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»

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

Направление подготовки
01.03.02 «Прикладная математика и информатика»

Выполнил студент гр. Б9120-01.03.02систпро

_______________ Беляков О.В.
(*подпись*) *(Ф.И.О.)*

Проверил доцент, к.ф.-м.н.

Колобов А.Г.__________________
(подпись)

«_____»______________2024г.

г. Владивосток
2024

Содержание

1. Введение

Объектом исследования являются численные методы решения задач линейной алгебры, а также программное обеспечение, реализующее эти методы.

Цель работы – ознакомиться с численными методами решения систем линейных алгебраических уравнений, нахождения обратных матриц, решения проблемы собственных значений, решить предложенные типовые задачи, сформулировать выводы по полученным решениям, отметить достоинства и недостатки методов, сравнить удобство использования и эффективность работы каждой использованной программы, приобрести практические навыки и компетенции, а также опыт самостоятельной профессиональной деятельности, а именно:

1. создать алгоритм решения поставленной задачи и реализовать его, протестировать программы;

2. освоить теорию вычислительного эксперимента; современных компьютерных технологий;

3. приобрести навыки представления итогов проделанной работы в виде отчета, оформленного в соответствии с имеющимися требованиями, с привлечением современных средств редактирования и печати.

Работа над курсовым проектом предполагает выполнение следующих задач:

1. дальнейшее углубление теоретических знаний обучающихся и их систематизацию;

2. получение и развитие прикладных умений и практических навыков по направлению подготовки;

3. овладение методикой решения конкретных задач;

4. развитие навыков самостоятельной работы;
5. развитие навыков обработки полученных результатов, анализа и осмысления их с учетом имеющихся литературных данных;
6. приобретение навыков оформления описаний программного продукта;
7. повышение общей и профессиональной эрудиции.

2. Основная часть

2.1. Постановка задачи

Решение системы линейных алгебраических уравнений специального вида:

$$A_kU_{k-1} + B_kU_k + C_kU_{k+1} = F_k, k = 1, \ldots, N, A_1 = C_N = 0 \ (2.1)$$

$A_k, B_k, C_k, k = 1, \ldots, N,$ - 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, $U_k, k = 1, \ldots, N$-искомые значения, решения данной системы. К системам вида (2.1) обычно приходят при решении одномерной краевой задачи для дифференциального уравнения второго порядка, одномерной задачи нестационарной тепловодности или диффузии, многомерных параболических уравнений, локально-нелинейных одномерных задача и т.д.

2.2. Метод монотонной прогонки

Данный метод пригоден только для систем с коэффициентами, удовлетворяющими следующим условиям:

1. $|A_k| + |C_k| \leq |B_k|, k = 1, ...N$
2. Ненулевые элементы на главной диагонали

Алгоритм состоит из двух шагов: прямой и обратных ход:

2.2.1. Прямой ход

Прямым ходом называется вычисление прогоночных коэффициентов

1. Выразим U_1 из 2.1

$$U_1 = -\frac{C_1}{B_1} U_2 + \frac{F_1}{B_1}$$

2. Пусть

$$\alpha_1 = -\frac{C_1}{B_1}, \beta_1 = \frac{F_1}{B_1}$$

3. Тогда соотношение из П.1 изменится следующим образом:

$$U_1 = \alpha_2 U_2 + \beta_2$$

4. Предположим, что $U_{k-1} = \alpha_k U_k + \beta_k$
5. Уберём из k-го уравнения системы 2.1 U_{k-1}, подставив соотношение из П.4 в k-е уравнение

$$A_k(\alpha_k U_k + \beta_k) + B_k U_k + C_k U_{k+1} = F_k$$

Отсюда выразим U_k:

$$U_k = -\frac{C_k U_{k+1}}{B_k + A_k \alpha_k} + \frac{F_k - A_k \beta_k}{B_k + A_k \alpha_k}$$

Теперь введём обозначения:

$$\beta_{k+1} = \frac{F_k - A_k\beta_k}{B_k + A_k\alpha_k}$$

$$\alpha_{k+1} = -\frac{C_k}{B_k + A_k\alpha_k}$$

6. В результате k-е уравнение сведётся к виду $U_k = \alpha_{k+1}U_{k+1} + \beta_{k+1}$

Отсюда видно, что уравнение системы 2.1 для каждого $k = 1, ..., N-1$ приводится к виду П.6.

Прогоночные коэффициенты $\alpha_{k+1}, \beta_{k+1}$ вычисляются по формулам из П.5.

2.2.2. Обратный ход

Обратным ходом называется вычисление U_k N-е уравнение системы 2.1 не участвовало на этапе прямого хода. Пусть $k = N-1$. Того оно вместе с соотношением из П.6 образует систему:

$$\begin{cases} A_N U_{N-1} + B_N U_N = F_N, \\ U_{N-1} = \alpha_N U_N + \beta_N \end{cases}$$

Используем эту систему для определения U_N:

$$U_N = \frac{F_N - A_N\beta_N}{A_N\alpha_N + B_N}$$

Остальные U_k при $k = N-1, ..., 1$ считаем по формуле из П.6.

2.3. Модифицированный метод монотонной прогонки

Алгоритм состоит из двух шагов: прямой и обратных ход:

2.3.1. Прямой ход

На этом этапе система 2.1 приводится к двухдиагональному виду и вычисляются прогоночные коэффициенты.

1. Пусть $\bar{\alpha_1} = B_1, \bar{\beta_1} = C_1, \bar{\gamma} = F_1$ (первое уравнение системы 2.1 имеет двухчленный вид)

2. Тогда первое уравнение системы 2.1 примет вид:

$$\alpha_1 U_1 + \beta_1 U_2 = \gamma_1, \text{где} \alpha_1 = \frac{\bar{\alpha_1}}{\lambda_1}, \beta_1 = \frac{\bar{\beta_1}}{\lambda_1}, \gamma_1 = \frac{\bar{\gamma_1}}{\lambda_1}, \lambda_1 = \max(|\bar{\alpha_1}||\bar{\beta_1}||\bar{\gamma_1}|)$$

   Т.к. коэффициенты $\alpha_1, \beta_1, \gamma_1$ определены с точностью до произвольного множителя, то мы делим их на λ_1, чтобы по модулю они не были больше единицы.

3. Пусть $(k-1)$-е уравнение было приведено к двучленному виду

$$\alpha_{k-1} U_{k-1} + \beta_{k-1} U_k = \gamma k - 1$$

4. Уберём U_{k-1} из k-го уравнения 2.1, подставив в него соотношение из П.3

$$A_K \left(-\frac{\beta_{k-1}}{\alpha_{k-1}} U_k + \frac{\gamma_{k-1}}{\alpha_{k-1}} \right) + B_k U_k + C_k U_{k+1} = F_k$$

   Получаем:

$$\left(B_k - A_k \frac{\beta_{k-1}}{\alpha_{k-1}} \right) U_k + C_k U_{k+1} = F_k - A_k \frac{\gamma_{k-1}}{\alpha_{k-1}}$$

   Домножим на α_{k-1} и получим:

$$(\alpha_{k-1} B_k - \beta_{k-1} A_k) U_k + \alpha_{k-1} C_k U_{k+1} = \alpha_{k-1} F_k - \gamma_{k-1} A_k$$

5. Пусть

$$\bar{\alpha_k} = \alpha_{k-1} B_k - \beta_{k-1} A_k$$

$$\bar{\beta_k} = \alpha_{k-1} C_k$$

$$\bar{\gamma_k} = \alpha_{k-1} F_k - \gamma_{k-1} A_k$$

6. После нормировки получим соотношение:

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

где $\alpha_k = \dfrac{\bar{\alpha_k}}{\lambda_k}, \beta_k = \dfrac{\bar{\beta_k}}{\lambda_k}, \gamma_k = \dfrac{\bar{\gamma_k}}{\lambda_k}, \lambda_k = \max(|\bar{\alpha_k}||\bar{\beta_k}||\bar{\gamma_k}|)$

Строим прогоночные коэффициенты для $k = 2, 3, ..., N - 1$ по формулам из П.5 и П.6

2.3.2. Обратный ход

N-е уравнение системы 2.1 не участвовало на этапе прямого хода. Пусть $k = N - 1$. Того оно вместе с соотношением из П.6 образует систему:

$$\begin{cases} A_N U_{N-1} + B_N U_N = F_N, \\ \alpha_{N-1} U_{N-1} + \beta_{N-1} U_N = \gamma_{N-1} \end{cases}$$

Из этой системы найдём значения U_N и U_{N-1}:

$$U_N = \frac{A_N \gamma_{N-1} - \alpha_{N-1} F_N}{A_N \beta_{N-1} - \alpha_{N-1} B_N}$$

$$U_{N-1} = \frac{F_N \beta_{N-1} - \gamma_{N-1} B_N}{A_N \beta_{N-1} - \alpha_{N-1} B_N}$$

$U_i, i = N - 2, N - 3, ..., 1$ находятся из соотношения из П.6. Если $\alpha_i = 0, i = N - 2, ..., 1$, то U_i ищется их $i + 1$-го уравнения системы 2.1: $A_{i+1} U_i + B_{i+1} U_{i+1} + C_{i+1} U_{i+2} = F_{i+1}$

2.4. Метод немонотонной прогонки

Алгоритм состоит из двух шагов: прямой и обратных ход:

2.4.1. Прямой ход

На этом этапе система 2.1 приводится к двухдиагональному виду и вычисляются прогоночные коэффициенты.

1. Пусть $\bar{\alpha_1} = B_1, \bar{\beta_1} = C_1, \bar{\gamma} = F_1$ (первое уравнение системы 2.1 имеет двухчленный вид)

2. Тогда первое уравнение примет вид:

   $$\alpha_1 U_1 + \beta_1 U_2 = \gamma_1, \text{где} \alpha_1 = \frac{\bar{\alpha_1}}{\lambda_1}, \beta_1 = \frac{\bar{\beta_1}}{\lambda_1}, \gamma_1 = \frac{\bar{\gamma_1}}{\lambda_1}, \lambda_1 = \max(|\bar{\alpha_1}||\bar{\beta_1}||\bar{\gamma_1}|)$$

   Т.к. коэффициенты $\alpha_1, \beta_1, \gamma_1$ определены с точностью до произвольного множителя, то мы делим их на λ_1, чтобы по модулю они не были больше единицы.

3. Пусть $(k-1)$-е уравнение имеет двухчленный вид

   $$\alpha_{k-1} U_{k-1} + \beta_{k-1} U_k = \gamma_{k-1}$$

4. Уберём из k-го уравнения системы 2.1 U_{k-1}:

   $$A_k \left(-\frac{\beta_{k-1}}{\alpha_{k-1}} U_k + \frac{\gamma_{k-1}}{\alpha_{k-1}} \right) + B_k U_k + C_k U_{k+1} = F_k$$

5. Отсюда получаем:

   $$\left(B_k - A_k \frac{\beta_{k-1}}{\alpha_{k-1}} \right) U_k + C_k U_{k+1} = F_k - A_k \frac{\gamma_{k-1}}{\alpha_{k-1}}$$

6. Домножаем на α_{k-1} и получаем

   $$\left(B_k \alpha_{k-1} - A_k \beta_{k-1} \right) U_k + \alpha_{k-1} C_k U_{k+1} = F_k \alpha_{k-1} - A_k \gamma_{k-1}$$

7. Пусть:

$$\bar{\alpha_k} = \alpha_{k-1}B_k - \beta_{k-1}A_k$$

$$\bar{\beta_k} = \alpha_{k-1}C_k$$

$$\bar{\gamma_k} = \alpha_{k-1}F_k - \gamma_{k-1}A_k$$

8. После нормировки получим соотношение:

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

9. Где

$$\alpha_1 = \frac{\bar{\alpha_1}}{\lambda_1}, \beta_1 = \frac{\bar{\beta_1}}{\lambda_1}, \gamma_1 = \frac{\bar{\gamma_1}}{\lambda_1}, \lambda_1 = \max(|\bar{\alpha_1}||\bar{\beta_1}||\bar{\gamma_1}|)$$

Строим прогоночные коэффициенты для $k = 2, 3, ..., N-1$ по формулам из П.7 и П.9

2.4.2. Обратный ход

N-е уравнение системы 2.1 не участвовало на этапе прямого хода. Пусть $k = N-1$. Того оно вместе с соотношением из П.7 образует систему:

$$\begin{cases} A_N U_{N-1} + B_N U_N = F_N, \\ \alpha_{N-1}U_{N-1} + \beta_{N-1}U_N = \gamma_{N-1} \end{cases}$$

Из этой системы найдём значения U_N и U_{N-1}:

$$U_N = \frac{A_N\gamma_{N-1} - \alpha_{N-1}F_N}{A_N\beta_{N-1} - \alpha_{N-1}B_N}$$

$$U_{N-1} = \frac{F_N\beta_{N-1} - \gamma_{N-1}B_N}{A_N\beta_{N-1} - \alpha_{N-1}B_N}$$

Рассматривая матрицу системы из П.8, можно выделить 3 блока:

1. Строки, образующие блоки, для которых $|\bar{\alpha}_i| \geq |\bar{\beta}_i|$

2. Строки, образующие блоки, для которых $0 < |\alpha_i| < |\beta_i|$.

3. Строки, в которых $\alpha_i = 0$

Для блоков первого типа вычисляем значения искомого решения по формулам из П.8. В блоках третьего типа с помощью соответствующих уравнений системы 2.1. Если $a_k = 0, k = i$, то $U_i = \dfrac{F_{i+1} - B_{i+1}U_{i+1} - C_{i+1}U_{i+2}}{A_{i+1}}$

2.5. Анализ методов

2.5.1. Тестирование

Для тестирования была заранее подготовлена 1 матрица с низким числом обусловленности и сгенерировано несколько дополнительных. В начале программа считает решение встроенной функцией библиотеки "numPy", а после ищет его всеми реализованными методами. Были получены следующие результаты:

```
Matrix 0
[10.80000000 0.04750000 0.00000000 0.00000000 ] 12.14300000
[0.03210000 9.90000000 0.05230000 0.00000000 ] 10.08970000
[0.00000000 0.03690000 10.00000000 0.05700000 ] 13.67440000
[0.00000000 0.00000000 0.04160000 10.10000000 ] 10.89720000
condition number: 4.00953882
solution: 1.11991694 1.00835840 1.35760113 1.07333899
Monotonic sweep
    solution: 1.12101003 1.00846956 1.35760196 1.07893069
    error: 0.24832708
Modified monotonic sweep
    solution: 1.12435185 1.00845710 1.35760201 1.07893069
    error: 0.31103267
Non-monotonic sweep
    solution: 1.12435185 1.00845710 1.35760201 1.07893069
    error: 0.31103267

Matrix 1
[0.26863309 0.36615205 6.50998246 -4.61415283 ] 9.83609393
[-7.87945595 9.55596466 -2.65205935 2.05993260 ] 4.41323899
[7.84316026 6.99062114 9.20905422 0.58636303 ] 8.65763584
[1.99215155 -4.36586486 -8.38060017 -9.19537707 ] 8.06733597
condition number: 5.64506455
solution: 0.02611000 0.91826064 0.32282529 -1.60186933
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: 36.61534693 -25.46243101 -22.48434327 -0.87732519
    error: 2696.64446027
Non-monotonic sweep
    solution: 36.61534693 -25.46243101 -22.48434327 -0.87732519
    error: 2696.64446027

Matrix 2
[6.41852738 1.66074382 0.26378215 8.23272815 ] 5.51803606
[1.17879336 0.44727778 5.76904238 -3.75946184 ] 7.68699376
[6.51571817 1.42391411 -3.43243761 0.24211922 ] 7.03536132
[4.90973732 8.91963184 5.62577758 -0.49921111 ] 0.26500792
condition number: 6.34121541
solution: 1.82388567 -1.48269855 0.76383511 -0.47708701
```

```
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: 0.85970437 3.58085068 2.92713035 -0.53085341
    error: 222.09070289
Non-monotonic sweep
    solution: 0.85970437 3.58085068 2.92713035 -0.53085341
    error: 222.09070289

Matrix 3
[-3.58216222 2.99746567 -2.22953903 8.48035301 ] 8.47323827
[-1.73886220 9.01054542 3.15228588 3.70894467 ] 3.07487834
[-9.88323229 6.83099087 -2.02402681 -6.82103713 ] 3.37615158
[3.70221131 -1.60707824 -7.87380617 0.02544294 ] 4.89389603
condition number: 5.64856302
solution: -0.29216515 0.35641451 -0.82994352 0.53157263
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: -2.36539770 329.69093204 -984.65255396 192.34786974
    error: 96972.08914480
Non-monotonic sweep
    solution: -2.36539770 329.69093204 -984.65255396 192.34786974
    error: 96972.08914480

Matrix 4
[-4.02692116 2.47013600 5.40280056 3.38101947 ] 4.81422788
[5.02359106 -6.42078771 -3.59058828 8.84250151 ] 6.95948292
[8.49086248 -4.54804873 -0.56861170 0.74101630 ] 8.03551319
[4.21787773 8.26453130 -5.69075411 3.03142255 ] 9.68295109
condition number: 5.91184938
solution: 1.33912902 0.77437642 0.93039019 0.96635562
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: -1.19551083 -2.76964118 -6.58305317 3.19419380
    error: 437.92097825
Non-monotonic sweep
    solution: -1.19551083 -2.76964118 -6.58305317 3.19419380
    error: 437.92097825

Matrix 5
[-6.34645569 6.65924321 -0.93725333 -6.62344610 ] 8.92275204
[-3.58818260 9.78598914 0.90723116 6.00366937 ] 8.52010229
[-4.33931655 -3.89044160 -3.29310285 1.84987470 ] 2.89787559
[-2.59737484 -4.23199068 9.71958308 7.54777346 ] 0.91885569
condition number: 5.11015448
solution: -1.00892176 0.45418567 -0.04102113 0.08202757
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: -1.40594254 0.29506910 0.02445052 0.12173864
    error: 39.13764844
Non-monotonic sweep
```

```
    solution: −1.40594254 0.29506910 0.02445052 0.12173864
    error: 39.13764844

Matrix 6
[1.01833371 1.84606030 1.60096820 −7.39971245 ] 2.01194038
[6.24949742 −9.34895816 9.74875456 −6.85345890 ] 3.16406453
[7.79535465 1.54356817 9.04598832 −2.97679104 ] 4.18327580
[−9.25240422 −2.21545695 9.87246876 −5.72215029 ] 9.07629189
condition number: 6.18620061
solution: −0.28501464 0.21754866 0.63136508 −0.12024481
Monotonic sweep
    does not meet the condition
Modified monotonic sweep
    solution: 1.97571813 0.34275320 1.79777132 −1.58616804
    error: 399.30151595
Non−monotonic sweep
    solution: 1.97571813 0.34275320 1.79777132 −1.58616804
    error: 399.30151595
```

2.5.2. Погрешность

Погрешность вычислений будем считать по следующей формуле:

$$\frac{||X_{\text{точное}} - X_{\text{полученное}}||}{||X_{\text{точное}}||}$$

Рассмотренные методы имеют сложность $O(N)$. Это линейная сложность, но, несмотря на это, если мы имеем большое число обусловленности (больше 5), то появляется серьёзная погрешность.

3. Заключение

В результате работы над курсовым проектом мною были приобретены практические навыки владения:

1. современными численными методами решения задач линейной алгебры;
2. основами алгоритмизации для численного решения задач линейной алгебры на языке программирования "Python"; инструментальными средствами, поддерживающими разработку программного обеспечения для численного решения задач линейной алгебры;
3. навыками представления итогов проделанной работы в виде отчета, оформленного в соответствии с имеющимися требованиями, с привлечением современных средств редактирования.

После проведения тестов было обнаружено, что методы прогонки надежно работают с матрицами с подходящим числом обусловленности, в противном случае наблюдается значительное накопление ошибок.

В результате работы над курсовым проектом были изучены 3 метода прогонки, и был проведен их анализ. Дополнительно были разработаны программные реализации этих методов на языке программирования "Python", после чего были сделаны выводы об их эффективности.

4. Список использованных источников

1. Прогонки. Пособие в электронном виде.
2. 16. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978 г. – 592 с.
3. 18. Уилкинсон Дж.Х. Алгебраическая проблема собственных значений - Изд-во "Наука", 1970. -565 с.

5. Приложение (тексты программ)

```python
import numpy as np

def methodMonotonSweep(A, B, C, F):
  N = len(A)
  for i in range (1, N - 1):
    if np.abs(B[i]) < (np.abs(C[i]) + np.abs(A[i])):
      return "does not meet the condition"
  for i in range(N):
    if B[i] == 0:
      return "zeroes on the main diagonal"
  U = np.zeros(N)

  #Forward
  alpha = np.zeros(N)
  beta = np.zeros(N)
  alpha[0] = -C[0] / B[0]
  beta[0] = F[0] / B[0]
  for k in range(0, N-1):
    alpha[k+1] = -C[k] / (B[k] + A[k] * alpha[k])
    beta[k+1] = (F[k] - A[k] * beta[k]) / (B[k] + A[k] * alpha[k])

  #Backward
  U[N-1] = (F[N-1] - A[N-1] * beta[N-1]) / (A[N-1] * alpha[N-1] + B[N-1])
  for i in range(N - 2, -1, -1):
    U[i] = alpha[i + 1] * U[i + 1] + beta[i + 1]
  return U
```

Метод монотонной прогонки

```python
import numpy as np

def methodMonotonSweepModify(A, B, C, F):
  N = len(A)
  U = np.zeros(N)
  #Initial values
  lambda_max = max(F[0], B[0], C[0])
  alpha = np.zeros(N)
  beta = np.zeros(N)
  gamma = np.zeros(N)
  alpha[0] = B[0] / lambda_max
  beta[0] = C[0] / lambda_max
  gamma[0] = F[0] / lambda_max

  #Forward
  for k in range(1, N):
    alpha_k = alpha[k - 1] * B[k] - beta[k - 1] * A[k]
    beta_k = alpha[k-1] * C[k]
    gamma_k = alpha[k-1] * F[k] - gamma[k - 1] * A[k]
    max_k = max(alpha_k, beta_k, gamma_k)
    alpha[k] = alpha_k / max_k
    beta[k] = beta_k / max_k
    gamma[k] = gamma_k / max_k

  #Backward
  N = N - 1
  U[N] = (A[N]*gamma[N-1]-alpha[N-1]*F[N])/(A[N]*beta[N-1]-alpha[N-1]*B[N])
  U[N-1] = (F[N] * beta[N-1] - gamma[N-1] * B[N]) / (A[N] * beta[N-1] - alpha[N-1] * B[N])
  N += 1
  for i in range(N-3, -1, -1):
    if alpha[i] == 0:
      U[i] = (F[i+1]-B[i+1]*U[i+1]-C[i+1]*U[i+2])/A[i+1]
      print(U[i])
    else:
      U[i] = (gamma[i]-beta[i]*U[i+1])/alpha[i]
  return U
```

Модифицированный метод монотонной прогонки

```python
import numpy as np

def methodNotmonotonSweep(A, B, C, F):
  N = len(A)
  U = np.zeros(N)
  #Initial values
  lambda_max = max(F[0], B[0], C[0])
  alpha = np.zeros(N)
  beta = np.zeros(N)
  gamma = np.zeros(N)
  alpha[0] = B[0] / lambda_max
  beta[0] = C[0] / lambda_max
  gamma[0] = F[0] / lambda_max
  #Forward
  for k in range(1, N):
    alpha_k = alpha[k - 1] * B[k] - beta[k - 1] * A[k]
    beta_k = alpha[k-1] * C[k]
    gamma_k = alpha[k-1] * F[k] - gamma[k - 1] * A[k]
    max_k = max(alpha_k, beta_k, gamma_k)
    alpha[k] = alpha_k / max_k
    beta[k] = beta_k / max_k
    gamma[k] = gamma_k / max_k
  #Backward
  N = N - 1
  U[N] = (A[N]*gamma[N-1]-alpha[N-1]*F[N])/(A[N]*beta[N-1]-alpha[N-1]*B[N])
  U[N-1] = (F[N]*beta[N-1]-gamma[N-1]*B[N])/(A[N]*beta[N-1]-alpha[N-1]*B[N])
  N += 1
  for i in range(N - 3, -1, -1):
    if (abs(alpha[i]) >= abs(beta[i])) or ((abs(beta[i]))> (abs(alpha[i])) > 0):
      U[i] = -beta[i] * U[i+1]/alpha[i] + gamma[i]/alpha[i]
    elif (alpha[i] == 0):
      U[i] = (F[i+1] - B[i+1] * U[i+1] - C[i+1]*U[i+2]) / A[i+1]
  return U
```

Метод немонотонной прогонки

```
import numpy as np

def coefficients(A_matrix):
  N = A_matrix.shape[0]
  A = np.zeros(N)
  B = np.zeros(N)
  C = np.zeros(N)
  B = A_matrix.diagonal()
  for i in range(1, N):
    C[i] = np.copy(A_matrix[i-1][i])
  for i in range(0, N-1):
    A[i] = np.copy(A_matrix[i+1][i])
  return A, B, C

def writeAndPrint(s, file):
  print(s)
  file.write(s)
  return

def generateMatrixs(matrixs, f, N, tests):
  for i in range(tests):
    newM = np.random.uniform(low=-9, high=9, size=(N, N))
    newF = np.random.uniform(low=0, high=9, size = N)
    matrixs.append(newM)
    f.append(newF)
```

Вспомогательные функции

```python
import myLibs.helper as helper
import numpy as np
from methods import method_1, method_2, method_4

def main():
    file = open("tests.txt", "w", encoding='utf-8')
    tests = 6
   N = 4
    methods = {
    method_1.methodMonotonSweep : "Monotonic sweep",
    method_2.methodMonotonSweepModify: "Modified monotonic sweep",
    method_4.methodNotmonotonSweep: "Non-monotonic sweep"
    }
    matrixs = [np.array([[10.800, 0.0475, 0.0000, 0.0000],
                         [0.0321, 9.9000, 0.0523, 0.0000],
                         [0.0000, 0.0369, 10.000, 0.0570],
                         [0.0000, 0.0000, 0.0416, 10.100]]),]
    f = [np.array([12.143, 10.0897, 13.6744, 10.8972])]
    helper.generateMatrixs(matrixs, f, N, tests)

    for i in range(len(matrixs)):
        mat = matrixs[i]
        solution = np.linalg.solve(matrixs[i], f[i])
        helper.writeAndPrint(s=f"Matrix {i}\n", file=file)
        for j in range(len(matrixs[i])):
            helper.writeAndPrint(s=f"[", file=file)
            for x in matrixs[i][j]:
                helper.writeAndPrint(s=f"{x:.8f} ", file=file)
            helper.writeAndPrint(s=f"] {f[i][j]:.8f}\n", file=file)
        helper.writeAndPrint(s=f"condition number: {(np.linalg.norm(mat)*np.linalg.norm(np.linalg.inv(
            mat))):.8f}\n", file=file)
        helper.writeAndPrint(s=f"solution: ", file=file)
        for x in solution:
            helper.writeAndPrint(s=f"{x:.8f} ", file=file)
        helper.writeAndPrint(s=f"\n", file=file)
        for method, name in methods.items():
            A, B, C = helper.coefficients(mat)
            U = method(A, B, C, f[i])
            if not (isinstance(U, str)):
                helper.writeAndPrint (f"{name}\n    solution: ", file=file)
                for x in U:
                    helper.writeAndPrint(s=f"{x:.8f} ", file=file)
                helper.writeAndPrint(s=f"\n", file=file)
                helper.writeAndPrint (f"    error: {(np.linalg.norm(solution - U)/np.linalg.norm(solution)
                    *100):.8f}\n", file=file)
            else:
                helper.writeAndPrint(f"Monotonic sweep\n    {U}\n", file=file)
        helper.writeAndPrint(f"\n", file=file)
    file.close()

if __name__ == "__main__":
    main()
```

Тестирование

6. Решения теоретических задач

6.1. Задание 1

6.1.1. Условие

Докажите теоретически и проверьте экспериментально неравенство

$\frac{1}{n}M(A) \leq ||A||_2 \leq M(A)$, где $M(A) = n \times \underbrace{\max}_{1\leq i,j\leq n} |a_{ij}|$

6.1.2. Решение

Докажем неравенство теоретически:

1. Верхняя граница $\|A\|_2 \leq M(A)$: Обозначим через λ наибольшее собственное значение матрицы A^*A, где A^* - это эрмитово сопряжение матрицы A. Тогда $\|A\|_2 = \sqrt{\lambda}$. С учетом этого и свойств собственных значений, можно получить, что $\|A\|_2^2 = \lambda \leq \max(\text{собственные значения}(A^*A)) = M(A)^2$. Таким образом, верно, что $\|A\|_2 \leq M(A)$.
2. Нижняя граница $\frac{1}{n}M(A) \leq \|A\|_2$: Обозначим через λ_{min} - наименьшее собственное значение матрицы A^*A. Тогда $\frac{1}{n}M(A)^2 \leq \frac{1}{n}\max(\text{собственные значения}(A^*A)) = \frac{1}{n}\max(\lambda) \leq \lambda_{\text{max}} \leq \|\|A\|_2\|\|_2^2 = \|A\|_2^2$. Следовательно $\frac{1}{n}M(A) \leq \|A\|_2$.

Теперь проверим это неравенство с помощью кода на Python, используя библиотеку "numPy":

```
import numpy as np

n = 3
A = np.random.rand(n, n)

spectral_norm = np.linalg.norm(A, 2)
M_A = n * np.max(np.abs(A))

if (1/n) * M_A <= spectral_norm <= M_A:
    print("True")
else:
    print("False")
```

Код

Этот код создает случайную матрицу A, вычисляет спектральную норму и значение M(A), затем проверят неравенство на основе этих значений.

6.2. Задание 2

6.2.1. Условие

Пусть $Ax = f, (A + \sigma A)(x + \sigma x) = f$ - возмущенная система линейных алгебраических уравнений.

Пусть $\det(A) \neq 0, \det(A + \sigma A) \neq 0$.

Докажите, что $\dfrac{||\sigma x||}{||x + \sigma x||} \leq \mu(A) \dfrac{||\sigma A||}{||A||}$

6.2.2. Решение

Возьмем возмущенную систему линейных алгебраических уравнений:

$$Ax = f, (A + \sigma A)(x + \sigma x) = f$$

Если $\det(A) \neq 0, \det(A + \sigma A) \neq 0$, то матрицы A и $A + \sigma A$ обратимы.

Теперь рассмотрим отношение:

$$\frac{(A + \sigma A)(x + \sigma x)}{f}$$

Упростим это выражение:

$$\frac{(A + \sigma A)(x + \sigma x)}{f} = \frac{x + \sigma x}{f} \cdot (A + \sigma A) = \frac{x + \sigma x}{f} \cdot A \cdot (1 + \sigma)$$

Далее, разделим обе части на $x + \sigma x$:

$$\frac{A \cdot (1 + \sigma)}{\frac{x + \sigma x}{f}}$$

На этом этапе мы увидим, что $\dfrac{x + \sigma x}{f}$ - это вектор. Тогда выражение может быть переписано в виде:

$$\frac{A \cdot (1 + \sigma)}{v},$$

где $v = \dfrac{x + \sigma x}{f}$.

Теперь мы можем применить неравенство числа обусловленности μ:

$$\frac{||\sigma A||}{||A||} \geq \frac{||\sigma x||}{||x||},$$

Таким образом, теоретически утверждение доказано.

Теперь проверим это утверждение с помощью кода на Python, используя библиотеку "numPy":

```
import numpy as np

#Iint
n = 3
A = np.random.rand(n, n)
f = np.random.rand(n)

sigma = np.random.rand()
x = np.random.rand(n)

#condNumber
cond_A = np.linalg.cond(A)
cond_A_sigma = np.linalg.cond(A + sigma * A)

#Check
left_side = sigma * x / (x + sigma * x)
right_side = cond_A * sigma * A / A
if np.all(left_side <= right_side):
    print("True")
else:
    print("False")
```

Код