МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1

ОТЧЕТ

о выполненном задании

студента 204 учебной группы факультета ВМК МГУ

Митрофанова Андрея Александровича

(фамилия, имя, отчество студента)

гор. Москва

2022 г.

Оглавление

Решение СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Написать программу, решающую систему линейных алгебраических уравнений заданного пользователем размера (n - параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента.

Цели и задачи практической работы

1. Изучить метод Гаусса.
2. Реализовать метод Гаусса для
    - решения СЛАУ.
    - Вычисления определителя.
    - Нахождения обратной матрицы.
3. Вычислить число обусловленности матрицы $M_A = ||A^{-1}|| * ||A||$, если существует матрица A^{-1}.
4. Исследовать вопрос вычислительной устойчивости метода Гаусса.
5. Подтвердить корректность работы программы с использованием набора тестов.

Описание алгоритма

Рассмотрим систему линейных алгебраических уравнения вида:

$$\begin{cases} aa_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = f_1 \\ aa_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = f_2 \\ \cdots \\ aa_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = f_n \end{cases}$$

с невырожденной матрицей $A = (a_{ij})$

Метод Гаусса

Метод Гаусса разделим на 2 этапа:

1. Прямой ход: система приводится к треугольному виду.
2. Обратный ход: последовательное отыскание неизвестных $x_1, x_2, \ldots, x_n$.

Прямой ход заключается в преобразовании исходной системы к эквивалентной системе линейных уравнений с верхнетреугольной матрицей. Для этого разделим все члены первого уравнения на $a_{11} \neq 0$ (если $a_{11} = 0$, поменяем местами уравнения с номерами 1 и i, где $a_{i1} \neq 0$, такой элемент найдется в силу невырожденности матрицы):

Затем вычитаем получившуюся после перестановки первую строку из остальных строк, получим:

$$\begin{cases} x_1 + \frac{a_{12}}{a_{11}} * x_2 + ... + \frac{a_{1n}}{a_{11}} * x_n = \frac{f_1}{a_{11}} \\ a_{22}^{(1)} * x_2 + ... + a_{2n}^{(1)} * x_n = f_2^{(1)} \\ ..... \\ a_{n2}^{(1)} * x_2 + ... + a_{nn}^{(1)} * x_n = f_n^{(1)} \end{cases},$$

где

$$a_{ij}^{(1)} = a_{ij} - a_{i1} * \frac{a_{1j}}{a_{11}}, \quad f_i^{(1)} = f_i - a_{i1} * \frac{f_1}{a_{11}}$$

После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают указанный процесс для всех последующих уравнений пока не останется уравнение с одной неизвестной. Переобозначим полученные коэффициенты как c_{ij} для неизвестных x_i, а правую часть через y_i, тогда получим систему вида:

$$\begin{cases} x_1 + c_{12} * x_2 + c_{13} * x_3 + ... + c_{1n} * x_n = y_1 \\ x_2 + c_{23} * x_3 + ... + c_{2n} * x_n = y_2 \\ ... \\ x_{n-1} + c_{n-1,n} * x_n = y_{n-1} \\ x_n = y_n \end{cases}$$

Обратный ход: обратная подстановка предполагает подстановку полученного на предыдущем шаге значения переменной x_i в предыдущие уравнения:

$$\begin{cases} x_n = y_n \\ x_{n-1} = y_{n-1} - c_{n-1,n} * x_n \\ ... \\ x_1 = y_1 - c_{12} * x_2 - c_{13} * x_3 - ... - c_{1n} * x_n \end{cases}$$

Метод Гаусса с выбором главного элемента

Модифицированный метод Гаусса отличается от рассмотренного в 1 пункте тем, что на этапе прямого хода, когда производится нормировка уравнений, каждый раз выбирается максимальный элемент и строка с этим элементом перемещается наверх. Все остальные шаги выполняются аналогично.

Определитель матрицы

Для вычисления определителя матрицы заметим, что определитель треугольной матрицы равен произведению ее диагональных элементов. Для приведения матрицы к верхнетреугольному виду воспользуемся алгоритмом, описанным в прямом ходе метода Гаусса и будем учитывать, что при делении строки на элемент матрицы, определитель матрицы умножается на этот элемент.

Обратная матрица

Обратную матрицу вычислим по методу Гаусса-Жордана: если с единичной матрицей E провести элементарные преобразования, которыми невырожденная квадратная матрица A приводится к E, то получится обратная матрица A^{-1} . Все преобразования будем проводить с расширенной матрицей $A|E$. Приведение матрицы A к единичной заключается в приведении ее сначала к верхнетреугольной, а затем к нижнетреугольной методом из п.1 Гаусса, таким образом главная диагональ будет нормирована.

Число обусловленности матрицы

Будем считать норму матрицы следующим образом:

$$\|\mathrm{A}\| = \max_{1 \le i \le n} \sum_{i=1}^{n} \left|a_{ij}\right|$$

Число обусловленности $M_A = ||A|| * ||A^{-1}||$

Тестирование и результаты эксперимента

Тестирование программы производилось с использованием предоставленных тестов, а также тестов, сгенерированных с использованием готовых формул. Ниже приведены входные данные и результаты работы программы на этих данных.

Тест 1

$$\begin{cases} 2x_1 + 2x_2 - x_3 + x_4 = 4, \\ 4x_1 + 3x_2 - x_3 + 2x_4 = 6, \\ 8x_1 + 5x_2 - 3x_3 + 4x_4 = 12, \\ 3x_1 + 3x_2 - 2x_3 + 4x_4 = 6. \end{cases}$$

Решение, найденное программой:

Расширенная матрица исходной системы линейных алгебраических уравнений:

$$A|b = \begin{pmatrix} 2 & 2 & -1 & 1 & 4 \\ 4 & 3 & -1 & 2 & 6 \\ 8 & 5 & -3 & 4 & 12 \\ 3 & 3 & -2 & 4 & 6 \end{pmatrix}$$

Определитель $|A|$ = 10.0000000

Обратная матрица:

$$A^{-1} = \begin{pmatrix} -0.6000 & 0.1000 & 0.3000 & -0.2000 \\ 1.0000 & 0.5000 & -0.5000 & 0.0000 \\ -1.0000 & 1.5000 & -0.5000 & 0.0000 \\ -0.8000 & 0.3000 & -0.1000 & 0.4000 \end{pmatrix}$$

Число обусловленности: 60.0000

Решения, полученные с помощью метода Гаусса:

X_1 = 0.6000000

X_2 = 1.0000000

X_3 = -1.0000000

X_4 = -0.2000000

Решения, полученные с помощью метода Гаусса с выбором главного элемента:

x_1 = 0.6000000

x_2 = 1.0000000

x_3 = −1.0000000

x_4 = −0.2000000

Ожидаемое решение (решение онлайн-системы WolframAlpha):

X_1 = 0.6

X_2 = 1

X_3 = -1

X_4 = -0.2

Тест 2

$$\begin{cases} 2x_1 + 5x_2 - 8x_3 + 3x_4 = 8, \\ 4x_1 + 3x_2 - 9x_3 + x_4 = 9, \\ 2x_1 + 3x_2 - 5x_3 - 6x_4 = 7, \\ x_1 + 8x_2 - 7x_3 = 12. \end{cases}$$

$$\begin{cases} x_1 + x_2 + 3x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 4x_3 - x_4 = 2, \\ 3x_1 + 3x_2 + 5x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 8x_3 - 3x_4 = 2. \end{cases}$$

Решение, найденное программой:

Расширенная матрица исходной системы линейных алгебраических уравнений:

Определитель $|A| = 0.0000000$

Данная система вырождена, поэтому имеет бесконечно много решений.

Тест 3

Решение, найденное программой:

Расширенная матрица исходной системы линейных алгебраических уравнений:

$$A|b = \begin{pmatrix} 2 & 5 & -8 & 3 & 8 \\ 4 & 3 & -9 & 1 & 9 \\ 2 & 3 & -5 & -6 & 7 \\ 1 & 8 & -7 & 0 & 12 \end{pmatrix}$$

Определитель $|A| = -189.0000000$.

$$A|b = \begin{pmatrix} 1 & 1 & 3 & -2 & 1 \\ 2 & 2 & 4 & -1 & 2 \\ 3 & 3 & 5 & -2 & 1 \\ 2 & 2 & 8 & -3 & 2 \end{pmatrix}$$

Обратная матрица:

$$A^{-1} = \begin{pmatrix} -1.7196 & 1.2222 & -0.6561 & 0.8624 \\ -0.6508 & 0.3333 & -0.2698 & 0.5079 \\ -0.9894 & 0.5556 & -0.4021 & 0.5608 \\ -0.0741 & 0.1111 & -0.1852 & 0.0741 \end{pmatrix}$$

Число обусловленности: 80.2857.

Решения, полученные с помощью метода Гаусса:
x_1 = 3.0000000
x_2 = 2.0000000

$$\mathrm{A}_{ij} = \begin{cases} \mathrm{q}_M^{i+j} + 0.1(j-i), i \neq j, \\ (\mathrm{q}_M - 1)^{i+j}, i = j, \end{cases}$$

x_3 = 1.0000000
x_4 = 0.0000000

Решения, полученные с помощью метода Гаусса с выбором главного элемента:
x_1 = 3.0000000
x_2 = 2.0000000
x_3 = 1.0000000
x_4 = -0.0000000

Ожидаемое решение, полученое с помощью WolframAlfa**:**
X_1 = 3
X_2 = 2
X_3 = 1
X_4 = 0

Тест 4

В данном тесте элементы матрицы A вычисляются по формулам:

где $q_M = 1.001 - 2 \times M \times 10^{-3},\ i,j = \overline{1,\ldots,n}.$

Элементы вектора f (вектор правой части системы) задаются формулами:

$$b_i = x * exp(\frac{x}{i}) * cos(\frac{x}{i}), i = 1, ..., n$$

В данном варианте M = 6, n = 100, неизвестная $x = 50$ выбрана случайно.

Решение, полученное программой:

```
x_1 = -253294058712648153825280.000000
x_2 = 2621781863950192214016.0000000
x_3 = 2758548097305612910592.0000000
x_4 = 2896272019899044331520.0000000
x_5 = 3034885197017157992448.0000000
x_6 = 3174315215085265485824.0000000
x_7 = 3314485494601852911616.0000000
x_8 = 3455315134639639101440.0000000
x_9 = 3596718752420155883520.0000000
x_10 = 3738606317814926016512.0000000
x_11 = 3880882982622364434432.0000000
x_12 = 4023448904472522653696.0000000
x_13 = 4166199065199555641344.0000000
x_14 = 4309023083530614210560.0000000
x_15 = 4451805021911128735744.0000000
x_16 = 4594423187319983964160.0000000
x_17 = 4736749925875362824192.0000000
x_18 = 4878651411077267456000.0000000
x_19 = 5019987425485190594560.0000000
x_20 = 5160611135652938907648.0000000
x_21 = 5300368860123246886912.0000000
x_22 = 5439099830288226713600.0000000
x_23 = 5576635943902545707008.0000000
x_24 = 5712801511047435386880.0000000
x_25 = 5847412992334358380544.0000000
x_26 = 5980278729109825126400.0000000
x_27 = 6111198665454694432768.0000000
x_28 = 6239964061731986079744.0000000
x_29 = 6366357199447482433536.0000000
x_30 = 6490151077176309448704.0000000
x_31 = 6611109097297724571648.0000000
x_32 = 6728984743295008636928.0000000
x_33 = 6843521247337147006976.0000000
x_34 = 6954451247863856889856.0000000
x_35 = 7061496436915778355200.0000000
x_36 = 7164367196896296435712.0000000
x_37 = 7262762226503291764736.0000000
x_38 = 7356368155453802151936.0000000
x_39 = 7444859147791177351168.0000000
x_40 = 7527896493366817325056.0000000
x_41 = 7605128187207720370176.0000000
x_42 = 7676188496410863206400.0000000
x_43 = 7740697514261076770816.0000000
x_44 = 7798260701131720097792.0000000
x_45 = 7848468411893117091840.0000000
x_46 = 7890895409392503488512.0000000
x_47 = 7925100363638552657920.0000000
x_48 = 7950625336276565360640.0000000
x_49 = 7966995249994792960000.0000000
x_50 = 7973717342369116848128.0000000
x_51 = 7970280603795934150656.0000000
x_52 = 7956155198982378749952.0000000
x_53 = 7930791871644485812224.0000000
x_54 = 7893621331853309902848.0000000
x_55 = 7844053625614446362624.0000000
x_56 = 7781477486153529556992.0000000
x_57 = 7705259666419697582080.0000000
x_58 = 7614744252313816268800.0000000
x_59 = 7509251956031075909632.0000000
x_60 = 7388079389103684059136.0000000
x_61 = 7250498314430872289280.0000000
x_62 = 7095754876874446077952.0000000
x_63 = 6923068811686159843328.0000000
x_64 = 6731632630278054215680.0000000
x_65 = 6520610782679720263680.0000000
x_66 = 6289138795964567388160.0000000
x_67 = 6036322388164124606464.0000000
x_68 = 5761236556817660641280.0000000
x_69 = 5462924641563542814720.0000000
x_70 = 5140397360047795994624.0000000
x_71 = 4792631816486771490816.0000000
x_72 = 4418570481961808166912.0000000
x_73 = 4017120145878420029440.0000000
x_74 = 3587150837706898014208.0000000
x_75 = 3127494718184283439104.0000000
x_76 = 2636944939254145875968.0000000
x_77 = 2114254471723486806016.0000000
x_78 = 1558134899904956596224.0000000
x_79 = 967255182397592567808.0000000
x_80 = 340240377820447899648.0000000
x_81 = -324329665073501044736.0000000
x_82 = -1027921654164905918464.0000000
x_83 = -1772050238494749753344.0000000
x_84 = -2558279431537063624704.0000000
x_85 = -3388224074237126639616.0000000
x_86 = -4263551339156206518272.0000000
x_87 = -5185982276346962771968.0000000
x_88 = -6157293402509840220160.0000000
x_89 = -7179318334431460065280.0000000
x_90 = -8253949467754060840960.0000000
x_91 = -9383139702680156372992.0000000
x_92 = -10568904217309197893632.0000000
x_93 = -11813322290362480328704.0000000
x_94 = -13118539174476022546432.0000000
x_95 = -14486768021371185266688.0000000
x_96 = -15920291860053204402176.0000000
x_97 = -17421465630099380371456.0000000
x_98 = -18992718270744030609408.0000000
x_99 = -20636554867860570112000.0000000
x_100 = -22355558859955048022016.0000000
```

Выводы

При выполнении поставленных целей был подробно изучен и воплощен метод Гаусса. Он достаточно удобен в использовании для систем с относительно небольшим порядком. Для плохо обусловленных же матриц и матриц достаточно большого порядка метод является вычислительно неустойчивым, что продемонстрировало бы различие в решениях, найденных обычными методом Гаусса и методом Гаусса с выбором главного элемента.

Решение СЛАУ итерационными методами. Метод верхней релаксации

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n*n$ с невырожденной матрицей A. Написать программу численного решения данной системы линейных алгебраических уравнений (n - параметр программы), использующую численный алгоритм итерационного метода верхней релаксации. Итерационный процесс имеет следующий вид:

$$(D + wA^{-})\frac{x^{k+1}-x^{k}}{w} + Ax^{k} = f$$

где w - итерационный параметр (при w = 1 метод верхней релаксации совпадает с методом Зейделя). Рекурректные формулы:

$$x_i^{k+1} = x_i^k + \frac{w}{a_{ii}}\left(f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right), i = \overline{1, \dots, n}.$$

Цели и задачи практической работы

1. Изучить метод верхней релоксации.
2. Реализовать метод верхней релаксации.
3. Разработать критерий остановки итерационного процесса, гарантирующий получение решения исходной СЛАУ с заданной точностью.
4. Изучить скорость сходимости итераций к точному решению задачи в зависимости от итерационного параметра ω.
5. Подтвердить корректность работы программы с использованием набора тестов.

Описание алгоритма

Рассмотрим систему линейных алгебраических уравнения вида: Рассмотрим систему линейных алгебраических уравнения вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = f_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = f_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = f_n \end{cases}$$

с невырожденной матрицей $A = (a_{ij})$.

Для метода верхней релаксации матрица A записывается в виде $A = A^{-} + D + A^{+}$, где D - диагональная матрица, т.ч. ее главная диагональ совпадает с главной диагональю матрицы A, A^{-} - нижняя треугольная матрица $n * n$, т.ч. ее ненулевые (поддиагональные) элементы также совпадают с элементами A, а главная

диагональ является нулевой. Аналогично, A^+ - верхняя треугольная матрица $n \times n$, ненулевые(наддиагональные) элементы которой совпадают с элементами A, а главная диагональ также является нулевой.

Был протестирован метод верхней релаксации, при этом критерием остановки служила норма невязки:

$$||x^{(s)-x}|| = ||z_{(s)}|| < ||A^{-1}||||\psi_{(s)}|| = \epsilon$$

где $x^{(s)}$ - приближение, полученное на итерации с номером s, $\psi_{(s)}$ - невязка уравнения, ϵ - точность, с которой необходимо вычислить приближенное значение.

Были исследованы разные параметры метода верхней релаксации. В итоге выявлено, что для каждой матрицы оптимальный параметр отличается, и для каждого параметра, таким образом, скорость отличается. Также данный параметр в большинстве случаев не может быть вычислен заранее, поэтому чаще всего подбирается.

Тестирование и результаты эксперимента

Тестирование программы производилось с использованием предоставленных тестов, а также тестов, сгенерированных с использованием готовых формул. Ниже приведены входные данные и результаты работы программы на этих данных. Помимо непосредственно решения системы, программа выводит количество итераций, затраченных на вычисление результата с нужной точностью.

Тест 1

$$\begin{cases} 2x_1 + 2x_2 - x_3 + x_4 = 4, \\ 4x_1 + 3x_2 - x_3 + 2x_4 = 6, \\ 8x_1 + 5x_2 - 3x_3 + 4x_4 = 12, \\ 3x_1 + 3x_2 - 2x_3 + 4x_4 = 6. \end{cases}$$

параметр ω	решение	итераций для 0.01	итераций для 0.00001
0.75	$x_1 = 0.6000017$ $x_2 = 0.9999983$ $x_3 = -0.9999956$ $x_4 = -0.1999979$	526	1166
1	$x_1 = 0.5999994$ $x_2 = 0.9999996$ $x_3 = -1.0000036$ $x_4 = -0.2000010$	304	685

1.25	$x_1 = 0.6000002$ $x_2 = 0.9999987$ $x_3 = -1.0000019$ $x_4 = -0.2000000$	294	625
1.5	$x_1 = 0.6000003$ $x_2 = 0.9999986$ $x_3 = -1.0000012$ $x_4 = -0.1999996$	406	831
1.75	$x_1 = 0.6000023$ $x_2 = 0.9999969$ $x_3 = -0.9999958$ $x_4 = -0.1999973$	807	1624

Ожидаемое решение, полученное с помощью WolframAlpha:

$x_1 = 0.6$

$x_2 = 1$

$x_3 = -1$

$x_4 = -0.2$

Наилучший выбор параметра $\omega = 1.25$

Тест 2

$$\begin{cases} x_1 + x_2 + 3x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 4x_3 - x_4 = 2, \\ 3x_1 + 3x_2 + 5x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 8x_3 - 3x_4 = 2. \end{cases}$$

Данная система вырождена, поэтому имеет бесконечно много решений

Тест 3

$$\begin{cases} 2x_1 + 5x_2 - 8x_3 + 3x_4 = 8, \\ 4x_1 + 3x_2 - 9x_3 + x_4 = 9, \\ 2x_1 + 3x_2 - 5x_3 - 6x_4 = 7, \\ x_1 + 8x_2 - 7x_3 = 12. \end{cases}$$

параметр ω	решение	итераций для 0.01	итераций для 0.00001

0.75	$x_1 = 2.9999941$ $x_2 = 1.9999977$ $x_3 = 0.9999967$ $x_4 = -0.0000005$	2871	6423
1	$x_1 = 2.9999954$ $x_2 = 1.9999982$ $x_3 = 0.9999974$ $x_4 = -0.0000004$	1633	3762
1.25	$x_1 = 3.00000292$ $x_2 = 2.0000011$ $x_3 = 1.0000016$ $x_4 = 0.0000002$	759	2032
1.5	$x_1 = 3.0000016$ $x_2 = 2.0000006$ $x_3 = 1.0000009$ $x_4 = 0.0000001$	743	1441
1.75	$x_1 = 3.0000006$ $x_2 = 2.0000002$ $x_3 = 1.0000003$ $x_4 = 0.0000000$	389	661

Ожидаемое решение, полученное с помощью WolframAlpha:

$X_1 = 3$

$X_2 = 2$

$X_3 = 1$

$X_4 = 0$

Наилучший выбор параметра $\omega = 1.75$

Выводы

В данной работе необходимо было изучить и реализовать итерационный метод верхней релаксации. Скорость сходимости данного метода существенно зависит от итерационного параметра ω (для сходимости метода необходимо $0<\omega<2$). Из вышеприведенной таблицы можно сделать вывод, что оптимальный параметр для скорости сходимости ω^* зависит от самих матриц исходных СЛАУ.

Программа

```python
# coding=utf-8
import math
import numpy as np

# Вычисление решения СЛАУ методом Гаусса
# СЛАУ задается в виде матрицы data размером n на n
def gauss(data, n):
    data_copy = np.copy(data)

    for i in range(0, n):
        if data_copy[i][i] == 0:
            for j in range(i + 1, n):
                if data_copy[j][i] != 0:
                    data_copy[[j, i]] = data_copy[[i, j]]
                    break
        data_copy[i] /= data_copy[i][i]
        for j in range(i + 1, n):
            data_copy[j] -= data_copy[i] * data_copy[j][i]

    x = data_copy[:, n]
    for i in range(n - 1, -1, -1):
        for j in range(n - 1, i, -1):
            x[i] -= data_copy[i][j] * x[j]
    return x


# Вычисление решения СЛАУ методом Гаусса с выбором главного элемента
# СЛАУ задается в виде матрицы data размером n на n
def gauss_diag(data, n):
    data_copy = np.copy(data)
    idx = np.array(range(0, n))
    for i in range(0, n):
        if data_copy[i][i] == 0:
            for j in range(i + 1, n):
                if data_copy[j][i] != 0:
                    data_copy[[j, i]] = data_copy[[i, j]]
                    break
        col_max = i + np.argmax(abs(data_copy[i][i:n]))
        idx[i], idx[col_max] = idx[col_max], idx[i]
        data_copy[:, [i, col_max]] = data_copy[:, [col_max, i]]
```

```
        data_copy[i] /= data_copy[i][i]
        for j in range(i + 1, n):
            data_copy[j] -= data_copy[i] * data_copy[j][i]
    x = data_copy[:, n]
    for i in range(n - 1, -1, -1):
        for j in range(n - 1, i, -1):
            x[i] -= data_copy[i][j] * x[j]

    res = np.array(list(zip(x, idx)))
    res = res[res[:, 1].argsort()]
    x = res[:, 0]
    return x


# Функция, вычисляющая определитель матрицы data, размер которой n на n
def det_calc(data, n):
    det = 1
    data_copy = np.copy(data[:, :n])
    for i in range(0, n):
        if data_copy[i][i] == 0:
            for j in range(i + 1, n):
                if data_copy[j][i] != 0:
                    data_copy[[j, i]] = data_copy[[i, j]]
                    det *= -1
                    break
            return 0
        det *= data_copy[i][i]
        for j in range(i + 1, n):
            data_copy[j] -= data_copy[i] * data_copy[j][i] / data_copy[i][i]
    return det


# Вычисление обратной матрицы к матрице data, размер которой n на n
def inverse(data, n):
    data_copy = np.hstack((np.copy(data[:, :n]), np.identity(n)))
    for i in range(0, n):
        if data_copy[i][i] == 0:
            for j in range(i + 1, n):
                if data_copy[j][i] != 0:
                    data_copy[[j, i]] = data_copy[[i, j]]
                    break
        data_copy[i] /= data_copy[i][i]
        for j in range(i + 1, n):
            data_copy[j] -= data_copy[i] * data_copy[j][i]
    for j in range(n - 1, 0, -1):
        for i in range(j - 1, -1, -1):
            data_copy[i] -= data_copy[j] * data_copy[i][j]
    inv = data_copy[:, n:]
    return inv
```

```
# Вычисление числа обусловленности для квадратной матрицы data, со стороной n
def mat_norm(data, n):
    max = 0
    for i in range(0, n):
        max_i = 0

    for j in range(0, n):
        max_i += abs(data[i][j])
    if max_i > max:
        max = max_i
    return max


# Проверка матрицы data размеров n на n
# на самосопряженность и положительную определенность
# параметр функции pr_par отвечает за вывод (если pr_par = 1),
# либо за возврат значения (1, если матрица самосопр. и полож. опред.)
# функцией (pr_par = 0)
def herm_and_pos(data, n, pr_par):
    herm = 0
    pos_def = 1
    data_copy = np.copy(data[:, :n])
    # Проверка самосопряжённости
    if np.array_equal(data_copy, data_copy.T):
        herm = 1

    # Проверка матрицы на положительную определенность
    if data_copy[0][0] > 0:
        for i in range(2, n + 1):
            corn_minor = det_calc(data_copy[:i, :i], i)
            if corn_minor <= 0:
                pos_def = 0
                break
    if pr_par == 1:
        if (herm == 0) and (pos_def == 0):
            print("\nМатрица коэффициентов данной системы не "
                  "является самосопряжённой и положительно определённой")
        elif (herm == 0) and (pos_def == 1):
            print("\nМатрица коэффициентов данной системы не "
                  "является самосопряжённой, но является положительно "
                  "определённой")
        elif (herm == 1) and (pos_def == 1):
            print("\nМатрица коэффициентов данной системы "
                  "является самосопряжённой, но не "
                  "является положительно "
                  "определённой")
        else:
            print("\nМатрица коэффициентов данной системы "
                  "является самосопряжённой и положительно"
                  "определённой")
    else:
        return herm & pos_def
```

```python
# Метод верхней релаксации
def relax(data, n, eps, w):
    data_copy = np.copy(data[:, :n])
    f = np.copy(data[:, n])
    if herm_and_pos(data_copy, n, 0) == 1:
        f = data_copy.T @ f
        data_copy = data_copy.T @ data_copy
        cond_num = mat_norm(data_copy, n) * mat_norm(inverse(data, n), n)
        print("\nЧисло обусловленности полученной "
              "симметрической и положительно определённой "
              "матрицы: %.4f" % cond_num)
    else:
        cond_num = mat_norm(data_copy, n) * mat_norm(inverse(data_copy, n), n)
        print("\nЧисло обусловленности: %.4f" % cond_num)

    x_new = np.zeros(n)
    discrep_y = eps + 1
    iter = 0
    first_m = np.zeros((n, n))
    for i in range(0, n):
        first_m[i][i] = data_copy[i][i] / w
    first_m += np.tril(data_copy, k=-1)
    if det_calc(first_m, n) == 0:
        return 1
    first_m = inverse(first_m, n)
    while (eps <= discrep_y) and (iter <= 500000):
        iter += 1
        x_prev = np.copy(x_new)
        second_m = f - (data_copy @ x_prev)
        x_new = (first_m @ second_m) + x_prev
        discrep_y = (np.linalg.norm(f - (data_copy @ x_new), 2)) * \
                    np.linalg.norm(np.linalg.inv(data_copy), 2)
    if iter > 500000:
        print("\nКол-во итераций превысило 500000")
    else:
        print("\nРешения, полученные с помощью"
              " метода верхней релаксации:")
        for i in range(0, n):
            print("x_%d = %.7f" % (i + 1, x_new[i]))
        print("\nКол-во итераций:", iter)
```

```python
# Вычисление элементов матрицы по формуле
def formula(n, m):
    x = 50
    data = np.zeros((n, n + 1))
    for i in range(1, n + 1):
        data[i - 1][n] = x * math.exp(x / i) * math.cos(x / i)
        for j in range(1, n + 1):
            q = 1.001 - 2 * m * 0.001
            if i == j:
                data[i - 1][j - 1] = (q - 1) ** (i + j)
            else:
                data[i - 1][j - 1] = q ** (i + j) + 0.1 * (j - i)
    return data


# Тело основной функции
global matrix
choice = 1
while choice == 1:  # Бесконечный цикл вычислений решений СЛАУ
    print("\nКаким методом Вы хотите решать "
          "систему?\n1 - методы Гаусса (1 подзадание)\n2 - метод верхней "
          "релаксации (2 подзадание)")
    task_num = int(input("\nВведите соответствующий номер: "))
    if (task_num < 1) or (task_num > 2):
        print("\nНеправильный номер")
        continue
    print("\nКак Вы хотите ввести систему: "
          "1 - из файла, 2 - с помощью формулы?")
    mode = int(input("\nВведите соответствующий номер: "))
    if mode == 1:  # Если ввод системы с из файла, названного "sys"+"№"+".txt"
        sys_num = int(input("\nНомер системы, "
                            "которую Вы хотите ввести (1/2/3): "))
        if (sys_num > 6) or (sys_num < 1):
            print("\nНеправильный номер системы\n")
            continue
        else:
            file_name = str("sys" + str(sys_num) + ".txt")
            f = open(file_name, 'r')
            file_contents = f.read()
            print("\nВы выбрали систему\n", file_contents, sep='')
            matrix = np.loadtxt(file_name)
    elif mode == 2:  # Если ввод системы с помощью формулы
        print()
        n = 100
        m = 6
        matrix = formula(n, m)
        print("\nПолученная система:")
        for i in range(0, n):
            for j in range(0, n):
                print("%*.4f " % (7, matrix[i][j]), end="")
            print("| %*.4f " % (7, matrix[i][n]))
    else:
        print("\nНеправильный способ ввода")
        continue
```

```
    cnt = matrix.shape[0]

    det = det_calc(matrix, cnt)
    print("\nОпределитель: %.7f" % det)
    if det == 0:  # Если определитель матрицы maxtrix = 0
        print("\nДанная система вырождена, "
              "поэтому имеет бесконечно много решений")
    elif task_num == 1:  # Если был выбран поиск решений СЛАУ
                         # с помощью метода Гаусса
        inv = inverse(matrix, cnt)
        print("\nОбратная матрица:")
        for i in range(0, cnt):
            for j in range(0, cnt):
                print("%*.4f " % (7, inv[i][j]), end="")
            print()

        cond_num = mat_norm(matrix, cnt) * mat_norm(inv, cnt)
        print("\nЧисло обусловленности: %.4f" % cond_num)
        x_gauss = gauss(matrix, cnt)
        print("\nРешения, полученные с помощью метода Гаусса:")
        for i in range(0, cnt):
            print("x_%d = %.7f" % (i + 1, x_gauss[i]))
        x_gauss_m_el = gauss_diag(matrix, cnt)
        print("\nРешения, полученные с помощью"
              " метода Гаусса с выбором главного элемента:")
        for i in range(0, cnt):
            print("x_%d = %.7f" % (i + 1, x_gauss_m_el[i]))
    else:  # Если был выбран поиск решений СЛАУ
           # с помощью метода Релаксации
        choice_2 = 1
        herm_and_pos(matrix, cnt, 1)
        cond_num = mat_norm(matrix, cnt) * mat_norm(inverse(matrix, cnt), cnt)
        print("\nЧисло обусловленности: %.4f" % cond_num)
        while choice_2 == 1:  # Запускаем бесконечный цикл,
                              # завершающийся при желании пользователя
            try:
                eps = float(input("\nВведите положительную точность eps, "
                                  "с которой Вы хотите получить решения: "))
                if eps <= 0:
                    raise RuntimeError
            except:
                eps = -1
                while eps <= 0:
                    try:
                        eps = float(input("\nВведена неправильная точность."
                                          " Попробуйте еще раз:"))
                    except:
                        eps = -1
            try:
                w = float(input("\nВведите положительный итерационный параметр w\n "
                                "(для симметрической положительно определенной "
                                "матрицы системы следует выбирать 0<w < 2): "))
                if w <= 0:
                    raise RuntimeError
```

```
            except:
                w = -1
                while w <= 0:
                    try:
                        w = float(input("\nВведён неправильный параметр."
                                        " Попробуйте ещё раз:"))
                    except:
                        w = -1
            relax(matrix, cnt, eps, w)

            try:
                choice_2 = int(input("\n\nВведите 1, если хотите заново протестировать"
                                     " метод при других параметрах,0 - если "
                                     "хотите закончить тестирование:"))
                if (choice_2 != 0) and (choice_2 != 1):
                    raise RuntimeError
            except:
                choice_2 = -1
                while (choice_2 != 0) and (choice_2 != 1):
                    try:
                        choice_2 = int(input("\nВведен неправильныйномер."
                                             " Попробуйте еще раз: "))
                    except:
                        choice_2 = -1
        print("\n\nТестирование метода верхней релаксации закончено")
    try:
        choice = int(input("\n\nВведите 1, если хотите заново "
                           "запустить программу, 0 - если хотите выйти: "))
        if (choice != 0) and (choice != 1):
            raise RuntimeError
    except:
        choice = -1
        while (choice != 0) and (choice != 1):
            try:
                choice = int(input("\nВведён неправильный номер. Попробуйте ещё раз: "))
            except:
                choice = -1
print("\nВыход")

```

Список литературы

[1] Костомаров Д.П., Фаворский А.П. Вводные лекции по численным методам: Учеб. Пособие. – М.: Университетская книга, Логос, 2006

[2] Ильин В.А., Ким Г.Д. Линейная алгебра и аналитическая геометрия: учебник. 3-е изд., перераб., 2014

[3] Самарский АА. Введение в численные методы. Москва: Издательство «Наука», 1982