МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №7

по дисциплине «Вычислительная математика»

Тема: Исследование обусловленности задачи решения систем линейных уравнений

Студент гр. 0304 Крицын Д. Р.

Преподаватель Попова Е. В.

Санкт-Петербург

2021

Цель работы: Изучение стандартной обусловленности задач решения систем линейных уравнений при различных вариантах неточных входных данных.

Основные теоретические положения.

Рассматривается система линейных уравнений n-го порядка с вещественными коэффициентами (1)

$$a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1$$
$$a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=b_2$$
$$\ldots$$
$$a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n=b_n.$$

В матричной форме записи эта система принимает вид (2)

$$AX=B$$
$$A=\begin{bmatrix} a_{11}\ldots a_{1n} \\ \ldots \\ a_{n1}\ldots a_{nn} \end{bmatrix},\ X=\begin{bmatrix} x_1 \\ \ldots \\ x_n \end{bmatrix}, B=\begin{bmatrix} b_1 \\ \ldots \\ b_n \end{bmatrix}, \quad (2)$$

где A – квадратная матрица коэффициентов системы, X – вектор решений системы, B – вектор свободных членов. Матрица A – невырожденная, тогда решение системы (1) существует, единственно и устойчиво по входным данным. Это означает, что задача нахождения вектора X – корректна.

Пусть $X^*=\begin{bmatrix} x^*_1 \\ \ldots \\ x^*_n \end{bmatrix}$ – приближенное решение системы, тогда $e=X-X^*$ называется вектором погрешности системы, необходимо стремиться к его уменьшению. Возможно рассматривать критерий малости вектора $r=B-AX^*=B-B^*=A\left(X-X^*\right)$, который называется невязкой системы. Эти вектора связаны $e=X-X^*=A^{-1}r$.

Удобной количественной характеристикой вектора является норма вектора. В вычислительной математике используются следующие три нормы:

$$\|x\|_1=\sum_{i=1}^{n}|x_i|,\ \|x\|_2=\left(\sum_{i=1}^{n}x_i^2\right)^{\frac{1}{2}}\ \|x\|_\infty=max_{1\le i\le n}|x_i|. \quad (3)$$

За норму матрицы принимают максимальную величину, на которую преобразование, описываемое матрицей, может растянуть любой ненулевой вектор в выбранной норме $\|A\|=max_{x\neq 0}\frac{\|Ax\|}{\|x\|}$. Векторным нормам соответствуют следующие нормы матрицы:

$$\|A\|_1=max_{1\le j\le n}\sum_{i=1}^{n}|a_{ij}|, \quad (4)$$
$$\|A\|_2=max_{1\le j\le n}\sqrt{\lambda_i\left(A^TA\right)},$$

$$\|A\|_\infty = max_{1\le i\le n}\sum_{j=1}^{n}|a_{ij}|,$$

где λ_i – собственные числа матрицы A. Задача вычисления вектора X может быть плохо или хорошо обусловлена.

Обусловленность задачи решения систем линейных алгебраических уравнений.

Рассмотрим случай, когда элементы матрицы A заданы точно, а вектор-столбец свободных членов – приближенно. Оценки для абсолютной и относительной погрешности (5)

$$\Delta(X^*)\le \nu_\Delta \Delta(B^*),$$
$$\delta(X^*)\le \nu_\delta \delta(B^*),$$

где $\nu_\Delta = \|A^{-1}\|$- абсолютное число обусловленности, а $\nu_\delta = \|A^{-1}\|\frac{\|B\|}{\|X\|}$ — относительное число обусловленности (естественное число обусловленности). Максимальное естественное число обусловленности

$$max_{x\ne 0}\nu_\delta(X) = max_{x\ne 0}\|A^{-1}\|\frac{\|AX\|}{\|X\|} = \|A^{-1}\|\cdot\|A\| = cond(A) \quad (6)$$

называют стандартным числом обусловленности.

$$\delta(X^*)\le cond(A)\delta(B^*).$$

Если элементы матрицы A заданы приближенно и равны A^*, а вектор-столбец свободных членов – точно, тогда оценка относительной погрешности (7)

$$\delta(X^*)\le cond(A)\delta(A^*), \quad (7)$$

где $\delta(X^*) = \frac{\|X - X^*\|}{\|X\|}$ и $\delta(A^*) = \frac{\|A - A^*\|}{\|A\|}$.

Если с погрешностью заданы как коэффициенты матрицы, так и элементы вектора свободных членов, то справедливо неравенство

$$\delta(X^*)\le cond(A)\left(\delta(A^*)+\delta(B^*)\right). \quad (8)$$

Использование wxMaxima для подсчета обратной матрицы

Матрица A – невырожденная, следовательно существует единственная обратная матрица A^{-1}. Для ее подсчета используется пакет системы компьютерной алгебры wxMaxima. Входная матрица задаётся с помощью выражения **matrix**(*cmp1, cmp2, ... cmpN*), а обратная получается с помощью функции **invert**(*M*) (рисунок 1)

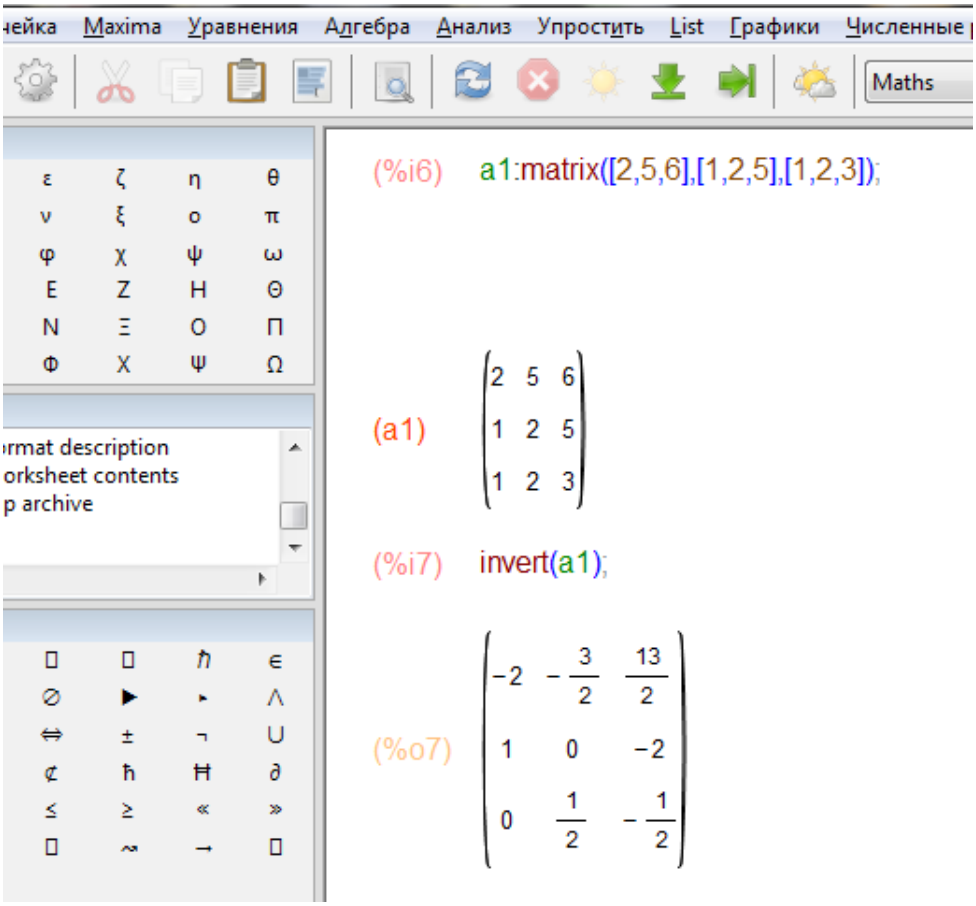

Рисунок 1 – Вычисление обратной матрицы с помощью функции invert

Порядок выполнения работы.

1 Составить подпрограмму для решения системы линейных уравнений методом Гаусса и методом обратной матрицы.

2 Решить систему, подсчитать стандартное число обусловленности, используя тип данных с двойной точностью. Подсчет обратной матрицы производить с помощью системы компьютерной алгебры wxMaxima.

3 С помощью встроенной функции – генератора случайных чисел, добавить ошибки в вектор свободных членов. Найти решение новой системы, стандартное число обусловленности (6) и оценку стандартного числа обусловленности (7).

4 С помощью встроенной функции – генератора случайных чисел, добавить ошибки в значения элементов матрицы. Найти решение новой системы, стандартное число обусловленности и оценку стандартного числа обусловленности.

5 Добавить ошибки в значения элементов матрицы и вектора свободных членов. Найти решение новой системы, стандартное число обусловленности и оценку стандартного числа обусловленности.

6 Сделать выводы по полученным значениям.

Выполнение работы.

Условие задания:

$$A=\begin{bmatrix}1 & 3 & -2\\ 3 & -1 & -7\\ 3 & 4 & 2\end{bmatrix}.$$

1. Составлена программа, решающая СЛУ методом Гаусса и методом обратной матрицы, которая также вносит возмущения в вектор свободных членов и матрицу A. Также для данной программы была составлена в дальнейшем используемая подпрограмма для получения обратной матрицы.

2. При помощи составленной программы была решена исходная система уравнений. Результаты приведены на Рисунке 1.

```
Эталонная СЛУ
1  3 -2 | -15
3 -1 -7 |   2
3  4  2 |   4
Вектор решений метода Гаусса: X = [6 -5 3]
Вектор решений метода обратной матрицы: X = [6 -5 3]

Стандартное число обусловленности: cond(A) = 2.63529
Абсолютное число обусловленности: v_Δ = 0.376471
Естественное число обсусловленности: v_δ = 0.564706
```

Рисунок 1 — результаты решения эталонной СЛУ

3. С помощью генератора случайных чисел были добавлены ошибки в вектор свободных членов B. При этом величина ошибки генерировалась таким образом, чтобы по модулю не превысить 0.2. Результаты работы программы приведены на Рисунке 2.

```
AX = B*
1  3 -2 | -14.8104
3 -1 -7 |  1.88286
3  4  2 |  4.09706
Вектор решений метода Гаусса: X = [5.94898 -4.9299 2.98486]
Вектор решений метода обратной матрицы: X = [5.94898 -4.9299 2.98486]

Стандартное число обусловленности: cond(A) = 2.63529
Абсолютное число обусловленности: v_Δ = 0.376471
Естественное число обсусловленности: v_δ = 0.564563
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / δ(B*) = 0.136254 / 0.403759 = 0.337465
```

Рисунок 2 — результаты решения СЛУ с внесением ошибок в вектор B

Решения методами Гаусса и обратной матрицы полностью совпали. Как и ожидалось, произошли небольшие изменения в значении естественного числа обусловленности: $\Delta\nu_\delta = -0{,}000143$.

При этом абсолютная погрешность вектора решений составила $\Delta X^* = \|X^* - X\| = 0{,}13626$.

4. Теперь с помощью генератора случайных чисел добавим ошибки в матрицу коэффициентов линейного уравнения A. Модуль величины ошибки всё так же не превышает 0.2. Результаты работы программы приведены на Рисунке 3.

```
A* X = B
0.948247   2.99621 -1.98941 | -15
 2.99395 -0.869252 -7.05331 |   2
 3.09652   4.06984   2.1837 |   4
Вектор решений метода Гаусса: X = [5.86126 -4.98938 2.81929]
Вектор решений метода обратной матрицы: X = [5.86126 -4.98938 2.81929]

Стандартное число обусловленности: cond(A) = 2.5191
Абсолютное число обусловленности: ν_Δ = 0.357892
Естественное число обсусловленности: ν_δ = 0.5498
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / δ(A*) = 0.330071 / 0.196797 = 1.67722
```

Рисунок 3 — результаты решения СЛУ с внесением ошибок в матрицу A

Векторы решений методов Гаусса и обратной матрицы снова совпали. Произошли небольшие изменения в стандартном, абсолютном и естественном числах обусловленности: $\Delta cond(A) = -0{,}11619$; $\Delta \nu_\Delta = -0{,}018579$; $\Delta \nu_\delta = -0{,}014906$.

Абсолютная погрешность вектора решений составила $\Delta X^* = \|X^* - X\| = 0{,}33007$.

5. С помощью генератора случайных чисел возмутим как матрицу коэффициентов, так и вектор свободных членов. Результаты работы программы приведены на Рисунке 4.

```
A* X = B*
0.965645  3.04363 -2.19351 | -14.8555
 2.89479 -1.04274 -6.88195 |  1.80343
 3.19996   3.8981  2.18872 |  4.11598
Вектор решений метода Гаусса: X = [5.15075 -4.63616 2.607]
Вектор решений метода обратной матрицы: X = [5.15075 -4.63616 2.607]

Стандартное число обусловленности: cond(A) = 2.44257
Абсолютное число обусловленности: ν_Δ = 0.345953
Естественное число обсусловленности: ν_δ = 0.579892
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / (δ(A*) + δ(B*)) = 1.60608 / (0.113268 + 0.457099) = 2.81588
```

Рисунок 4 — результаты решения СЛУ с внесением ошибок в матрицу A и вектор B

Видно, что произошли изменения такого же масштаба во всех трёх числах обусловленности: $\Delta cond(A) = -0{,}19272$; $\Delta \nu_\Delta = -0{,}030518$; $\Delta \nu_\delta = 0{,}015186$.

Абсолютная погрешность решения СЛУ составила $\Delta X^* = \|X^* - X\| = 1{,}60609$.

Вывод: наибольшее естественное число обусловленности из всех неточных вычислений $\nu_\delta = 0{,}579892$ было получено при одновременном изменении матрицы коэффициентов A и вектора свободных членов B, а наименьшее — при изменении лишь A: $\nu_\delta = 0{,}5498$. Наимеьшнее абсолютное число обусловленности было достигнуто (среди систем с матрицей коэффициентов A*) при одновременном возмущении A и B: $\nu_\Delta = 0{,}345953$, наибольшее — при возмущении лишь матрицы A: $\nu_\Delta = 0{,}357892$. При этом во всех случаях было верным неравенство $\nu_\delta < cond(A)$. При возмущении как матрицы A, так и вектора B абсолютная погрешность доистгает своей наибольшей абсолютной погрешности.

При этом только при возмущении как A, так и B, оценка стандартного числа обусловленности оказалась ложной (видимо, из-за больших результирующих погрешностей).

Исследование обусловленности задачи решения СЛУ с матрицей коэффициентов с двумя строками, заменёнными на строки матрицы Гильберта

6. Матрица Гильберта — квадратная матрица размера M x M, определённая следующим образом: $\Gamma = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \dots & \dots & \dots & \dots \\ a_{M1} & a_{M2} & \dots & a_{MM} \end{bmatrix}, a_{ij} = \frac{1}{i+j-1}.$

Таким образом, заменяя 1 и 3 строки в матрице A: $A^* = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 3 & -1 & -7 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}.$

7. Рассчитаем решение модифицированной СЛУ при помощи реализованной программы. Результаты предоставлены на Рисунке 5.

```
Эталонная СЛУ
       1  0.5 0.333333 | -15
       3   -1       -7 |   2
0.333333 0.25      0.2 |   4
Вектор решений метода Гаусса: X = [-80.8875 171.4 -59.4375]
Вектор решений метода обратной матрицы: X = [-80.8875 171.4 -59.4375]

Стандартное число обусловленности: cond(A) = 22.75
Абсолютное число обусловленности: ν_Δ = 5.25
Естественное число обсусловленности: ν_δ = 0.353677
```

Рисунок 5 — результаты решения СЛУ с 2мя Гильбертовыми строками

Видно, что векторы решений у обоих методов решения СЛУ совпадают.

8. С помощью генератора случайных чисел были добавлены ошибки в вектор свободных членов B. При этом величина ошибки будет сгенерирована таким образом, чтобы по модулю не превысить 0.2. Результаты работы программы приведены на Рисунке 6.

```
AX = B*
       1  0.5 0.333333 | -15.1298
       3   -1       -7 |  2.02729
0.333333 0.25      0.2 |  4.19112
Вектор решений метода Гаусса: X = [-82.7031 175.703 -60.8342]
Вектор решений метода обратной матрицы: X = [-82.7031 175.703 -60.8342]

Стандартное число обусловленности: cond(A) = 22.75
Абсолютное число обусловленности: ν_Δ = 5.25
Естественное число обсусловленности: ν_δ = 0.351079
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / δ(B*) = 7.51479 / 0.348245 = 21.5791
```

Рисунок 6 — результаты решения СЛУ с 2мя Гильбертовыми строками и возмущённым вектором B

Векторы решений, полученные методами Гаусса и обратной матрицы совпали. Произошли небольшие изменения в значении естественного числа обусловленности: $\Delta \nu_\delta = -0{,}002598$. Абсолютная погрешность решения СЛУ равна $\Delta X^* = \|X^* - X\| = 7{,}5153$.

9. С помощью генератора случайных чисел была возмущена матрица коэффициентов СЛУ A. При этом величина ошибки была подобрана по модулю меньше или равной 0.2. Результаты работы программы приведены на Рисунке 7.

```
A* X = B
0.843144  0.608947 0.210303 | -15
 2.94121 -0.901493 -7.04017 |   2
0.171596 0.0967543 0.356033 |   4
Вектор решений метода Гаусса: X = [22.7029 -61.9856 17.1378]
Вектор решений метода обратной матрицы: X = [22.7029 -61.9856 17.1378]

Стандартное число обусловленности: cond(A) = 5.89308
Абсолютное число обусловленности: v_Δ = 1.48968
Естественное число обсусловленности: v_δ = 0.307221
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / δ(A*) = 413.551 / 0.054208 = 7628.97
```

Рисунок 7 — результаты решения СЛУ с 2мя Гильбертовыми строками и возмущённой матрицей A

Векторы решений методов Гаусса и обратной матрицы снова совпали. Произошли существенные изменения в стандартном, абсолютном и естественном числах обусловленности: $\Delta cond(A) = -16{,}85692; \Delta \nu_\Delta = -3{,}76032; \Delta \nu_\delta = -0{,}046456$. Впрочем, естественное число обусловленности по-прежнему меньше стандартного числа обусловленности. Абсолютная погрешность вектора решения СЛУ составила $\Delta X^* = \|X^* - X\| = 289{,}5801$.

10. С помощью генератора случайных чисел были возмущены как матрица A, так и вектор B. Величина ошибки была подобрана по модулю меньше или равной 0.2. Результаты работы программы приведены на Рисунке 8.

```
A* X = B*
 1.04886 0.540708   0.420114 | -14.8143
 3.19555 -1.06278   -7.04338 |  2.19766
0.512582 0.352868 0.00025685 |  3.95751
Вектор решений метода Гаусса: X = [-36.7068 64.5556 -26.7065]
Вектор решений метода обратной матрицы: X = [-36.7068 64.5556 -26.7065]

Стандартное число обусловленности: cond(A) = 8.85618
Абсолютное число обусловленности: v_Δ = 1.86172
Естественное число обсусловленности: v_δ = 0.305068
Оценка стандартного числа обусловленности: cond(A) >= δ(X*) / (δ(A*) + δ(B*)) = 183.756 / (0.423652 + 0.425883) = 216.302
```

Рисунок 8 — результаты решения СЛУ с 2мя Гильбертовыми строками и возмущёнными матрицей A и вектором B

Векторы решений методов Гаусса и обратной матрицы снова совпали. Произошли серьёзные (но меньшие по величине по сравнению с предыдущим случаем) изменения в стандартном, абсолютном и естественном числах обусловленности: $\Delta cond(A) = -13{,}89382; \Delta \nu_\Delta = -3{,}38828; \Delta \nu_\delta = -0{,}048609$. Естественное число обусловленности остаётся меньше стандартного числа обусловленности. Погрешность решения СЛУ равна $\Delta X^* = \|X^* - X\| = 183{,}7561$.

Вывод: добавление строк Гильбертовой матрицы в матрицу A хоть и увеличило абсолютное число обусловленности, но естественное (относительное) число обусловленности стало меньше; кроме того, сама система уравнений стала куда более чувствительной к точно таким же по амплитуде возмущениям элементов матрицы A, показывая значительное уменьшение стандартного и абсолютного чисел обусловленности при внесении таких возмущений.

В данных результатах работы программы видно, что минимальное значение естественное число обусловленности принимает при возмущении A и B: $\nu_\delta = 0{,}305068$, максимальное — при возмущении лишь нетронутого заменой Гильбертовых строк вектора свободных членов B: $\nu_\delta = 0{,}351079$.

Минимальные стандартные и абсолютные числа обусловленности достигаются при возмущении лишь матрицы A: $\nu_\Delta = 1{,}48968, cond(A) = 5{,}89308$, а максимальные — при одновременном внесении возмущений в A и B: $\nu_\Delta = 1{,}86172, cond(A) = 8{,}85618$. При возмущении только матрицы A погрешность решения СЛУ достигает своего максимума, что отражает минимум стандартных и абсолютных чисел обусловленности при данном виде возмущений.

При этом из-за больших погрешностей в результатах решения СЛУ только при отсутствии возмущений в исходных данных СЛУ оценка стандартного числа обусловленности снизу была верной.

Выводы.

Была изучена обусловленность (в различных вариациях) задачи решения системы линейных уравнений при различных вариантах неточности входных данных (возмущениях на небольшие случайные значения матрицы коэффициентов системы A, вектора свободных членов B, и одновременно A и B).

Изменение элементов вектора свободных членов приводит к наименьшему изменению числа обусловленности ν_δ (и только его), в то время как возмущение матрицы коэффициентов СЛУ A приводит к наибольшему изменению всех чисел обусловленности. При одновременном изменении A и B изменение чисел обусловленности меньше, чем при лишь изменении A, но больше, чем при изменении B.

Данное наблюдение верно и для плохо обусловленной матрицы (т. е. матрицы, строки/столбцы которой заменены строками/столбцами плохо обусловленной матрицы Гильберта), стандартное число обусловленности для которой напорядок более чувствительно к возмущениям матрицы коэффициентов СЛУ, чем у исходной, эталонной матрицы.

Кроме того, оценка стандартного числа обусловленности зачастую верна для небольших возмущений в системе линейных уравнений.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <limits.h>

//-------
// Вектор
//-------

typedef struct{
     double* data;
     size_t dim;
} vec;


void vec_copy(vec* dest, const vec* src)
{
     dest->dim = src->dim;
     dest->data = malloc(src->dim * sizeof(double));
     for(size_t i = 0; i < src->dim; ++i)
           dest->data[i] = src->data[i];
}

void vec_cleanup(vec* v)
{
     for(size_t i = 0; i < v->dim; ++i)
           if(v->data[i] == 0)
                 v->data[i] = 0;
}


void vec_print(const vec* v)
{
     printf("[");
     for(size_t i = 0; i < v->dim; ++i)
           printf("%lg%s", v->data[i],
                       i == v->dim - 1 ? "]\n" : " ");
```

```
}


static inline void vec_add_vec(vec* v1, vec* v2)
{
     for(size_t i = 0; i < v1->dim; ++i)
          v1->data[i] += v2->data[i];
}
static inline void vec_sub_vec(vec* v1, vec* v2)
{
     for(size_t i = 0; i < v1->dim; ++i)
          v1->data[i] -= v2->data[i];
}
static inline void vec_mul_double(vec* v1, double a)
{
     for(size_t i = 0; i < v1->dim; ++i)
          v1->data[i] *= a;
}
static inline void vec_div_double(vec* v1, double a)
{
     for(size_t i = 0; i < v1->dim; ++i)
          v1->data[i] /= a;
}

// вычисление нормы вектора
double vec_norm1(const vec* v)
{
     double norm = 0;
     for(size_t i = 0; i < v->dim; ++i)
          norm += fabs(v->data[i]);
     return norm;
}

//--------------------
// Квадратная матрица
//--------------------

typedef struct{
     double* data;
     size_t dim;
} sq_matrix;
```

```
void sq_matrix_copy(sq_matrix* dest, const sq_matrix* src)
{
     size_t elems = src->dim * src->dim;
     dest->data = malloc(elems * sizeof(double));
     for(size_t i = 0; i < elems; ++i)
           dest->data[i] = src->data[i];
     dest->dim = src->dim;
}

void sq_matrix_alloc_ident(sq_matrix* dest, size_t dim)
{
     dest->dim = dim;
     dest->data = malloc(dim * dim * sizeof(double));
     for(size_t i = 0; i < dim; ++i)
           for(size_t j = 0; j < dim; ++j)
                 dest->data[i * dim + j] = (i == j) ? 1 : 0;
}

void sq_matrix_cleanup(sq_matrix* m)
{
     for(size_t i = 0; i < m->dim * m->dim; ++i)
           if(m->data[i] == 0)
                 m->data[i] = 0;
}


static inline vec sq_matrix_get_row(sq_matrix* m, size_t row)
{
     return (vec){.data = m->data + row * m->dim, .dim = m->dim};
}


void sq_matrix_print(const sq_matrix* m)
{
     char sbuf[64];
     int* max_w = malloc(m->dim * sizeof(int));
     for(size_t i = 0; i < m->dim; ++i)
           max_w[i] = 0;

     for(size_t i = 0; i < m->dim; ++i)
     {
           for(size_t j = 0; j < m->dim; ++j){
                 sprintf(sbuf, "%lg", m->data[i * m->dim + j]);
```

```
                int sym_count = (int)strlen(sbuf);
                if(sym_count > max_w[j])
                      max_w[j] = sym_count;
          }
     }

     for(size_t i = 0; i < m->dim; ++i)
     {
          for(size_t j = 0; j < m->dim; ++j)
                printf("%*lg ", max_w[j], m->data[i * m->dim + j]);
          puts("");
     }

     free(max_w);
}


// умножение матрицы на вектор-столбец
vec sq_matrix_mul_vec(sq_matrix* m, vec* v)
{
     vec res;
     res.data = malloc(sizeof(double) * v->dim);
     res.dim = v->dim;
     for(size_t i = 0; i < m->dim; ++i){
          res.data[i] = 0;
          for(size_t j = 0; j < m->dim; ++j)
                res.data[i] += m->data[i * m->dim + j] * v->data[j];
     }
     return res;
}

static inline void sq_matrix_sub_sq_matrix(sq_matrix* m1, const sq_matrix* m2)
{
     for(size_t i = 0; i < m1->dim * m1->dim; ++i)
          m1->data[i] -= m2->data[i];
}


// вычисление нормы матрицы
double sq_matrix_norm1(const sq_matrix* m)
{
     double norm = -INFINITY;
     for(size_t j = 0; j < m->dim; ++j)
```

```
	{
		double sum = 0;
		for(size_t i = 0; i < m->dim; ++i)
			sum += m->data[i * m->dim + j];
		if(sum > norm)
			norm = sum;
	}
	return norm;
}


void sq_matrix_inverse(sq_matrix* m1, int free_m1)
{
	// Прямой ход (получаем верхнетреугольную матрицу с единицами на диагонали в m1)
	sq_matrix m2;
	sq_matrix_alloc_ident(&m2, m1->dim);

	for(size_t j = 0; j < m1->dim; ++j)
	{
		// делим текущий ряд на опорный (диагональный) элемент
		vec v1 = sq_matrix_get_row(m1, j);
		vec v2 = sq_matrix_get_row(&m2, j);
		double base = v1.data[j];
		vec_div_double(&v1, base);
		vec_div_double(&v2, base);

		// отнимаем текущий ряд от всех более нижних рядов
		for(size_t i = j+1; i < m1->dim; ++i)
		{
			vec v1i = sq_matrix_get_row(m1, i); // i - текущий ряд
			vec v2i = sq_matrix_get_row(&m2, i);

			// "Выбиваем" ноль в столбце j
			double base = v1i.data[j];
			vec_mul_double(&v1, base); vec_mul_double(&v2, base);
			vec_sub_vec(&v1i, &v1); vec_sub_vec(&v2i, &v2);
			vec_div_double(&v1, base); vec_div_double(&v2, base);
		}
	}

	// Обратный ход (превращаем верхнетреугольную матрицу в диагональную
	for(size_t _j = 0; _j < m1->dim; ++_j)
	{
```

```
            // начинаем с последнего ряда (в котором стоит одна единица в последнем
столбце)
            size_t j = m1->dim - 1 - _j;

            vec v1 = sq_matrix_get_row(m1, j);
            vec v2 = sq_matrix_get_row(&m2, j);

            for(size_t _i = _j+1; _i < m1->dim; ++_i)
            {
                  size_t i = m1->dim - 1 - _i;

                  vec v1i = sq_matrix_get_row(m1, i);
                  vec v2i = sq_matrix_get_row(&m2, i);

                  double base = v1i.data[j];
                  vec_mul_double(&v1, base); vec_mul_double(&v2, base);
                  vec_sub_vec(&v1i, &v1); vec_sub_vec(&v2i, &v2);
                  vec_div_double(&v1, base); vec_div_double(&v2, base);
            }
      }

      if(free_m1)
            free(m1->data);
      sq_matrix_cleanup(&m2);
      m1->data = m2.data;
}

//------------------
// Система уравнений
//------------------

typedef struct{
      sq_matrix A;
      vec B;
} eq_sys;

void eq_sys_print(const eq_sys* sys)
{
      char sbuf[64];
      int* max_w = malloc((sys->A.dim + 1) * sizeof(int));
      for(size_t i = 0; i < sys->A.dim + 1; ++i)
            max_w[i] = 0;
```

```
	for(size_t i = 0; i < sys->A.dim; ++i)
	{
		for(size_t j = 0; j < sys->A.dim; ++j){
			sprintf(sbuf, "%lg", sys->A.data[i * sys->A.dim + j]);
			int sym_count = (int)strlen(sbuf);
			if(sym_count > max_w[j])
				max_w[j] = sym_count;
		}
	}
	for(size_t i = 0; i < sys->A.dim; ++i){
		sprintf(sbuf, "%lg", sys->B.data[i]);
		int sym_count = (int)strlen(sbuf);
		if(sym_count > max_w[sys->A.dim])
			max_w[sys->A.dim] = sym_count;
	}

	for(size_t i = 0; i < sys->A.dim; ++i)
	{
		for(size_t j = 0; j < sys->A.dim; ++j)
			printf("%*lg ", max_w[j], sys->A.data[i * sys->A.dim + j]);
		printf("| %*lg\n", max_w[sys->A.dim], sys->B.data[i]);
	}

	free(max_w);
}


vec eq_sys_solve_gauss(eq_sys* sys)
{
	// Прямой ход (получаем верхнетреугольную матрицу с единицами на диагонали)
	sq_matrix A;
	sq_matrix_copy(&A, &sys->A);
	vec B;
	vec_copy(&B, &sys->B);

	for(size_t j = 0; j < A.dim; ++j)
	{
		// делим текущий ряд на опорный (диагональный) элемент
		vec v1 = sq_matrix_get_row(&A, j);
		B.data[j] /= v1.data[j]; vec_div_double(&v1, v1.data[j]);

		// отнимаем текущий ряд от всех более нижних рядов
		for(size_t i = j+1; i < A.dim; ++i)
```

```
		{
			vec v2 = sq_matrix_get_row(&A, i); // i - текущий ряд

			// "Выбиваем" ноль в столбце j
			double base = v2.data[j];
			vec_mul_double(&v1, base);
			vec_sub_vec(&v2, &v1); B.data[i] -= B.data[j] * base;
			vec_div_double(&v1, base);
		}
	}

	// Обратный ход (превращаем верхнетреугольную матрицу в диагональную
	for(size_t _j = 0; _j < A.dim; ++_j)
	{
		// начинаем с последнего ряда (в котором стоит одна единица в последнем
столбце)
		size_t j = A.dim - 1 - _j;

		for(size_t _i = _j+1; _i < A.dim; ++_i)
		{
			size_t i = A.dim - 1 - _i;

			vec v2 = sq_matrix_get_row(&A, i);

			double base = v2.data[j];
			v2.data[j] = 0; // "отнимаем" текущий ряд
			B.data[i] -= B.data[j] * base;
		}
	}

	// очищаем выделенную память
	free(A.data);
	// выводим результат в вектор X
	vec X = (vec){.data = B.data, .dim = B.dim};
	vec_cleanup(&X);
	return X;
}

vec eq_sys_solve_inverse(eq_sys* sys)
{
	// находим A^{-1}
	sq_matrix inv_A;
	sq_matrix_copy(&inv_A, &sys->A);
```

```
    sq_matrix_inverse(&inv_A, 1);

    // умножаем A^{-1} на b, находя X
    vec X = sq_matrix_mul_vec(&inv_A, &sys->B);
    vec_cleanup(&X);
    free(inv_A.data);
    return X;
}


#define TCLR_RED_B "\033[1;31m"
#define TCLR_RESET "\033[0m"
#define TCLR_GREEN_B "\033[1;32m"

#define EXECUTE_TWICE(action, presecond)\
{ action; }\
{ presecond; }\
{ action; }

int main()
{
    static const size_t rand_seed = 1637509548;
    static const double rand_distr = 0.2; // погрешности генерируются в интервале [-
rand_distr; rand_distr]
    srand(rand_seed);
    printf("Зерно ГСЧ: %lu\n\n", rand_seed);

    eq_sys esys;
    esys.A = (sq_matrix){.data = (double[]){1,  3, -2,
                                    3, -1, -7,
                                    3,  4,  2},
             .dim = 3};
    esys.B = (vec){.data = (double[]){-15, 2, 4}, .dim = 3};

    EXECUTE_TWICE(
    printf("%sЭталонная СЛУ%s\n", TCLR_RED_B, TCLR_RESET);
    {
          eq_sys_print(&esys);

          vec X1 = eq_sys_solve_gauss(&esys);
          vec X2 = eq_sys_solve_inverse(&esys);

          printf("Вектор решений метода Гаусса: X = "); vec_print(&X1);
```

```
        printf("Вектор решений метода обратной матрицы: X = "); vec_print(&X2);

        sq_matrix A_inv;
        sq_matrix_copy(&A_inv, &esys.A);
        sq_matrix_inverse(&A_inv, 1);

        double cond_A = sq_matrix_norm1(&A_inv) * sq_matrix_norm1(&esys.A);
        printf("\nСтандартное число обусловленности: cond(A) = %lg\n", cond_A);
        double abs_cond_A = sq_matrix_norm1(&A_inv);
        printf("Абсолютное число обусловленности: ν_Δ = %lg\n", abs_cond_A);
        double nat_cond_A = sq_matrix_norm1(&A_inv) * vec_norm1(&esys.B) /
vec_norm1(&X1);
        printf("Естественное число обсусловленности: ν_δ = %lg\n", nat_cond_A);

        free(A_inv.data);
    }

    printf("\n%sAX = B*%s\n", TCLR_RED_B, TCLR_RESET);
    {
        eq_sys esys1;
        sq_matrix_copy(&esys1.A, &esys.A);
        vec_copy(&esys1.B, &esys.B);

        for(size_t i = 0; i < esys1.B.dim; ++i)
            esys1.B.data[i] += ((double)rand() / RAND_MAX) * rand_distr * 2 -
rand_distr;

        eq_sys_print(&esys1);

        vec X = eq_sys_solve_gauss(&esys);
        vec X1 = eq_sys_solve_gauss(&esys1);
        vec X2 = eq_sys_solve_inverse(&esys1);

        printf("Вектор решений метода Гаусса: X = "); vec_print(&X1);
        printf("Вектор решений метода обратной матрицы: X = "); vec_print(&X2);

        sq_matrix A_inv;
        sq_matrix_copy(&A_inv, &esys1.A);
        sq_matrix_inverse(&A_inv, 1);

        double max_cond_A = sq_matrix_norm1(&A_inv) * sq_matrix_norm1(&esys1.A);
        printf("\nСтандартное число обусловленности: cond(A) = %lg\n",
max_cond_A);
```

```
            double abs_cond_A = sq_matrix_norm1(&A_inv);
            printf("Абсолютное число обусловленности: ν_Δ = %lg\n", abs_cond_A);
            double  nat_cond_A  =  sq_matrix_norm1(&A_inv)  *  vec_norm1(&esys1.B)  /
vec_norm1(&X1);
            printf("Естественное число обсусловленности: ν_δ = %lg\n", nat_cond_A);

            vec delta_X; vec_copy(&delta_X, &X1); vec_sub_vec(&delta_X, &X);
            vec delta_B; vec_copy(&delta_B, &esys1.B); vec_sub_vec(&delta_B, &esys.B);
            double cond_estimate_A = vec_norm1(&delta_X) / vec_norm1(&delta_B);
            printf("Оценка  стандартного  числа  обусловленности:  cond(A)  >=  δ(X*)  /
δ(B*) = %lg / %lg = %lg\n",
                  vec_norm1(&delta_X), vec_norm1(&delta_B), cond_estimate_A);

            free(A_inv.data);
            free(delta_X.data);
            free(delta_B.data);
      }

      printf("\n%sA* X = B%s\n", TCLR_RED_B, TCLR_RESET);
      {
            eq_sys esys1;
            sq_matrix_copy(&esys1.A, &esys.A);
            vec_copy(&esys1.B, &esys.B);

            for(size_t i = 0; i < esys1.A.dim * esys.A.dim; ++i)
                  esys1.A.data[i]  +=  ((double)rand()  /  RAND_MAX)  *  rand_distr  *  2  -
rand_distr;

            eq_sys_print(&esys1);

            vec X = eq_sys_solve_gauss(&esys);
            vec X1 = eq_sys_solve_gauss(&esys1);
            vec X2 = eq_sys_solve_inverse(&esys1);

            printf("Вектор решений метода Гаусса: X = "); vec_print(&X1);
            printf("Вектор решений метода обратной матрицы: X = "); vec_print(&X2);

            sq_matrix A_inv;
            sq_matrix_copy(&A_inv, &esys1.A);
            sq_matrix_inverse(&A_inv, 1);

            double max_cond_A = sq_matrix_norm1(&A_inv) * sq_matrix_norm1(&esys1.A);
```

```
          printf("\nСтандартное число обусловленности: cond(A) = %lg\n",
max_cond_A);
          double abs_cond_A = sq_matrix_norm1(&A_inv);
          printf("Абсолютное число обусловленности: ν_Δ = %lg\n", abs_cond_A);
          double nat_cond_A = sq_matrix_norm1(&A_inv) * vec_norm1(&esys1.B) /
vec_norm1(&X1);
          printf("Естественное число обсусловленности: ν_δ = %lg\n", nat_cond_A);

          vec delta_X; vec_copy(&delta_X, &X1); vec_sub_vec(&delta_X, &X);
          sq_matrix delta_A; sq_matrix_copy(&delta_A, &esys1.A);
sq_matrix_sub_sq_matrix(&delta_A, &esys.A);
          double cond_estimate_A = vec_norm1(&delta_X) / sq_matrix_norm1(&delta_A);
          printf("Оценка стандартного числа обусловленности: cond(A) >= δ(X*) /
δ(A*) = %lg / %lg = %lg\n",
                vec_norm1(&delta_X), sq_matrix_norm1(&delta_A), cond_estimate_A);

          free(A_inv.data);
          free(delta_X.data);
          free(delta_A.data);
     }

     printf("\n%sA* X = B*%s\n", TCLR_RED_B, TCLR_RESET);
     {
          eq_sys esys1;
          sq_matrix_copy(&esys1.A, &esys.A);
          vec_copy(&esys1.B, &esys.B);

          for(size_t i = 0; i < esys1.A.dim * esys.A.dim; ++i)
               esys1.A.data[i] += ((double)rand() / RAND_MAX) * rand_distr * 2 -
rand_distr;
          for(size_t i = 0; i < esys1.B.dim; ++i)
               esys1.B.data[i] += ((double)rand() / RAND_MAX) * rand_distr * 2 -
rand_distr;

          eq_sys_print(&esys1);

          vec X = eq_sys_solve_gauss(&esys);
          vec X1 = eq_sys_solve_gauss(&esys1);
          vec X2 = eq_sys_solve_inverse(&esys1);

          printf("Вектор решений метода Гаусса: X = "); vec_print(&X1);
          printf("Вектор решений метода обратной матрицы: X = "); vec_print(&X2);
```

```
            sq_matrix A_inv;
            sq_matrix_copy(&A_inv, &esys1.A);
            sq_matrix_inverse(&A_inv, 1);

            double max_cond_A = sq_matrix_norm1(&A_inv) * sq_matrix_norm1(&esys1.A);
            printf("\nСтандартное число обусловленности: cond(A) = %lg\n",
max_cond_A);
            double abs_cond_A = sq_matrix_norm1(&A_inv);
            printf("Абсолютное число обусловленности: ν_Δ = %lg\n", abs_cond_A);
            double nat_cond_A = sq_matrix_norm1(&A_inv) * vec_norm1(&esys1.B) /
vec_norm1(&X1);
            printf("Естественное число обсусловленности: ν_δ = %lg\n", nat_cond_A);

            vec delta_X; vec_copy(&delta_X, &X1); vec_sub_vec(&delta_X, &X);
            sq_matrix delta_A; sq_matrix_copy(&delta_A, &esys1.A);
sq_matrix_sub_sq_matrix(&delta_A, &esys.A);
            vec delta_B; vec_copy(&delta_B, &esys1.B); vec_sub_vec(&delta_B, &esys.B);
            double cond_estimate_A = vec_norm1(&delta_X) / (sq_matrix_norm1(&delta_A)
+ vec_norm1(&delta_B));
            printf("Оценка стандартного числа обусловленности: cond(A) >= δ(X*) /
(δ(A*) + δ(B*)) = %lg / (%lg + %lg) = %lg\n",
                  vec_norm1(&delta_X), sq_matrix_norm1(&delta_A), vec_norm1(&delta_B),
cond_estimate_A);

            free(A_inv.data);
            free(delta_A.data);
            free(delta_B.data);
      }
      ,

      printf("\n\n%sЗаменим 1 и 3 строки на строки Гильбертовой матрицы:%s\n\n",
TCLR_GREEN_B, TCLR_RESET);
      size_t i = 0;
      vec v = sq_matrix_get_row(&esys.A, i);
      for(size_t j = 0; j < v.dim; ++j)
            v.data[j] = 1. / ((i+1) + (j+1) - 1.);
      i = 2;
      v = sq_matrix_get_row(&esys.A, i);
      for(size_t j = 0; j < v.dim; ++j)
            v.data[j] = 1. / ((i+1) + (j+1) - 1.);

      );
}
```