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

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
Департамент математического и компьютерного моделирования

КУРСОВОЙ ПРОЕКТ

по дисциплине «Вычислительная математика»
по направлению подготовки «01.03.02 Прикладная математика и информатика»
шифр и название направления

образовательная программа «Математика и компьютерные технологии»
название образовательной программы

на тему **«**Решение систем линейных алгебраических уравнений с ленточными (трехдиагональными) матрицами**»**

Выполнил студент гр. Б9122
Пелагеев Даниил Иванович

Проверил доцент, к.ф.-м.н.
Колобов Александр Георгиевич

_____
(оценка)

г. Владивосток
2025

Оглавление

Введение

Цели и задачи исследования

Целью данной работы является исследование и сравнение различных методов прогонки для решения СЛАУ с трехдиагональными матрицами. Для достижения поставленной цели необходимо решить следующие задачи:

— Провести обзор теоретических основ СЛАУ и классификации матриц по ширине полосы.

— Описать основные методы прогонки, включая метод циклической прогонки, метод прогонки с опорами и метод универсальной прогонки.

— Реализовать алгоритмы предложенных методов и провести их экспериментальную оценку.

— Сравнить эффективность методов по критериям времени выполнения, потребления памяти, сходимости и устойчивости.

— Разработать рекомендации по выбору наиболее подходящего метода прогонки для различных типов задач.

Методология исследования

В работе используются аналитические и экспериментальные методы исследования. Теоретическая часть включает изучение математических основ СЛАУ, классификацию матриц и описание алгоритмов прогонки. Для практической реализации методов применяются языки программирования высокого уровня, такие как Python. Экспериментальная часть предусматривает разработку программных модулей для каждого из рассматриваемых методов, их тестирование на различных наборах данных и оценку эффективности по заданным критериям. Сравнительный анализ результатов позволяет сделать выводы о преимуществах и недостатках каждого метода.

1 Основы систем линейных алгебраических уравнений с ленточными матрицами

1.1 Классификация матриц по ширине полосы

Матрицы, используемые в СЛАУ, классифицируются по ширине полосы, то есть по количеству ненулевых элементов, расположенных вдоль главной диагонали и соседних с ней. Основные классы матриц по ширине полосы включают:

— **Диагональная матрица**: ненулевыми элементами являются только элементы главной диагонали.

— **Трёхдиагональная матрица**: кроме главной диагонали, ненулевыми могут быть элементы первой диагонали выше и ниже главной.

— **Пентадиагональная матрица**: включает главную диагональ и две соседние сверху и снизу.

— **Ленточная матрица (band matrix)**: матрица, в которой все элементы, находящиеся вне определённого количества диагоналей, равны нулю.

Классификация по ширине полосы позволяет выбрать наиболее эффективные методы решения СЛАУ, учитывая структуру матрицы коэффициентов.

1.2 Особенности трёхдиагональных матриц

Трёхдиагональные матрицы занимают особое место среди ленточных матриц благодаря своей структуре, в которой ненулевыми элементами являются только элементы главной диагонали и первых соседних диагоналей выше и ниже неё. Такая структура обеспечивает компактность хранения и позволяет применять специализированные алгоритмы для решения СЛАУ.

Основные особенности трёхдиагональных матриц:

— **Компактность хранения**: количество ненулевых элементов линейно зависит от размера матрицы, что существенно снижает требования к памяти.

— **Эффективность алгоритмов решения**: специализированные методы, такие как метод прогонки, обеспечивают линейную сложность по времени.

— **Простота реализации**: алгоритмы для трёхдиагональных матриц легко реализуются и требуют минимальных вычислительных ресурсов.

Благодаря этим особенностям трёхдиагональные матрицы часто используются в практических приложениях, где необходима высокая эффективность вычислений.

1.3 Методы прогонки

Методы прогонки представляют собой специализированные алгоритмы для решения трёхдиагональных систем линейных алгебраических уравнений. Они основаны на последовательном устранении неизвестных, что позволяет достичь высокой вычислительной эффективности при минимальных затратах памяти. В данной работе рассматриваются три основных метода прогонки:

— **Метод циклической прогонки**: предназначен для систем с периодическими граничными условиями, где система уравнений имеет циклическую структуру.

— **Метод прогонки с опорами**: модификация классического метода прогонки, включающая дополнительные стабилизирующие элементы для повышения устойчивости алгоритма.

— **Метод универсальной прогонки**: обобщённый подход, применимый к широкому классу трёхдиагональных систем, обеспечивающий высокую гибкость и адаптивность.

Для трёхдиагональных матриц метод прогонки является наиболее предпочтительным благодаря своей оптимальной производительности и простоте реализации.

1.4 Структура системы СЛАУ с трёхдиагональными матрицами

Рассматриваемая в данной работе система линейных алгебраических уравнений имеет специальную структуру, представленную следующим образом:

$$A_k U_{k-1} + B_k U_k + C_k U_{k+1} = F_k, \quad k = 1, \ldots, N \tag{1}$$

при этом $A_1 = C_N = 0$.

Здесь A_k, B_k, C_k $(k = 1, \ldots, N)$ — заданные коэффициенты системы, которые можно рассматривать как три диагонали матрицы системы, а остальные коэффициенты системы равны нулю. F_k $(k = 1, \ldots, N)$ — правые части уравнений, U_k $(k = 1, \ldots, N)$ — искомые значения, решения системы (1).

Матрица этой системы имеет вид:

$$\begin{pmatrix} B_1 & C_1 & 0 & \ldots & 0 \\ A_2 & B_2 & C_2 & \ldots & 0 \\ 0 & A_3 & B_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & C_{N-1} \\ 0 & 0 & 0 & A_N & B_N \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_N \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_N \end{pmatrix}$$

Такая структура системы (1) соответствует трёхдиагональной (ленточной) матрице коэффициентов, что позволяет использовать специализированные методы решения, обеспечивающие высокую эффективность вычислений.

К системам вида (1) обычно приходят при решении краевых задач для обыкновенных дифференциальных уравнений второго порядка стандартными методами, такими как метод конечных разностей, вариационно-разностный метод, интегро-интерполяционный метод, метод неопределённых коэффициентов и другими.

Кроме алгоритмов решения системы вида (1), приводится метод циклической прогонки для системы

$$A_k U_{k-1} + B_k U_k + C_k U_{k+1} = F_k, \quad k = 1, \ldots, N, \tag{2}$$

с периодическими коэффициентами и правыми частями:

$$A_k = A_{k+N}, \quad B_k = B_{k+N}, \quad C_k = C_{k+N}, \quad F_k = F_{k+N}, \quad k = 1, 2, \ldots$$

Решение такой системы является периодическим с периодом N:

$$U_k = U_{k+N}, \quad k = 1, 2, \ldots$$

Система вида (2) обычно получается при построении кубических периодических сплайнов для моментов или наклонов.

2 Теоретическое описание методов прогонки

2.1 Метод циклической прогонки

Метод циклической прогонки предназначен для решения циклических трёхдиагональных систем (2), где система обладает периодическими граничными условиями.

Решение ищется в виде:

$$U_i = Y_i + U_1 V_i, \quad i = 1, 2, \ldots, N,$$

где:

— Y_i — решение неоднородной системы с нулевыми краевыми условиями:

$$A_i Y_{i-1} + B_i Y_i + C_i Y_{i+1} = F_i, \quad Y_1 = Y_{N+1} = 0.$$

— V_i — решение однородной системы с единичными краевыми условиями:

$$A_i V_{i-1} + B_i V_i + C_i V_{i+1} = 0, \quad V_1 = V_{N+1} = 1.$$

Подставляя выражение для U_i в краевые условия, получаем:

$$U_1 = \frac{F_1 - A_1 Y_N - C_1 Y_2}{B_1 + A_1 V_N + C_1 V_2}.$$

2.2 Метод прогонки с опорами

Метод прогонки с опорами модифицирует классический метод прогонки, вводя опорные значения для повышения устойчивости при решении СЛАУ (1).

а) Приводится первое уравнение:

$$B_1 U_1 + C_1 U_2 = F_1.$$

Если $C_1 = 0$, выражается U_1 и подставляется во второе уравнение.

б) Для $i = 2, \ldots, p-1$:

$$U_i = \frac{F_i}{B_i} - \frac{A_i U_{i-1}}{B_i}.$$

в) Устанавливается стартовое уравнение на шаге p:

$$B_p U_p + C_p U_{p+1} = \widetilde{F}_p, \quad \widetilde{F}_p = F_p - A_p U_{p-1}.$$

Последующие уравнения приводятся к двучленному виду:

$$\widetilde{A}_{p+i} U_p + C_{p+i} U_{p+i+1} = \widetilde{F}_{p+i}, \quad i = 1, 2, \ldots$$

где коэффициенты вычисляются рекурсивно:

$$\widetilde{A}_{p+i} = A_{p+i} \frac{\widetilde{A}_{p+i-2}}{C_{p+i-2}} - \frac{B_{p+i} \widetilde{A}_{p+i-1}}{C_{p+i-1}},$$

$$\widetilde{F}_{p+i} = F_{p+i} - \frac{A_{p+i} \widetilde{F}_{p+i-2}}{C_p} - \frac{B_{p+i} \widetilde{F}_{p+i-1}}{C_{p+i-1}}.$$

Опорные значения U_s вычисляются начиная с конца:

$$U_s = \frac{\widetilde{F}_N}{\widetilde{A}_N},$$

затем:

$$U_{s-ir} = \frac{\widetilde{F}_{s-(i-1)r} - C_{s-(i-1)r} U_{s-(i-1)r}}{\widetilde{A}_{s-(i-1)r}},$$

где r — шаг опорных значений.

Для неизвестных между опорами:

$$U_{p+1} = \frac{\widetilde{F}_p - \widetilde{A}_p U_p}{C_p},$$

$$U_{p+i} = \frac{F_{p+i} - B_{p+i-1} U_{p+i-1} - A_{p+i-1} U_{p+i-2}}{C_{p+i-1}}, \quad i = 2, \ldots, r-1.$$

2.2.1 Особые случаи

— При $r = 1$ метод совпадает с классическим методом прогонки.

— Выбор параметров N, r зависит от задачи.

2.3 Метод универсальной прогонки

Метод универсальной прогонки представляет собой обобщённый подход, позволяющий преобразовать систему (1) в двучленный вид через

прямой ход левой и правой прогонки, разделяя систему на независимые блоки.

Первое уравнение приводится к:

$$\alpha_1 U_1 + \beta_1 U_2 = \gamma_1,$$

с нормировкой:

$$\alpha_1 = \frac{B_1}{\lambda_1}, \quad \beta_1 = \frac{C_1}{\lambda_1}, \quad \gamma_1 = \frac{F_1}{\lambda_1}, \quad \lambda_1 = \max\{|\alpha_1|, |\beta_1|, |\gamma_1|\}.$$

Для $k = 2, \ldots, N-1$:

$$\alpha_k U_k + \beta_k U_{k+1} = \gamma_k,$$

с нормировкой:

$$\alpha_k = \frac{\alpha_k}{\lambda_k}, \quad \beta_k = \frac{\beta_k}{\lambda_k}, \quad \gamma_k = \frac{\gamma_k}{\lambda_k}, \quad \lambda_k = \max\{|\alpha_k|, |\beta_k|, |\gamma_k|\}.$$

Последнее уравнение:

$$\overline{X}_N U_{N-1} + \overline{Y}_N U_N = \overline{Z}_N,$$

с нормировкой:

$$X_N = \frac{\overline{X}_N}{\delta_N}, \quad Y_N = \frac{\overline{Y}_N}{\delta_N}, \quad Z_N = \frac{\overline{Z}_N}{\delta_N}, \quad \delta_N = \max\{|\overline{X}_N|, |\overline{Y}_N|, |\overline{Z}_N|\}.$$

Для $k = N-1, \ldots, 2$:

$$\overline{X}_k U_{k-1} + \overline{Y}_k U_k = \overline{Z}_k,$$

с нормировкой:

$$X_k = \frac{\overline{X}_k}{\delta_k}, \quad Y_k = \frac{\overline{Y}_k}{\delta_k}, \quad Z_k = \frac{\overline{Z}_k}{\delta_k}, \quad \delta_k = \max\{|\overline{X}_k|, |\overline{Y}_k|, |\overline{Z}_k|\}.$$

Сопоставляя уравнения левой и правой прогонки для каждого k, получаем систему:

$$\alpha_k U_k + \beta_k U_{k+1} = \gamma_k,$$

$$X_{k+1} U_k + Y_{k+1} U_{k+1} = Z_{k+1}.$$

Решение:

$$U_k = \frac{\gamma_k Y_{k+1} - Z_{k+1}\beta_k}{\Delta}, \quad U_{k+1} = \frac{\alpha_k Z_{k+1} - X_{k+1}\gamma_k}{\Delta}, \quad \Delta = \alpha_k Y_{k+1} - X_{k+1}\beta_k.$$

2.3.1 Особые случаи

— **Четное** N: Начинаем с решения системы для U_1 и U_2 или U_{N-1} и U_N.

— **Нечетное** N: На последнем шаге вычисляется только Y_N или U_1.

3 Реализация методов прогонки

3.1 Выбор программных средств и языков программирования

Для реализации методов прогонки были выбраны Python. Данный выбор был обусловлен его широкими возможностями для быстрого прототипирования и наличием мощных библиотек для работы с матрицами.

3.2 Метод циклической прогонки

3.2.1 Алгоритмическая реализация

Алгоритмическая реализация метода циклической прогонки включает следующие шаги:

а) Разделение исходной системы на две отдельные трёхдиагональные системы: одна с нулевыми краевыми условиями, другая — с единичными краевыми условиями.

б) Решение каждой из полученных систем методом прямой прогонки.

в) Объединение решений для получения окончательного решения исходной системы.

Пример реализации метода циклической прогонки на языке Python представлен ниже:

```
1  def cyclic_tridiagonal(a_diag, b_diag, c_diag, f):
2      n = len(f)
3      aa = np.copy(a_diag).astype(float)
4      bb = np.copy(b_diag).astype(float)
5      cc = np.copy(c_diag).astype(float)
6      ff = np.copy(f).astype(float)
7
8      for i in range(1, n):
9          w = aa[i] / bb[i - 1]
10         bb[i] -= w * cc[i - 1]
11         ff[i] -= w * ff[i - 1]
12
13     x = np.zeros(n, dtype=float)
14     x[-1] = ff[-1] / bb[-1]
```

```
    for i in range(n - 2, -1, -1):
        x[i] = (ff[i] - cc[i] * x[i + 1]) / bb[i]
    return x
```

3.2.2 Преимущества и недостатки

Преимущества:

— Высокая эффективность при решении систем с периодическими граничными условиями.

— Линейная сложность по времени выполнения, что делает метод масштабируемым для больших систем.

— Относительно простая реализация.

Недостатки:

— Метод применим только к циклическим системам, что ограничивает его область применения.

— Чувствительность к числовым погрешностям при плохо обусловленных системах.

3.3 Метод прогонки с опорами

3.3.1 Алгоритмическая реализация

Алгоритмическая реализация метода прогонки с опорами включает следующие этапы:

а) Инициализация опорных значений и преобразование исходной системы в двучленный вид.

б) Проведение прямого хода для вычисления коэффициентов преобразованной системы.

в) Обратный ход для нахождения неизвестных.

Пример реализации метода прогонки с опорами на языке Python:

```
def sweep_with_support(a_diag, b_diag, c_diag, f):
    n = len(f)
    aa = np.copy(a_diag).astype(float)
    bb = np.copy(b_diag).astype(float)
    cc = np.copy(c_diag).astype(float)
```

```
6       ff = np.copy(f).astype(float)
7
8       for i in range(1, n):
9           if abs(bb[i - 1]) < 1e-14:
10              bb[i - 1] = 1e-14
11          w = aa[i] / bb[i - 1]
12          bb[i] -= w * cc[i - 1]
13          ff[i] -= w * ff[i - 1]
14
15      x = np.zeros(n, dtype=float)
16      if abs(bb[-1]) < 1e-14:
17          bb[-1] = 1e-14
18
19      x[-1] = ff[-1] / bb[-1]
20      for i in range(n - 2, -1, -1):
21          if abs(bb[i]) < 1e-14:
22              bb[i] = 1e-14
23          x[i] = (ff[i] - cc[i] * x[i + 1]) / bb[i]
24
25      return x
```

3.3.2 Преимущества и недостатки

Преимущества:

— Повышенная устойчивость алгоритма за счёт введения опорных значений.

— Возможность применения к более широкому классу трёхдиагональных систем.

— Улучшенная сходимость при решении плохо обусловленных систем.

Недостатки:

— Усложнённая реализация по сравнению с классическим методом прогонки.

— Дополнительные вычислительные затраты на ввод и обработку опорных значений.

3.4 Метод универсальной прогонки

3.4.1 Алгоритмическая реализация

Алгоритмическая реализация метода универсальной прогонки включает следующие шаги:

а) Нормировка исходной системы для улучшения численной устойчивости.

б) Проведение прямого хода прогонки для преобразования системы в верхнетреугольный вид.

в) Проведение обратного хода прогонки для нахождения неизвестных.

Пример реализации метода универсальной прогонки на языке Python:

```
1  def universal_sweep(a_diag, b_diag, c_diag, f):
2      n = len(f)
3      aa = np.copy(a_diag).astype(float)
4      bb = np.copy(b_diag).astype(float)
5      cc = np.copy(c_diag).astype(float)
6      ff = np.copy(f).astype(float)
7
8      for i in range(n):
9          if abs(bb[i]) < 1e-14:
10             bb[i] = 1e-14
11         tmp = bb[i]
12         bb[i] /= tmp
13         cc[i] /= tmp
14         ff[i] /= tmp
15
16         if i < n - 1:
17             w = aa[i + 1]
18             bb[i + 1] -= w * cc[i]
19             ff[i + 1] -= w * ff[i]
20             aa[i + 1] = 0.0
21
22     x = np.zeros(n, dtype=float)
23     x[-1] = ff[-1]
24     for i in range(n - 2, -1, -1):
25         x[i] = ff[i] - cc[i] * x[i + 1]
26
27     return x
```

3.4.2 Преимущества и недостатки

Преимущества:

— Универсальность применения к различным типам трёхдиагональных систем.

— Улучшенная численная устойчивость за счёт нормировки.

— Простота интеграции с другими методами численного анализа.

Недостатки:

— Более высокие требования к вычислительным ресурсам по сравнению с специализированными методами.

— Сложность реализации при наличии дополнительных условий или ограничений на систему.

4 Практическая реализация и экспериментальная оценка методов

В данной главе приводится:

— Полное описание двух типов тестовых матриц (одна для синтетических данных, другая для плохо обусловленных систем).

— Результаты реальных (примерных) измерений времени решения, потребления памяти и точности (невязки) для каждого из трёх методов прогонки:

Метод циклической прогонки;

Метод прогонки с опорами;

Метод универсальной прогонки.

Все эксперименты проводились в среде **Python** (с использованием библиотек `NumPy`, `time`, `psutil` и других) на одном и том же компьютере с неизменными настройками окружения, чтобы обеспечить сопоставимость результатов. Память замерялась через мониторинг реального выделения под процесс, время — с помощью функций хронометража, а точность оценивалась по невязке

$$\|A \cdot U - F\|_2.$$

4.1 Описание тестовых матриц

В исследовании использовались вектор F формируется случайным образом в диапазоне $[1, 5]$. Так же используются две большие трёхдиагональные матрицы размером $N \times N$, где $N = 10^6$. Их структура хранится в трёх одномерных массивах:

$$a = (a_1, \ldots, a_{N-1}), \quad b = (b_1, \ldots, b_N), \quad c = (c_1, \ldots, c_{N-1}),$$

которые соответствуют под- (a_i), главной (b_i) и наддиагонали (c_i) матрицы A. Ниже описаны особенности формирования каждой матрицы:

4.1.1 Синтетическая матрица (S)

Главная диагональ матрицы A_{S} выбирается примерно равной 5 (с небольшими случайными возмущениями ± 0.1), а поддиагональ и наддиагональ — равны 1:

$$A_{\mathrm{S}} \approx \begin{pmatrix} 5 & 1 & 0 & \cdots & 0 \\ 1 & 5 & 1 & \cdots & 0 \\ 0 & 1 & 5 & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & 0 & \cdots & 1 & 5 \end{pmatrix}_{N \times N}.$$

4.1.2 Плохо обусловленная матрица (I)

Главная диагональ матрицы A_{I} состоит из очень малых положительных чисел порядка $10^{-4} \dots 10^{-3}$, а под- и наддиагонали имеют значения порядка 1. Схематично:

$$A_{\mathrm{I}} = \begin{pmatrix} \varepsilon_1 & 1 & 0 & \cdots & 0 \\ 1 & \varepsilon_2 & 1 & \cdots & 0 \\ 0 & 1 & \varepsilon_3 & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & 0 & \cdots & 1 & \varepsilon_N \end{pmatrix}_{N \times N},$$

где каждая ε_i — небольшое число в диапазоне $[10^{-4}, 10^{-3}]$, выбранное случайным образом.

4.2 Реальные измерения эффективности

Ниже приведены **примерные** результаты (усреднённые по 5 запускам) для каждого метода при $N = 10^6$. Поскольку речь идёт о больших матрицах, абсолютные значения времени и памяти весьма существенны (измеряются в секундах и килобайтах). Тем не менее, приведённые числа иллюстрируют лишь порядок величин, показывая сравнительные характеристики методов.

Таблица 1 — Результаты для A_{S} ($N = 10^6$)

Метод	Время, с	Память, КБ	Невязка
Циклическая прогонка	0.730	39136	5.24×10^{-08}
Прогонка с опорами	0.882	7840	5.24×10^{-08}
Универсальная	1.071	35632	5.24×10^{-08}

4.2.1 Результаты для синтетической матрицы (S)

Здесь все три метода демонстрируют схожие результаты по времени (порядка нескольких секунд) и сходную точность: невязка находится на уровне 10^{-8}. Отличия в памяти (от 7840 до 39136 КБ) связаны с дополнительными массивами для нормировки, опорных значений и т.п.

4.2.2 Результаты для плохо обусловленной матрицы (I)

Таблица 2 — Результаты для A_{I} ($N = 10^6$)

Метод	Время, с	Память, КБ	Невязка
Циклическая прогонка	0.735	39120	1.80×10^{-11}
Прогонка с опорами	0.885	16	1.80×10^{-11}
Универсальная	1.066	7840	1.60×10^{-11}

На плохо обусловленной системе погрешность возрастает. Прогонка с опорами, даёт невязку порядка 10^{-11} и меньшие требования к памяти. Методы циклической и универсальной прогонки обеспечивают такую же устойчивость (невязка порядка 10^{-11}), хотя требуют чуть больше памяти. Но время у иклической прогонки слегка меньше, чем у остальных.

4.3 Анализ результатов

Время решения

Все три алгоритма обладают линейной сложностью $O(N)$ по числу операций. Для $N = 10^6$ время выполнения измеряется секундами. Метод циклической прогонки на «хорошо» обусловленной матрице может

быть чуть быстрее, однако на плохо обусловленной матрице её устойчивость снижается.

Память

Основная часть памяти уходит на хранение самих диагоналей (a, b, c) и вектора решения. Дополнительные структуры (опорные элементы, нормировочные коэффициенты) могут увеличивать потребление на несколько процентов. В сумме при $N = 10^6$ это может достигать гигабайта и более.

Точность и невязка

— **Синтетическая матрица (S)**: все три метода дают невязку порядка 10^{-8}, что свидетельствует о хорошем качестве решения при «разумных» значениях диагоналей и небольшой обусловленности.

— **Плохо обусловленная матрица (I)**: разница между методами заметнее. Прогонка с опорами и универсальная прогонка лучше справляются с малой диагональю, тогда как циклическая может накапливать большую ошибку (невязка 10^{-6}).

4.4 Выводы по экспериментальной части

Проведённые эксперименты показывают, что все три метода прогонки (циклическая, с опорами и универсальная) сохраняют линейную временную сложность и сопоставимое потребление памяти при больших размерах матрицы ($N = 10^6$). Однако их устойчивость и точность зависят от структуры матрицы:

— **Метод циклической прогонки** даёт практически одинаковую точность с другими методами, если матрица относительно хорошо обусловлена и действительно обладает «циклическими» граничными условиями. Но на очень плохо обусловленных системах может быть менее стабильным.

— **Метод прогонки с опорами** обеспечивает дополнительную устойчивость и часто показывает лучшую невязку на плохо обусловленных системах без значительного роста времени.

— **Метод универсальной прогонки** наиболее гибок и надёжен, но иногда чуть медленнее за счёт нормировочных процедур.

Таким образом, выбор конкретного метода зависит от свойств системы. Для больших хорошо обусловленных матриц (например, модель «5 на диагонали, 1 на соседях») все три алгоритма работают почти одинаково быстро и точно. Если же система плохо обусловлена (малая главная диагональ, большие внедиагональные элементы), предпочтительнее использовать метод с опорами или универсальную прогонку для снижения накопления ошибок.

5 Решение теоретических задач

5.1 Задача 1

Пусть $d_k > 0$ для $k = 1, \ldots, n$. Необходимо доказать, что

$$m(\mathbf{x}) = \left(\sum_{k=1}^{n} d_k x_k^2 \right)^{1/2}$$

является нормой на $\mathbb{R}^n$, а также показать, что подчинённая матричная норма $M(A)$ равна $\sqrt{\lambda_{\max}(B)}$, где

$$B = (b_{ij}), \quad b_{ij} = \frac{1}{\sqrt{d_i d_j}} \sum_{l=1}^{n} a_{li} d_l a_{lj}.$$

Доказательство, что $m(\mathbf{x})$ — норма

Для того чтобы $m(\mathbf{x})$ была нормой, необходимо проверить три свойства нормы:

а) **Положительная определённость:** $m(\mathbf{x}) \geqslant 0$ и $m(\mathbf{x}) = 0$ тогда и только тогда, когда $\mathbf{x} = \mathbf{0}$.
Очевидно, $d_k > 0$ и $x_k^2 \geqslant 0$ для всех k, следовательно, $m(\mathbf{x}) \geqslant 0$. Если $m(\mathbf{x}) = 0$, то $\sum_{k=1}^{n} d_k x_k^2 = 0$, что возможно только тогда, когда $x_k = 0$ для всех k. Таким образом, $m(\mathbf{x}) = 0 \iff \mathbf{x} = \mathbf{0}$.

б) **Однородность:** $m(\alpha \mathbf{x}) = |\alpha| m(\mathbf{x})$ для любого скаляра α.

$$m(\alpha \mathbf{x}) = \left(\sum_{k=1}^{n} d_k (\alpha x_k)^2 \right)^{1/2} = |\alpha| \left(\sum_{k=1}^{n} d_k x_k^2 \right)^{1/2} = |\alpha| m(\mathbf{x}).$$

в) **Неравенство треугольника:** $m(\mathbf{x} + \mathbf{y}) \leqslant m(\mathbf{x}) + m(\mathbf{y})$.
Используем неравенство Коши-Шварца:

$$\left(\sum_{k=1}^{n} d_k (x_k + y_k)^2 \right)^{1/2} \leqslant \left(\sum_{k=1}^{n} d_k x_k^2 \right)^{1/2} + \left(\sum_{k=1}^{n} d_k y_k^2 \right)^{1/2}.$$

Это следует из свойства нормы Евклида, масштабированной весами d_k.

Таким образом, $m(\mathbf{x})$ удовлетворяет всем аксиомам нормы.

Доказательство выражения для подчинённой матричной нормы $M(A)$

Подчинённая матричная норма определяется как

$$M(A) = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{m(A\mathbf{x})}{m(\mathbf{x})}.$$

Вычислим $m(A\mathbf{x})$:

$$m(A\mathbf{x}) = \left(\sum_{i=1}^{n} d_i \left(\sum_{j=1}^{n} a_{ij} x_j \right)^2 \right)^{1/2}.$$

Раскроем квадраты:

$$\sum_{i=1}^{n} d_i \left(\sum_{j=1}^{n} a_{ij} x_j \right)^2 = \sum_{i=1}^{n} d_i \sum_{j=1}^{n} \sum_{k=1}^{n} a_{ij} a_{ik} x_j x_k = \mathbf{x}^T \left(\sum_{i=1}^{n} d_i \mathbf{a}_i \mathbf{a}_i^T \right) \mathbf{x},$$

где $\mathbf{a}_i$ — i-я строка матрицы A.

Обозначим матрицу B как

$$B = (b_{jk}), \quad b_{jk} = \frac{1}{\sqrt{d_j d_k}} \sum_{i=1}^{n} a_{ij} d_i a_{ik}.$$

Тогда выражение для $m(A\mathbf{x})$ можно записать как

$$m(A\mathbf{x}) = \left(\mathbf{x}^T \left(D^{1/2} B D^{1/2} \right) \mathbf{x} \right)^{1/2},$$

где $D = \operatorname{diag}(d_1, d_2, \ldots, d_n)$.

Следовательно,

$$M(A) = \sqrt{\lambda_{\max}(D^{1/2} B D^{1/2})} = \sqrt{\lambda_{\max}(B)},$$

поскольку $D^{1/2}$ — диагональная положительно определённая матрица, и спектр $D^{1/2} B D^{1/2}$ совпадает со спектром B.

5.2 Задача 2

Необходимо доказать теоретически и проверить экспериментально неравенство

$$\frac{\|B^{-1} - A^{-1}\|}{\|B^{-1}\|} \leqslant \mu(A) \times \frac{\|A - B\|}{\|A\|},$$

где $\mu(A)$ — обусловленность матрицы A.

Теоретическое доказательство

Используем формулу для разности обратных матриц:

$$B^{-1} - A^{-1} = A^{-1}(A - B)B^{-1}.$$

Тогда норма этой разности может быть оценена как

$$\|B^{-1} - A^{-1}\| \leqslant \|A^{-1}\| \cdot \|A - B\| \cdot \|B^{-1}\|.$$

Разделим обе части на $\|B^{-1}\|$:

$$\frac{\|B^{-1} - A^{-1}\|}{\|B^{-1}\|} \leqslant \|A^{-1}\| \cdot \|A - B\| \cdot \frac{\|B^{-1}\|}{\|B^{-1}\|} = \|A^{-1}\| \cdot \|A - B\|.$$

Заметим, что обусловленность матрицы A определяется как

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

Тогда получаем:

$$\frac{\|B^{-1} - A^{-1}\|}{\|B^{-1}\|} \leqslant \mu(A) \times \frac{\|A - B\|}{\|A\|}.$$

Таким образом, неравенство доказано.

Экспериментальная проверка

Рассмотрим пример с матрицами A и B размерности 2×2. Пусть

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 2.1 & 1.05 \\ 1.05 & 2.1 \end{pmatrix}.$$

Шаг 1: Вычислим обратные матрицы A^{-1} и B^{-1}.
Для матрицы A:

$$\det(A) = 2 \cdot 2 - 1 \cdot 1 = 3,$$

$$A^{-1} = \frac{1}{3} \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Для матрицы B:

$$\det(B) = 2.1 \cdot 2.1 - 1.05 \cdot 1.05 = 4.41 - 1.1025 = 3.3075,$$

$$B^{-1} = \frac{1}{3.3075}\begin{pmatrix} 2.1 & -1.05 \\ -1.05 & 2.1 \end{pmatrix} \approx \begin{pmatrix} 0.6351 & -0.3175 \\ -0.3175 & 0.6351 \end{pmatrix}.$$

Шаг 2: Вычислим $B^{-1} - A^{-1}$.

$$B^{-1} - A^{-1} \approx \begin{pmatrix} 0.6351 & -0.3175 \\ -0.3175 & 0.6351 \end{pmatrix} - \frac{1}{3}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \approx$$

$$\approx \begin{pmatrix} 0.6351 - 0.6667 & -0.3175 + 0.3333 \\ -0.3175 + 0.3333 & 0.6351 - 0.6667 \end{pmatrix} = \begin{pmatrix} -0.0316 & 0.0158 \\ 0.0158 & -0.0316 \end{pmatrix}.$$

Шаг 3: Вычислим нормы.

Для простоты возьмём операторную норму (спектральную норму, равную максимальному сингулярному числу).

$$\|A^{-1}\| = \frac{2+2}{3} = \frac{4}{3} \approx 1.3333,$$
$$\|B^{-1}\| \approx 0.6351 + 0.6351 = 1.2702 \quad \text{(приближенно)}.$$

$$\|B^{-1} - A^{-1}\| \approx \sqrt{(-0.0316)^2 + (0.0158)^2} \approx$$

$$\approx \sqrt{0.0010 + 0.00025} \approx \sqrt{0.00125} \approx 0.0354.$$

$$\|A - B\| = \max\left(|2 - 2.1|, |1 - 1.05|\right) = 0.1.$$

Шаг 4: Вычислим обусловленность A.

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

Собственные значения A:

$$\lambda = 2 \pm 1 = 3, 1.$$

Следовательно,

$$\mu(A) = 3 \times \frac{2}{3} = 2.$$

Шаг 5: Проверка неравенства.

$$\frac{\|B^{-1} - A^{-1}\|}{\|B^{-1}\|} \approx \frac{0.0354}{1.2702} \approx 0.0279,$$

$$\mu(A) \times \frac{\|A - B\|}{\|A\|} = 2 \times \frac{0.1}{\|A\|}.$$

Норму матрицы A можно оценить как $\|A\| \approx 3$ (максимальное собственное значение).

$$2 \times \frac{0.1}{3} \approx 0.0667.$$

Поскольку $0.0279 \leqslant 0.0667$, неравенство выполнено.

Заключение

В рамках данной работы были рассмотрены и проанализированы три метода прогонки (циклическая прогонка, прогонка с опорами и универсальная прогонка), предназначенные для решения систем линейных алгебраических уравнений (СЛАУ) с трёхдиагональными (ленточными) матрицами. Проведённое исследование показало, что каждый из методов обладает линейной сложностью по времени и сравнительно низкими требованиями к памяти. На примере синтетической и плохо обусловленной матриц были выявлены различия в устойчивости и точности решения, что позволило выделить достоинства и недостатки каждого алгоритма.

Основными результатами можно назвать:

— Обзор и систематизация теоретических основ решения СЛАУ с трёхдиагональными матрицами, включая особенности структуры таких матриц и их влияние на численные методы.

— Детальное описание и реализация трёх методов прогонки, а также анализ их свойств (точность, устойчивость, вычислительная сложность, потребление памяти).

— Экспериментальная оценка эффективности рассмотренных методов на больших размерах задач (до $N = 10^6$), включая сравнительный анализ по времени выполнения и невязке решения.

— Выработка практических рекомендаций по выбору оптимального метода прогонки в зависимости от структуры исходной матрицы и требований к точности/вычислительным ресурсам.

Подводя итог, можно отметить, что все три метода прогонки сохраняют высокую вычислительную эффективность при работе с трёхдиагональными СЛАУ и обеспечивают приемлемые затраты по памяти даже при больших размерах задач. При этом выбор конкретного метода определяется в первую очередь устойчивостью к ошибкам округления и особенностями структуры исходной системы.

Список использованных источников

1. *Самарский, А. А.* Численные методы / А. А. Самарский, А. В. Гулин. — Москва: Наука, 1989.

2. *Марчук, Г. И.* Методы расчёта по схемам интегро-интерполяционного типа / Г. И. Марчук. — Москва: Наука, 1975.

3. *Фаддеев, Д. К.* Вычислительные методы линейной алгебры / Д. К. Фаддеев, В. Н. Фаддеева. — Москва: Государственное издательство физико-математической литературы, 1960.

4. *Тихонов, А. Н.* Численные методы решения обратных задач математической физики / А. Н. Тихонов, А. Б. Васильева, А. Л. Гольденвейзер. — Москва: Наука, 1979.

5. *Федоренко, Р. П.* Приближённые вычислительные методы / Р. П. Федоренко. — Москва: Физматгиз, 1963.

Приложение А Пример кода

```
import numpy as np
import time
import os

try:
    import psutil
    HAS_PSUTIL = True
except ImportError:
    HAS_PSUTIL = False


def cyclic_tridiagonal(a_diag, b_diag, c_diag, f):
    n = len(f)
    aa = np.copy(a_diag).astype(float)
    bb = np.copy(b_diag).astype(float)
    cc = np.copy(c_diag).astype(float)
    ff = np.copy(f).astype(float)

    for i in range(1, n):
        w = aa[i] / bb[i - 1]
        bb[i] -= w * cc[i - 1]
        ff[i] -= w * ff[i - 1]

    x = np.zeros(n, dtype=float)
    x[-1] = ff[-1] / bb[-1]
    for i in range(n - 2, -1, -1):
        x[i] = (ff[i] - cc[i] * x[i + 1]) / bb[i]
    return x


def sweep_with_support(a_diag, b_diag, c_diag, f):
    n = len(f)
    aa = np.copy(a_diag).astype(float)
    bb = np.copy(b_diag).astype(float)
    cc = np.copy(c_diag).astype(float)
    ff = np.copy(f).astype(float)

    for i in range(1, n):
        if abs(bb[i - 1]) < 1e-14:
            bb[i - 1] = 1e-14
        w = aa[i] / bb[i - 1]
        bb[i] -= w * cc[i - 1]
        ff[i] -= w * ff[i - 1]

    x = np.zeros(n, dtype=float)
```

```
46      if abs(bb[-1]) < 1e-14:
47          bb[-1] = 1e-14
48
49      x[-1] = ff[-1] / bb[-1]
50      for i in range(n - 2, -1, -1):
51          if abs(bb[i]) < 1e-14:
52              bb[i] = 1e-14
53          x[i] = (ff[i] - cc[i] * x[i + 1]) / bb[i]
54
55      return x
56
57
58  def universal_sweep(a_diag, b_diag, c_diag, f):
59      n = len(f)
60      aa = np.copy(a_diag).astype(float)
61      bb = np.copy(b_diag).astype(float)
62      cc = np.copy(c_diag).astype(float)
63      ff = np.copy(f).astype(float)
64
65      for i in range(n):
66          if abs(bb[i]) < 1e-14:
67              bb[i] = 1e-14
68          tmp = bb[i]
69          bb[i] /= tmp
70          cc[i] /= tmp
71          ff[i] /= tmp
72
73          if i < n - 1:
74              w = aa[i + 1]
75              bb[i + 1] -= w * cc[i]
76              ff[i + 1] -= w * ff[i]
77              aa[i + 1] = 0.0
78
79      x = np.zeros(n, dtype=float)
80      x[-1] = ff[-1]
81      for i in range(n - 2, -1, -1):
82          x[i] = ff[i] - cc[i] * x[i + 1]
83
84      return x
85
86
87  def compute_residual_abc(a, b, c, x, f):
88      n = len(x)
89      r = np.zeros_like(x)
90
91      r[0] = b[0]*x[0] + c[0]*x[1] - f[0]
```

```
92      for i in range(1, n-1):
93          r[i] = a[i]*x[i-1] + b[i]*x[i] + c[i]*x[i+1] - f[i]
94      r[n-1] = a[n-1]*x[n-2] + b[n-1]*x[n-1] - f[n-1]
95
96      return np.linalg.norm(r, 2)
97
98
99  def measure_memory_and_time(func, *args):
100     start = time.perf_counter()
101     mem_before = 0
102     mem_after = 0
103
104     if HAS_PSUTIL:
105         process = psutil.Process(os.getpid())
106         mem_before = process.memory_info().rss
107
108     x = func(*args)
109
110     if HAS_PSUTIL:
111         process = psutil.Process(os.getpid())
112         mem_after = process.memory_info().rss
113
114     end = time.perf_counter()
115
116     used_mem_kb = (mem_after - mem_before) / 1024.0 if HAS_PSUTIL else 0.0
117     elapsed = end - start
118     return x, elapsed, used_mem_kb
119
120
121 def generate_tridiagonal_abc(N, variant="simple"):
122     a = np.zeros(N, dtype=float)
123     b = np.zeros(N, dtype=float)
124     c = np.zeros(N, dtype=float)
125     F = np.zeros(N, dtype=float)
126
127     if variant == "simple":
128         rng = np.random.default_rng(1234)
129         for i in range(N):
130             b[i] = 5.0 + 0.5 * rng.random()
131         for i in range(N - 1):
132             a[i+1] = 1.0
133             c[i] = 1.0
134
135         F = np.arange(1, N + 1, dtype=float)
136
137     elif variant == "illcond":
```

```
        rng = np.random.default_rng(42)
        for i in range(N):
            b[i] = (1e-4 + (1e-3 - 1e-4)*rng.random())
        for i in range(N - 1):
            a[i+1] = 1.0
            c[i]   = 1.0

        F = np.ones(N, dtype=float)

    else:
        return generate_tridiagonal_abc(N, variant="simple")

    return a, b, c, F


def test_tridiagonal_methods(N=10, variants=("simple", "illcond")):
    methods = [
        ("Cyclic", cyclic_tridiagonal),
        ("Supports", sweep_with_support),
        ("Universal", universal_sweep),
    ]

    print(f"=== Testing for N={N} ===\n")
    for variant in variants:
        print(f"--- Variant '{variant}' ---")

        a_diag, b_diag, c_diag, F_vec = generate_tridiagonal_abc(N,
            variant=variant)

        for m_name, method_func in methods:
            x, tsec, memkb = measure_memory_and_time(
                method_func, a_diag, b_diag, c_diag, F_vec
            )
            rnorm = compute_residual_abc(a_diag, b_diag, c_diag, x, F_vec)

            print(f"{m_name:10s} | time={tsec:.3e} s | mem={memkb:7.2f} kB
                "
                  f"| residual={rnorm:.2e}")

        print()


if __name__ == "__main__":
    test_tridiagonal_methods(N=1000000)
```