054 & 0.6982 & 0.9345 \\ 0.1107 & 0.4870 & 0.6264 & 0.2323 & 0.4920 \\ 0.8828 & 0.8043 & 0.1297 & 0.3736 & 0.1516 \end{matrix}$$

Результаты:

- Левая часть: 0.0453
- Правая часть: 0.1755
- Неравенство выполняется? Да

Выводы:

- Во всех трех экспериментах неравенство выполняется, что подтверждает его корректность.
- Левая часть существенно меньше правой во всех экспериментах, что говорит о запасе выполнения неравенства.
- Результаты подтверждают теоретическую справедливость выражения для произвольных обратимых матриц.
- Проверка продемонстрировала применимость неравенства к численным расчетам, связанным с изменениями обратных матриц.
- Полученные данные свидетельствуют о возможности использования данного подхода для анализа устойчивости численных методов.